MINTRL: OFF-POLICY INTERVENTION CAN BOOST ON-POLICY RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards is typically performed on-policy, keeping training data close to the current policy but limiting learning to trajectories that the policy can discover itself. Off-policy methods such as supervised fine-tuning, on the other hand, can leverage external knowledge beyond the base model's capabilities, but may suffer from large distribution shift. The key challenge is thus to expand exploration without sacrificing learnability. In this work, we introduce *Minimal Intervention Reinforcement Learning* (MInTRL), which expands the exploration frontier through sparse, local interventions in otherwise on-policy rollouts. During generation, a judge–intervention policy periodically reviews the current policy's output, replaces erroneous suffixes with short corrections, and immediately returns control to the policy. During training, MInTRL adopts a sequence-level advantage-regression objective that eliminates the need for importance sampling. We show that sparse, local interventions can substantially improve coverage beyond finite-budget on-policy sampling while preserving the overall on-policy nature of the resulting trajectories. Across math and code benchmarks, MInTRL consistently outperforms standard on-policy and off-policy baselines. Ablations show that MInTRL remains effective with self-intervention and across different judge policies, while performance peaks at moderate intervention intensity, highlighting the importance of intervening minimally. These results establish minimal intervention as an effective paradigm for enhancing on-policy RL.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has become a key component of post-training large language models (LLMs) for advanced reasoning (Guo et al., 2025; Team et al., 2025b; Yang et al., 2025). By learning from self-generated experience with automatically verifiable outcome rewards, large-scale RL post-training has produced substantial improvements in mathematical reasoning, code generation, and other challenging reasoning tasks.

Most existing RLVR methods are built around *on-policy* learning, where the policy is optimized using rollouts sampled from the current policy. While this design avoids behavior-policy mismatch, it also restricts the experience available for learning to what the policy can discover through its own sampling. Recent studies suggest that on-policy RLVR often struggles to discover reasoning trajectories beyond those already reachable by the base model under finite sampling, limiting its ability to expand the model's reasoning capabilities (Chen et al., 2026b; Wu et al., 2025).

External knowledge can expose the policy to successful reasoning paths that are rarely reached through its own rollouts. However, incorporating this knowledge during RL introduces off-policy information, and the resulting trajectories may differ substantially from the current policy's own behavior. The challenge is therefore to expand exploration while preserving the policy's ability to learn from this additional experience. This raises a fundamental question: **can LLMs effectively incorporate external knowledge during RL training? If so, how much off-policy information should be introduced?**

To address this challenge, we propose *Minimal Intervention Reinforcement Learning* (MInTRL), which incorporates external knowledge through sparse local interventions while keeping most of each training trajectory on-policy. Figure 1 illustrates the overall pipeline. During rollouts, the current policy remains in control by default, while a judge–intervention policy provides short local corrections

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.

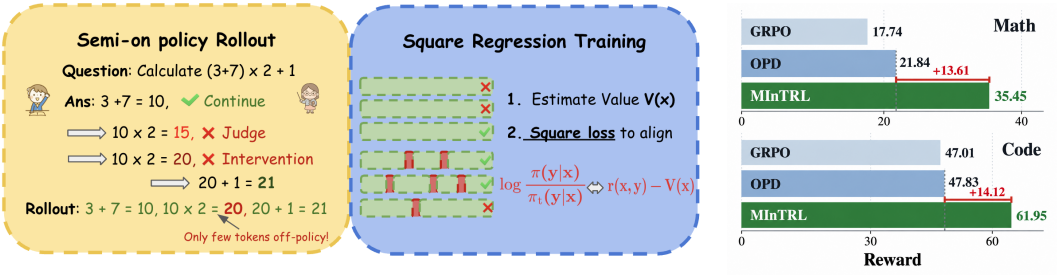


Figure 1: Overview and main results of MInTRL.

when errors are detected before returning control to the current policy. These interventions help the policy reach successful reasoning paths that are otherwise unlikely to emerge from its own rollouts, expanding the experience available for learning with only a small fraction of off-policy tokens. To effectively learn from these mixed-provenance trajectories, we further adopt a regression-based RL objective that avoids behavior-policy importance weighting. Our contributions are threefold:

- **Method.** We introduce MInTRL, a semi-on-policy RL framework that uses minimal intervention to expose the current policy to otherwise difficult-to-discover reasoning paths, together with a regression objective that learns from mixed-provenance trajectories without behavior-policy importance weighting.
- **Theory.** We provide a stylized support analysis illustrating a coverage–learnability trade-off under sparse optimal corrections: sparse off-policy interventions can expand the set of successful trajectories visible under finite sampling, while excessive intervention can push those trajectories beyond what the current policy can reliably learn from.
- **Experiments.** Across mathematical reasoning and code generation with Qwen3-1.7B and Qwen3-4B, MInTRL outperforms on-policy RL, distillation, and intervention-based baselines by up to 9.44 percentage points over the strongest competitor.

2 Method

We now present MInTRL, a framework that incorporates external guidance through minimal intervention while keeping the resulting training experience predominantly on-policy. At each step t, MInTRL consists of two stages. During *semi-on-policy rollout sampling*, trajectories are generated predominantly by the current policy π_t, with a judge–intervention policy supplying only a small number of corrective tokens. During *regression-based RL training*, we optimize the policy over the collected semi-on-policy rollouts using a sequence-level regression objective. We next describe the two stages in detail.

2.1 Semi-on-policy Rollout Sampling

We first describe the *semi-on-policy rollout sampling* procedure. Let $x \sim \mathcal{D}$ denote an input prompt, π_t the current policy at iteration t, and $y = (y_1, \ldots, y_T)$ a response with verifiable outcome reward $r(x, y)$. Standard on-policy RLVR generates the entire response from π_t. In contrast, our procedure keeps π_t in control of generation while allowing a judge–intervention policy π_{JI} to contribute a small number of corrective tokens when needed. The judge–intervention policy can be either a stronger teacher model or the same policy model with privileged context (Zhao et al., 2026; Shenfeld et al., 2026). Algorithm 1 summarizes the procedure.

Algorithm 1 Semi-on-policy Rollout Sampling

Require: Input x; policies π_t and π_{JI}
1: Initialize the accepted prefix $p \leftarrow \varnothing$.
2: **while** the response is incomplete **do**
3: Sample $c \sim \pi_t(\cdot \mid x, p)$.
4: π_{JI} reviews c and returns KEEP or REVISE.
5: **KEEP:** update $p \leftarrow p \| c$.
6: **REVISE:** sample $h \sim \pi_{\text{JI}}(\cdot \mid x, p \| c_{<e})$.
7: Update $p \leftarrow p \| c_{<e} \| h$ and resume with π_t.
8: **return** p.

Specifically, for each rollout, π_t generates the response incrementally in short chunks, with each newly generated chunk reviewed by π_{JI} in the context of the accepted prefix so far. For example, the first chunk is sampled as

$$c_1 = (y_1, \ldots, y_k) \sim \pi_t(\cdot \mid x). \tag{1}$$

The judge-intervention policy π_{JI} then reviews c_1 together with the input x, deciding whether to keep the chunk or revise it and, in the latter case, identifying the earliest erroneous token:

$$(a_1, e_1) = \mathcal{J}_{\text{JI}}(x, c_1), \qquad a_1 \in \{\textsc{Keep}, \textsc{Revise}\}. \tag{2}$$

If $a_1 = \textsc{Keep}$, the entire chunk c_1 is retained and appended to the accepted prefix. If $a_1 = \textsc{Revise}$, we truncate c_1 at the detected error position e_1 and preserve only the prefix $(y_1, \ldots, y_{e_1-1})$. The intervention policy π_{JI} then generates a short corrective continuation $h_1 \sim \pi_{\text{JI}}(\cdot \mid x, y_1, \ldots, y_{e_1-1})$ from the retained prefix. We then append h_1 to the retained prefix, yielding $(y_1, \ldots, y_{e_1-1}) \| h_1$. After either keeping the original chunk or forming the corrected prefix, control returns to π_t, which resumes generation from the resulting prefix. We repeat this generate–review–intervene process until the response is completed.

The key property of this construction is its locality. The judge-intervention policy generates only a short corrective span, after which the current policy immediately resumes generation. Consequently, the resulting trajectory incorporates external information at the point of error while remaining predominantly generated by π_t.

2.2 Learning from the Collected Rollouts

Given the semi-on-policy rollouts generated above, the next question is how to optimize the policy with these trajectories. We use the rule-based reward as the learning signal, rather than directly distilling the intervention policy, since the intervention is only intended to steer generation and its local corrections may not always provide reliable supervision. A natural first choice is to apply a standard GRPO-style objective. However, for these mixed-policy trajectories, doing so requires importance-sampling corrections to account for the behavior-policy mismatch. Even when interventions affect only a small fraction of tokens, importance ratios can accumulate across the trajectory and become highly unstable. To avoid this issue, we adopt a *regression-based RL objective* that learns directly from rewarded semi-on-policy trajectories without explicitly dividing by their behavior probabilities.

Regression target. Let π_t denote the current policy before the update. We begin from the standard KL-regularized RL objective

$$\max_{\pi} \; \mathbb{E}_{y \sim \pi(\cdot|x)}[r(x, y)] - \beta D_{\text{KL}}(\pi(\cdot \mid x) \,\|\, \pi_t(\cdot \mid x)) \,. \tag{3}$$

The optimal policy and its normalizing value have the closed forms

$$V^\star_\beta(x) = \beta \log \mathbb{E}_{y \sim \pi_t(\cdot|x)} \left[\exp\left(r(x, y)/\beta\right)\right], \tag{4}$$

$$\pi^\star_\beta(y \mid x) = \pi_t(y \mid x) \exp\left(\big(r(x, y) - V^\star_\beta(x)\big)/\beta\right). \tag{5}$$

Equivalently, their pointwise optimality condition is

$$\beta \log \frac{\pi^\star_\beta(y \mid x)}{\pi_t(y \mid x)} = r(x, y) - V^\star_\beta(x) \equiv A^\star_\beta(x, y). \tag{6}$$

Thus, $A^\star_\beta(x, y)$ directly specifies how the likelihood of each trajectory should change relative to π_t. Following squared advantage-regression methods (Team et al., 2025b; Brantley et al., 2026; Team et al., 2025a; Ritter et al., 2026), we fit this condition directly. The corresponding regression objective is

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y)\in\mathcal{D}} \left[\beta \log \frac{\pi_\theta(y \mid x)}{\pi_t(y \mid x)} - A^\star_\beta(x, y)\right]^2. \tag{7}$$

Crucially, the regression target in equation 7 is well-defined for any trajectory y with $\pi_t(y \mid x) > 0$, regardless of the behavior policy that generated it. Thus, semi-on-policy trajectories can be used directly as regression examples without behavior-policy importance weighting; the sampling distribution only determines which trajectories are observed.

2.3 PRACTICAL IMPLEMENTATION

Step-level intervention localization. In practice, reliably identifying the exact first erroneous token is impossible. We therefore perform intervention at the level of reasoning or code steps rather than individual tokens. Specifically, each policy-generated chunk is first segmented into steps using double-newline delimiters (`\n\n`) or code-fence boundaries. The judge then reviews the resulting sequence of steps and identifies the earliest incorrect step. The judge and intervention prompts are provided in Appendix C. We retain all steps preceding this point, discard the remaining suffix, and invoke π_{JI} to generate a short corrective continuation from the retained prefix.

Control-based advantage estimation. Computing the soft value $V^{\star}_{\beta}(x)$ in equation 4 exactly requires marginalizing over all possible responses. In practice, for each prompt x, we collect N_m semi-on-policy rollouts using Alg 1 together with N_c pure on-policy control rollouts from the current policy π_t,

$$y_j^{\mathrm{ctrl}} \sim \pi_t(\cdot \mid x), \qquad j = 1, \ldots, N_c. \tag{8}$$

The control rollouts are used to estimate the value baseline, while both control and semi-on-policy rollouts are included as regression examples. To decouple value estimation from the strength of policy regularization, we use two temperatures: β_1 for the value target and β_2 for policy regression, following prior advantage-regression formulations (Brantley et al., 2026; Ritter et al., 2026). The resulting Monte Carlo estimate of the soft value is

$$\widehat{V}_{\beta_1}(x) = \beta_1 \log \left[\frac{1}{N_c} \sum_{j=1}^{N_c} \exp\left(r(x, y_j^{\mathrm{ctrl}}) / \beta_1 \right) \right]. \tag{9}$$

The temperature β_1 controls how the value target aggregates rewards: as $\beta_1 \to 0$, the soft value approaches the maximum observed reward, whereas as $\beta_1 \to \infty$, it approaches the empirical mean reward. We adopt the latter regime because the mean-reward baseline has been found effective in practice (Team et al., 2025b), and choosing $\beta_1 \gg \beta_2$ leads to a more conservative regression target that is less sensitive to a small number of high-reward off-policy trajectories (Sakhi et al., 2026). Accordingly, we take the large-β_1 limit and use

$$\widehat{V}_{\mathrm{ctrl}}(x) = \frac{1}{N_c} \sum_{j=1}^{N_c} r(x, y_j^{\mathrm{ctrl}}), \qquad \widehat{A}(x, y) = r(x, y) - \widehat{V}_{\mathrm{ctrl}}(x). \tag{10}$$

Importantly, we estimate the baseline exclusively from on-policy control rollouts, while using both control and semi-on-policy rollouts as regression examples. The same rule-based reward is applied to both groups.

A relaxed anchor for intervention tokens. The regression objective in equation 7 anchors the likelihood of each trajectory to the current policy π_t. For intervention-authored tokens, however, π_t may assign extremely small probabilities, which can make the original reference overly restrictive. To relax this reference, we consider using their log probabilities under the intervention policy π_{JI}. These log probabilities may be unavailable when π_{JI} is accessed as a black-box model. We therefore anchor intervention tokens with a constant $\kappa < 0$:

$$\log \widetilde{\pi}_t(y_s \mid x, y_{<s}) = \begin{cases} \log \pi_t(y_s \mid x, y_{<s}), & y_s \text{ is policy-authored}, \\ \kappa, & y_s \text{ is intervention-authored}. \end{cases} \tag{11}$$

Intuitively, κ can be viewed as a shared reference level motivated by the intervention policy's expected per-token log probability, $\mathbb{E}_v[\log \pi_{\mathrm{JI}}(v \mid x, y_{<s})]$. This interpretation is heuristic: κ remains a tunable reference value, and we do not assume that it accurately estimates this expectation. The anchor modification enters the sequence-level reference score additively in log space and is confined to sparse intervention spans, while policy-authored tokens remain anchored to π_t. Accordingly, we use it as a practical anchoring device rather than interpreting it as a standalone normalized policy.

Early stopping. In practice, the judge-intervention policy π_{JI} is itself imperfect, and the corrections it provides may occasionally be erroneous. Nevertheless, during the early stage of training, these interventions can still provide useful guidance by helping the current policy reach successful

trajectories that it would otherwise struggle to discover through on-policy sampling alone. As the policy improves, the marginal benefit of intervention diminishes, while erroneous or unnecessary corrections continue to introduce harmful off-policy information. We therefore use intervention only during an initial phase of training. After a preset intervention phase, we disable π_{JI} and continue training with standard on-policy RL. This concentrates external guidance in the early stage, where it is most useful, while keeping the later stage fully on-policy.

3 Theoretical Perspective

We formalize MInTRL as a sparse perturbation of the current policy: its rollout distribution μ_N follows π_t except at most N corrected token positions. This turns our problem into analyzing how sparse corrections expand the support of correct rollouts. Under a finite rollout budget, on-policy training behaves as though the population reward were truncated to empirically visible support. We first characterize this objective gap and then bound how far MInTRL can expand support while remaining learnable. This perspective is related to empirical-support analyses of RLVR and classical coverage conditions in reinforcement learning (Wu et al., 2025; Xie et al., 2022).

Definition 3.1 (ϵ-support of correct rollouts)**.** *For the binary verifier reward considered in this section, let $\mathcal{C}(x) = \{y \in \mathcal{Y} : r(x, y) = 1\}$ denote the set of correct rollouts for prompt x. For any threshold $\epsilon \in (0, 1)$, the ϵ-support of correct rollouts under policy π is*

$$\mathrm{supp}_{\epsilon}(\pi; x) = \{y \in \mathcal{C}(x) : \pi(y \mid x) > \epsilon\}. \tag{12}$$

Finite-budget visibility and trainability need not share the same likelihood threshold. We capture this distinction as follows.

Assumption 3.2 (Inference and training support)**.** *There exist thresholds $\epsilon_{\mathrm{train}} < \epsilon_{\mathrm{inf}}$ such that $\mathrm{supp}_{\epsilon_{\mathrm{inf}}}(\pi_t; x)$ denotes the rollouts that can be empirically sampled under finite-budget inference, while $\mathrm{supp}_{\epsilon_{\mathrm{train}}}(\pi_t; x)$ denotes the rollouts from which RL training can still learn.*

The ordering $\epsilon_{\mathrm{train}} < \epsilon_{\mathrm{inf}}$ is reasonable: finite-budget sampling only reveals an empirically visible subset of correct rollouts (Wu et al., 2025; Chen et al., 2026b), while asynchronous and stale-rollout studies show that data can remain useful for training after becoming off-policy (Hou et al., 2026a; Zheng et al., 2026). We first characterize the optimization limit imposed by the smaller inference support.

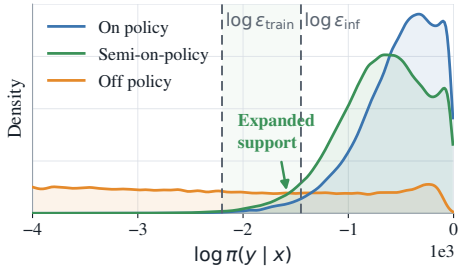


Figure 2: **Empirical support expansion.** The dashed thresholds visualize Assumption 3.2. MInTRL adds mass beyond inference support while remaining within training support, illustrating Theorem 3.5; off-policy rollouts extend largely beyond the trainable region.

Lemma 3.3 (Finite-rollout objective gap)**.** *Under Assumption 3.2, the empirical RL objective at iteration t is*

$$\mathbb{E}_{y \sim \pi(\cdot|x)}\left[r(x, y)\, \mathbb{I}\{y \in \mathrm{supp}_{\epsilon_{\mathrm{inf}}}(\pi_t; x)\}\right]. \tag{13}$$

In this regard, the objective differs from the population objective $\mathbb{E}_{y \sim \pi(\cdot|x)}[r(x, y)]$: the optimality gap is one when the inference support is empty and zero otherwise.

The proof is provided in Appendix B.1. Lemma 3.3 shows why a larger inference support helps: it makes more correct rollouts visible under finite-budget sampling, reducing the gap between the empirical and population objectives. We next quantify how far sparse optimal corrections can expand this support under a mild token-level reachability assumption.

Assumption 3.4 (Optimal-token reachability)**.** *There exists $\nu > 0$ such that $\pi(y_t^\star \mid x, y_{<t}) \geq \nu$ for every policy $\pi \in \Pi$, prompt x, prefix $y_{<t}$, and optimal next token $y_t^\star$.*

This is a simplifying reachability assumption used to obtain a uniform support envelope. Under this assumption, the support expansion of MInTRL is characterized by the following envelope.

Theorem 3.5 (Support envelope of sparse optimal corrections)**.** *Fix the current policy π_t and a prompt x. Let μ_N follow π_t except at a set of at most N positions, where it emits an optimal next token $y_t^\star$. The visible correct support of the combined current-policy and corrected rollout pool is*

$$\mathcal{S}_{\epsilon_{\mathrm{inf}}, N}(\pi_t, \mu_N; x) = \mathrm{supp}_{\epsilon_{\mathrm{inf}}}(\pi_t; x) \cup \mathrm{supp}_{\epsilon_{\mathrm{inf}}}(\mu_N; x). \tag{14}$$

Under Assumption 3.4,

$$\mathrm{supp}_{\epsilon_{\mathrm{inf}}}(\pi_t; x) \subseteq \mathcal{S}_{\epsilon_{\mathrm{inf}},N}(\pi_t, \mu_N; x) \subseteq \mathrm{supp}_{\epsilon_{\mathrm{inf}}\nu^N}(\pi_t; x). \tag{15}$$

The proof is provided in Appendix B.2.

The left inclusion preserves the current policy's visible support, while the right inclusion shows that N corrections can expose rollouts up to a factor of $(1/\nu)^N$ less likely under π_t. This expansion can provide the positive signal missing in Lemma 3.3. However, the theorem guarantees that corrected rollouts remain within training support only while $\epsilon_{\mathrm{inf}}\nu^N \geq \epsilon_{\mathrm{train}}$. Beyond this range, the expanded support may include rollouts from which the policy can no longer reliably learn. Thus, N trades broader coverage against the learnability boundary set by $\epsilon_{\mathrm{train}}$.

4 EXPERIMENTS

Experimental setup. In the main experiments, we use Qwen3-1.7B and Qwen3-4B, both operated in non-thinking mode, as the policy models (Yang et al., 2025), and Qwen3-4B-Instruct-2507 as the judge–intervention model. We study both math reasoning and code generation. For math, we use a subset of the AceReason-Nemotron training prompts (Chen et al., 2026a), filtering out overly easy problems to avoid spending rollout compute on zero advantage examples. For code, we use the DeepCoder-Preview training set (Luo et al., 2025).

The semi-on-policy rollout generation is controlled by three hyperparameters: 1). *chunk size*, which determines how many tokens π_t generates between reviews; 2). *number of judge reviews*, which limits how many times π_{JI} can inspect a trajectory; and 3). *intervention continuation length*, which bounds the number of tokens generated by π_{JI} at each intervention. We use $(512, 4, 64)$ for math and $(128, 4, 64)$ for code, respectively. We implement two variants of the algorithm: MInTRL-Proxy retains the current-policy log probabilities as anchors for intervention-authored tokens, whereas MInTRL-Const replaces them with the constant anchor in equation 11.

We evaluate math reasoning on AIME 2025 (Balunovic et al., 2026), AIME 2026 (Dekoninck et al., 2026), and HMMT February 2025 (Balunovic et al., 2026), and code generation on LiveCodeBench (Jain et al., 2025), HumanEval+ (Liu et al., 2023), and MBPP+ (Liu et al., 2023). For every problem, we draw 32 samples and report their mean correctness, which estimates Pass@1. More hyperparameter settings and implementation details are provided in Appendix I.

Baselines. We compare MInTRL against four categories of baselines:

1. **Reference baselines.** The unmodified base policy and standard on-policy GRPO (Shao et al., 2024), which represent no post-training and conventional on-policy RL, respectively.
2. **Off-policy baselines.** SFT+GRPO first fine-tunes the policy on complete trajectories sampled from π_{JI}, and then continues training with on-policy GRPO.
3. **On-policy baselines.** Standard OPD (Agarwal et al., 2024; Lu & Lab, 2025) performs token-level distillation on trajectories sampled from the policy itself.
4. **Intervention-based baseline.** MENTOR (Jiang et al., 2026) selectively injects expert guidance into rollout generation and optimizes the resulting trajectories with a GRPO-style objective, making it the closest existing intervention-based RL method to MInTRL.

Together, these baselines provide broad coverage of representative on-policy and off-policy post-training methods, enabling a comprehensive comparison with MInTRL.

4.1 OVERALL RESULTS

As shown in Table 1, MInTRL achieves the best overall performance across both model scales and domains, outperforming standard on-policy RL, distillation-based methods, and teacher-guided baselines. For Qwen3-1.7B, MInTRL-Const reaches average scores of 35.45 on mathematics and 61.95 on code. Compared with the stronger of GRPO and OPD, this corresponds to gains of $+13.61$ and $+14.12$ percentage points (pp), respectively, with improvements on every individual benchmark. The gains remain substantial at the 4B scale: MInTRL-Const achieves 55.73 on mathematics and

Table 1: Performance on math and code benchmarks.[1][2]

	Mathematics				Code			
Method	AIME25	AIME26	HMMT25	Avg.	LCB	HE+	MBPP+	Avg.
Qwen3-4B-Instruct-2507	47.50	53.02	28.65	43.06	35.65	82.34	71.58	63.19
Qwen3-1.7B								
Base	8.85	8.85	4.48	7.40	12.99	55.89	52.18	40.35
GRPO	22.08	16.46	14.69	17.74	21.74	64.63	54.65	47.01
OPD	25.73	24.27	15.52	21.84	24.13	63.66	55.70	47.83
MENTOR	31.04	28.75	18.23	26.01	33.46	62.80	56.02	50.76
SFT+GRPO	31.56	27.60	16.46	25.21	34.20	78.28	67.15	59.88
MInTRL-Proxy	36.77	32.50	19.58	29.62	32.84	73.95	60.09	55.63
MInTRL-Const	**40.73**	**41.56**	**24.06**	**35.45**	**37.85**	**80.81**	**67.20**	**61.95**
Qwen3-4B								
Base	17.40	15.10	11.77	14.76	23.34	72.71	61.40	52.48
GRPO	60.83	60.31	36.98	52.71	50.06	80.32	67.13	65.83
OPD	43.02	50.10	28.44	40.52	35.11	77.31	63.98	58.80
MENTOR	61.88	**65.00**	**39.17**	55.35	53.20	86.38	75.83	71.80
SFT+GRPO	57.60	63.65	34.27	51.84	49.25	87.21	75.98	70.81
MInTRL-Proxy	57.60	58.23	37.71	51.18	52.68	86.79	**76.20**	71.89
MInTRL-Const	**65.31**	64.17	37.71	**55.73**	**54.19**	**87.84**	75.85	**72.63**

72.63 on code, improving over the two baselines by $+3.02$ pp and $+6.80$ pp, respectively. Moreover, MInTRL-Const generally outperforms the theoretically motivated MInTRL-Proxy, achieving higher aggregate performance across both domains and model scales. We hypothesize that this is because, under relatively weak policies, useful intervention tokens can have extremely low probability under the current policy and are therefore overly constrained by the policy-dependent anchor used in MInTRL-Proxy. Overall, these results demonstrate that sparse local interventions can effectively leverage off-policy guidance while preserving the benefits of predominantly on-policy rollout generation.

4.2 Training Dynamics

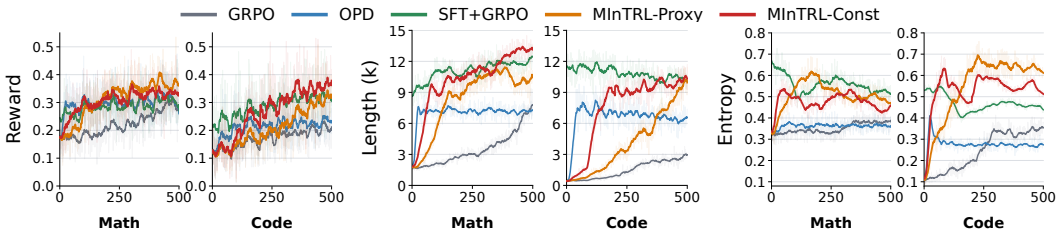
Having established the overall performance gains of MInTRL, we next investigate how these gains develop throughout training. We analyze the effects of MInTRL on policy exploration and diversity, dependence on intervention, and observable reasoning behavior. Unless otherwise noted, the following training-dynamics analyses are based on the 1.7B experiments reported above.

MInTRL promotes exploration and policy diversity. As shown in Figure 3, MInTRL achieves higher training reward while maintaining substantially higher policy entropy than GRPO and OPD. It also ultimately surpasses SFT+GRPO in training reward, despite relying only on sparse local interventions rather than full teacher-generated trajectories. Figure 4(a) further shows that both MInTRL variants outperform GRPO in Pass@k across the evaluated sampling budgets in both domains, with gains persisting at $k = 32$. Together, these results suggest that MInTRL improves access to successful reasoning paths while preserving policy diversity. Local corrections introduce promising prefixes that are difficult for the current policy to reach through on-policy sampling alone. By returning control to the policy after each correction, MInTRL allows it to explore its own continuations from these prefixes, combining external guidance with continued policy exploration.

MInTRL progressively reduces reliance on intervention. Figure 4(b) provides two complementary signals that the policy gradually learns from π_{JI} during training. First, the empirical forward KL

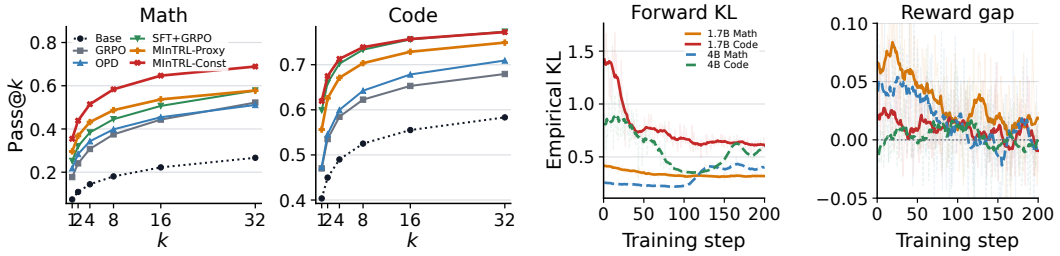
[1] All values come from the checkpoint with the highest three-benchmark average. Final-checkpoint results are reported in Appendix E.

[2] A recent method combining intervention with OPD (Xu et al., 2026) reports scores of 32.81 and 30.52 on AIME25 and AIME26, respectively. We did not reproduce these results locally; both reported scores are below those of our algorithm.



(a) Training reward. **(b)** Response length. **(c)** Entropy.

Figure 3: **Training dynamics.** MInTRL improves reward while maintaining longer responses and higher empirical token entropy throughout training.



(a) Pass@k performance. **(b)** Acquisition of intervention knowledge.

Figure 4: **Inference-time scaling and intervention knowledge.** Panel (a) reports domain-averaged Pass@k on evaluation benchmarks. Panel (b) shows the empirical forward KL on intervention tokens and the reward gap between semi-on-policy rollouts and on-policy rollouts

measured on intervention tokens generally decreases from its initial level, suggesting closer alignment between the policy and π_{JI} on these tokens. Second, the reward gap between semi-on-policy and purely on-policy rollouts progressively narrows toward zero. Interventions provide a clear reward improvement early in training, but offer diminishing additional benefits as training progresses. These trends suggest that the policy increasingly incorporates useful intervention guidance into its own generation.

MInTRL changes observable reasoning behavior. Keyword analysis across math and code shows more reconsideration and branching markers and fewer execution- and commitment-oriented markers in MInTRL evaluation rollouts than in GRPO rollouts. Due to the page limit, the detailed analysis and qualitative rollout examples are provided in Appendices F and D, respectively.

5 Ablation

5.1 MInTRL with Self Judge-Intervention

In the above section, the judge-intervention policy is instantiated by a model that is stronger than the policy being trained. We next examine whether this assumption is necessary. Instead of using a separate stronger teacher, we construct π_{JI} from the base policy π_0 itself, while providing it with additional privileged context. Following recent context-conditioned on-policy self-distillation (OPSD) approaches (Zhao et al., 2026; Shenfeld et al., 2026), for each x, we construct an auxiliary context z that provides additional information unavailable to the policy during normal generation. We then define the judge-intervention policy as $\pi_{\text{JI}}(\cdot \mid x) = \pi_0(\cdot \mid x, z)$, while the policy being trained only conditions on the original input x. In this ablation, we use Qwen3-4B-Instruct-2507 for both roles. The results are shown in Table 2. Both MInTRL variants outperform the corresponding baselines

Table 2: Performance with self judge-intervention.

Method	AIME25	AIME26	HMMT25	Avg.
GRPO	50.63	58.13	<u>33.02</u>	47.26
OPSD	47.71	55.73	31.67	45.03
MInTRL-Proxy	**59.06**	**61.56**	**36.25**	**52.29**
MInTRL-Const	<u>55.10</u>	<u>58.75</u>	31.56	<u>48.47</u>

on average. Interestingly, MInTRL-Proxy outperforms MInTRL-Const in this setting, achieving an average score of 52.29 versus 48.47, in contrast to the trend observed in Table 1. We hypothesize that this is because Qwen3-4B-Instruct-2507 provides a substantially stronger base policy, which already assigns reasonable probability to many intervention-authored tokens. Consequently, relaxing their reference probabilities with the constant anchor becomes less necessary, and retaining the original policy probabilities provides a more stable regularization signal.

5.2 Off-policy intensity

We next study how the amount of off-policy intervention affects RL performance. We vary the intervention intensity by changing the maximum number of judge reviews and the maximum intervention continuation length, which together control the fraction of rollout tokens authored by π_{JI}. Figure 5 plots the Pass@1 improvement over the no-intervention baseline[3] against the resulting off-policy token fraction, averaged over training. Both MInTRL variants exhibit a clear non-monotonic trend. Performance initially improves as a small amount of off-policy guidance is introduced, but degrades once interventions become too frequent or too long. The best performance is achieved at a moderate off-policy fraction of roughly 2–4%, whereas excessive intervention eventually underperforms the purely on-policy baseline. This inverted-U trend is consistent with the coverage–learnability trade-off analyzed in Section 3: moderate intervention expands rollout coverage, whereas excessive off-policy content makes the resulting trajectories harder for the current policy to learn from.

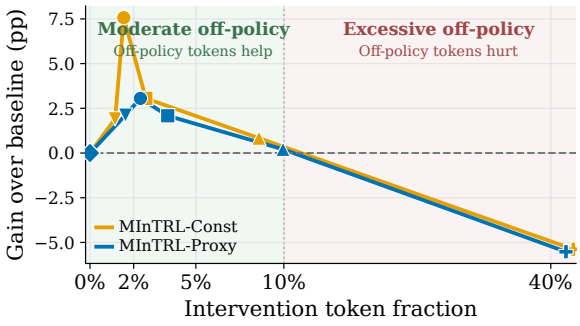


Figure 5: **Performance peaks at moderate off-policy intensity.** Moderate off-policy guidance improves performance, whereas excessive off-policy content erodes the gain and can even underperform the baseline.

5.3 Robustness to the Judge–Intervention Model

Our main experiments instantiate the judge–intervention policy with Qwen3-4B-Instruct-2507. To assess whether MInTRL depends on this particular model, we replace it with DeepSeek-V4-Flash, a strong model from a different model family. As shown in Appendix G, MInTRL remains effective with DeepSeek-V4-Flash as the judge–intervention model, consistently outperforming the GRPO and OPD baselines across policy scales and task domains. Interestingly, using a stronger standalone judge does not consistently yield better downstream performance. Although DeepSeek-V4-Flash improves some configurations, the gains vary across model scales, domains, and anchoring variants. We hypothesize that stronger corrections may also induce greater policy mismatch: compared with Qwen3-4B-Instruct-2507, DeepSeek-V4-Flash may generate intervention tokens that are less likely under the Qwen policy, making these corrections more difficult to learn from despite the judge's higher standalone quality.

6 Conclusion

We introduced MInTRL, which augments otherwise on-policy rollouts with sparse, local interventions and learns from the resulting trajectories using a regression-based RL objective. Our theoretical and empirical results reveal a coverage–learnability trade-off: moderate intervention exposes useful reasoning paths beyond ordinary on-policy sampling, whereas excessive off-policy content makes the resulting trajectories harder to learn from. Across mathematical reasoning and code generation, MInTRL consistently improves over on-policy RL and distillation baselines, suggesting that carefully controlled off-policy experience can expand exploration without sacrificing policy proximity.

[3]The 0% intervention condition is a pure on-policy control trained with the same advantage-regression objective and training setup, with intervention disabled.

Ethics Statement

This work studies reinforcement learning methods for improving mathematical reasoning and code generation in large language models. Our experiments use existing publicly available datasets and benchmarks and do not involve human-subject studies or the collection of personal data. We do not foresee direct negative societal impacts specific to the proposed method.

Reproducibility Statement

We describe the rollout construction, training objective, and practical implementation of MInTRL in Section 2. The models, datasets, training hyperparameters, and evaluation protocol are detailed in Section 4 and Appendix I. The judge and intervention prompt templates are provided in Appendix C. The assumptions underlying our theoretical analysis are stated in Section 3, with proofs provided in Appendix B. The protocol for measuring and comparing training efficiency is described in Appendix H.

AI Use Statement

Generative AI tools were used to assist with drafting and refining parts of the manuscript, identifying relevant literature, supporting brainstorming and discussions of experimental design, implementing the experimental code. All AI-assisted code was reviewed and checked by the authors.

We did not use generative AI tools to generate synthetic datasets; develop theoretical models or conceptual frameworks; formulate mathematical claims; provide critical ingredients for proving mathematical claims; write proofs; propose or refine research hypotheses; translate research materials; clean or reformat datasets; conduct qualitative or thematic data analysis; or interpret experimental results.

The authors take full responsibility for the final content of this work, including all technical claims, experimental results, and conclusions.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *Advances in Neural Information Processing Systems*, 38, 2026.

Kianté Brantley, Mingyu Chen, Zhaolin Gao, Jason Lee, Wen Sun, Wenhao Zhan, and Xuezhou Zhang. Accelerating rl for llm reasoning with optimal advantage regression. *Advances in Neural Information Processing Systems*, 38:151492–151531, 2026.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning. *Advances in neural information processing systems*, 38:110320–110345, 2026a.

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? *Advances in Neural Information Processing Systems*, 38:57654–57689, 2026b.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. *arXiv preprint arXiv:2605.00674*, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Peixuan Han, Runhui Wang, Ketan Ramaneti, Jie Hao, Gerald Friedland, and Chris Kong. Cliff: Learning process rewards from the first mistake. *arXiv preprint arXiv:2609.02817*, 2026.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Namgyu Ho, Laura Schmid, and Se-Young Yun. Large language models are reasoning teachers. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 14852–14882, 2023.

Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-rollout asynchronous optimization for agentic reinforcement learning. *arXiv preprint arXiv:2607.07508*, 2026a.

Zhichao Hou, Ferhat Erata, Joe Lilien, and MohamadAli Torkamani. Stratified consistency distillation for natural language formalization. *arXiv preprint arXiv:2608.30258*, 2026b.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the association for computational linguistics: ACL 2023*, pp. 8003–8017, 2023.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Naman Jain, 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*, volume 2025, pp. 58791–58831, 2025.

Zishang Jiang, Jinyi Han, Xinyi Wang, Sihang Jiang, Zhaoqian Dai, Ma Shuguang, Fei Yu, Jiaqing Liang, Yanghua Xiao, et al. Selective expert guidance for effective and diverse exploration in reinforcement learning of llms. In *International Conference on Learning Representations*, volume 2026, pp. 62980–63006, 2026.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1317–1327, 2016.

Xiaofeng Lin, Sirou Zhu, Yilei Chen, Mingyu Chen, Hejian Sang, Ioannis Paschalidis, Zhipeng Wang, Aldo Pacchiano, and Xuezhou Zhang. Scaling in-context online learning capability of llms via cross-episode meta-rl. *arXiv preprint arXiv:2602.04089*, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, et al. Deepcoder: A fully open-source 14b coder at o3-mini level. *Notion Blog*, 1, 2025.

Yuanjie Lyu, Chengyu Wang, Jun Huang, and Tong Xu. Student-centered distillation narrows the agentic gap between small and large llms. In *Forty-third International Conference on Machine Learning*, 2026.

Lu Ma, Hao Liang, Meiyi Qiang, Lexiang Tang, Xiaochen Ma, Zhen Wong, Junbo Niu, Chengyu Shen, Runming He, Yanhao Li, et al. Learning what reinforcement learning can't: Interleaved online fine-tuning for hardest questions. In *International Conference on Learning Representations*, volume 2026, pp. 80802–80821, 2026.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1773–1781, 2023.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*, 2023.

Daniel Ritter, Owen Oertell, Bradley Guo, Jonathan Chang, Kianté Brantley, and Wen Sun. Llms can learn to reason via off-policy rl. *arXiv preprint arXiv:2602.19362*, 2026.

Otmane Sakhi, Aleksei Arzhantsev, Imad Aouali, and Flavian Vasile. Off-policy learning to reason works because it is more pessimistic than you think. *arXiv preprint arXiv:2605.28150*, 2026.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025a.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025b.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash? why RLVR may or may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025.

Yangzhen Wu, Shanda Li, Zixin Wen, Xin Zhou, Ameet Talwalkar, Yiming Yang, Wenhao Huang, and Tianle Cai. Learn hard problems during rl with reference guided fine-tuning. *arXiv preprint arXiv:2603.01223*, 2026.

Tengyang Xie, Dylan J Foster, Yu Bai, Nan Jiang, and Sham M Kakade. The role of coverage in online reinforcement learning. *arXiv preprint arXiv:2210.04157*, 2022.

Haolei Xu, Xiaowen Xu, Haiwen Hong, Zixuan Ni, Hongxing Li, Yiwen Qiu, Weiming Lu, and Yongliang Shen. Pass the baton: Trajectory-relayed on-policy distillation. *arXiv preprint arXiv:2607.26057*, 2026.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. *Advances in Neural Information Processing Systems*, 38:117157–117186, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Matthew Yang, Hao Bai, Ian Wu, Gene Yang, Amrith Setlur, and Aviral Kumar. Int: Self-proposed interventions enable credit assignment in llm reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 85054–85091, 2026.

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

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, et al. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Xuechen Zhang, Zijian Huang, Yingcong Li, Chenshun Ni, Jiasi Chen, and Samet Oymak. Bread: Branched rollouts from expert anchors bridge sft & rl for reasoning. *Advances in Neural Information Processing Systems*, 38:96726–96752, 2026.

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

Haizhong Zheng, Jiawei Zhao, and Beidi Chen. Prosperity before collapse: How far can off-policy rl reach with stale data on llms? In *International Conference on Learning Representations*, volume 2026, pp. 82657–82679, 2026.

A Related works

A.1 RLVR and LLM reasoning

Reinforcement learning with verifiable rewards has become a central approach to post-training large language models for complex reasoning, particularly in mathematics and code, where solution correctness can be automatically verified (Shao et al., 2024; Jaech et al., 2024; Guo et al., 2025). Most RLVR methods build on policy-gradient algorithms such as PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024). Subsequent work has refined the optimization objectives and training procedures underlying RLVR to address objective bias, unstable policy updates, and training inefficiency. Representative methods include Dr. GRPO (Liu et al., 2025), DAPO (Yu et al., 2026), CISPO (Chen et al., 2025), VAPO (Yue et al., 2025), and A*PO (Brantley et al., 2026). Beyond single-turn reasoning, RLVR has also been extended to agentic settings, where models invoke tools, interact with external environments over long horizons, and receive verifiable feedback from task execution (Jin et al., 2025; Zeng et al., 2026; Lin et al., 2026; Hou et al., 2026a).

A.2 Knowledge distillation and on-policy distillation

Knowledge distillation (KD) transfers knowledge from a teacher model to a student by matching the teacher's predictive distributions or imitating its generated outputs (Hinton et al., 2015; Kim & Rush, 2016). For LLMs, teacher-generated responses, reasoning traces, and pseudo-labels have been widely used to transfer instruction-following and reasoning capabilities to smaller models (Hsieh et al., 2023; Ho et al., 2023; Mukherjee et al., 2023; Magister et al., 2023; Hou et al., 2026b). However, distillation based on teacher-generated trajectories is off-policy with respect to the student and can introduce a mismatch between its training and inference distributions. On-policy distillation mitigates this mismatch by training on student-generated trajectories with teacher supervision at the visited prefixes. GKD formalizes this paradigm (Agarwal et al., 2024), which has since been adopted at scale in reasoning models such as Qwen3 (Yang et al., 2025) and explored as an efficient alternative to RL (Lu & Lab, 2025). Recent work further extends this paradigm to self-distillation, where privileged context enables the same model to serve as both teacher and student (Zhao et al., 2026; Shenfeld et al., 2026).

A.3 Learning from off-policy experience

External experience can expose models to reasoning paths that are difficult to discover through their own sampling. One line of work incorporates such experience through supervised learning. ReGFT constructs reference-guided trajectories for SFT before RL (Wu et al., 2026), while ReLIFT interleaves supervised updates on difficult examples with RL training (Ma et al., 2026). More localized approaches, including InT (Yang et al., 2026) and SCoRe (Lyu et al., 2026), identify and correct the first error in student-generated trajectories, using the resulting experience for supervised training before subsequent RL.

Another line of work incorporates external guidance directly into RL. LUFFY combines expert demonstrations with on-policy rollouts and applies policy shaping to facilitate learning from off-policy traces (Yan et al., 2026). Zhang et al. (2026) use contextual hints or expert prefixes to steer student-generated rollouts. Several recent methods leverage expert feedback during RL to improve exploration and provide finer-grained credit assignment (Jiang et al., 2026; Han et al., 2026). These RL-based approaches rely on policy-gradient updates, often with modified importance weights, advantages, or rewards. MInTRL combines sparse corrections during online rollout generation with a sequence-level advantage-regression objective, learning from verifiable outcome rewards without auxiliary imitation losses or behavior-policy importance weighting. We further examine how intervention intensity affects the coverage and learnability of the resulting experience.

B Proofs for the Coverage–Learnability Trade-off

B.1 Proof of Lemma 3.3

Proof. Assumption 3.2 gives (13). Its optimum is zero when the inference support is empty; otherwise, placing all probability on any rollout in that support attains the population optimum of one. □

B.2 Proof of Theorem 3.5

Proof. Consider any correct rollout y generated by μ_N. The two distributions have the same token probabilities outside the correction set $I(y)$, while each corrected token is emitted deterministically by μ_N. Autoregressive factorization therefore gives

$$\frac{\pi_t(y \mid x)}{\mu_N(y \mid x)} = \prod_{t \in I(y)} \pi_t(y_t^\star \mid x, y_{<t}) \geq \nu^{|I(y)|} \geq \nu^N. \tag{16}$$

Thus, $\mu_N(y \mid x) > \epsilon_{\inf}$ implies $\pi_t(y \mid x) > \epsilon_{\inf}\nu^N$, proving the second inclusion in (15). The first follows directly from (14). □

C Prompt Templates

Prompt 1: Math judge prompt

System

You are a strict process verifier for mathematical reasoning. The trusted prefix is the immutable state accepted before the solver generated the new chunk. Use it as context, but inspect and number only the new chunk, checking its steps in order against the problem and trusted prefix. Classify the new chunk as exactly one of:

- `continue`: the new chunk is still logically recoverable, so all of it should be kept.
- `correct`: a new-chunk step contains an error after which the reasoning cannot be repaired without replacing that step. Return the 1-based number of the first such new-chunk step.

Important rules

- Never select text from the trusted prefix. `error_step` always indexes `numbered_new_chunk_steps`.
- Return `correct` only for the earliest mathematical step that must be replaced. Otherwise return `continue`.
- A different valid approach is not an error.
- An incomplete argument, unfinished sentence, or step that can still be completed correctly requires `continue`. In particular, do not mistake a sentence truncated by a length limit for a mathematical error.
- Do not propose, write, or reveal a correction. Your only task is to locate the first fatal error.
- For `correct`, `error_step` must be a valid positive step number. For `continue`, it must be 0.

Keep `critique` concise and specific.

User

```
{
  "problem": "{problem}",
  "latest_qwen_finish_reason": "{finish reason}",
  "trusted_prefix": "{accepted prefix}",
  "numbered_new_chunk_steps": [
    {"step_number": 1, "text": "{step 1}"},
    ...
  ]
}
```

Structured output

```
{"verdict": "{continue or correct}", "error_step": {0 or first error index}, "critique": "{brief reason}"}
```

Prompt 2: Code judge prompt

System

You are a strict process verifier for competitive-programming reasoning. The trusted prefix is the immutable state accepted before the programmer generated the new chunk. Use it as context, but inspect and number only the new chunk against the public problem statement, constraints, input/output contract, trusted prefix, and Python semantics. Return JSON with exactly two fields: first `reasoning`, then `error_step`.

Use `reasoning` to check the numbered steps carefully before deciding. State why the candidate is still recoverable, or identify the concrete public-specification violation and the earliest step that causes it. Then:

- Return 0 when the new chunk remains recoverable, so all of it should be kept.
- Otherwise return the 1-based number of the first new-chunk step containing an error after which a correct solution cannot be produced without revisiting that step.

Important rules

- Never select text from the trusted prefix. `error_step` always indexes `numbered_new_chunk_steps`.
- Focus primarily on the reasoning before the program: algorithm choice, invariants, edge cases, complexity, and input/output interpretation. Return a positive index only for a concrete error that materially leads away from a correct solution.
- A complete fenced code block is one atomic numbered step, even when it contains blank lines. Do not treat its internal paragraphs as separate steps.
- Judge only from the public specification and candidate text. Do not assume hidden tests or a reference implementation, and do not require one particular valid algorithm.
- If later reasoning or the final program has already corrected an earlier imprecision, return 0. Do not flag style, redundancy, missing explanation, or the existence of a simpler approach.
- An incomplete explanation, open code fence, unfinished program, or text truncated by a length limit requires returning 0 unless an earlier completed reasoning step already contains a fatal error.
- Do not propose or write a correction. Keep `reasoning` focused on verification and under 200 words, then return the error index.

User

```
{
  "problem": "{problem}",
  "latest_qwen_finish_reason": "{finish reason}",
  "trusted_prefix": "{accepted prefix}",
  "numbered_new_chunk_steps": [
    {"step_number": 1, "text": "{step 1}"},
    ...
  ]
}
```

Structured output

```
{"reasoning": "{brief verification}", "error_step": {0 or first
error index}}
```

Prompt 3: Intervention continuation prompt

User

```
{original problem}
```

Assistant prefill

```
{trusted prefix}{new-chunk text before the selected error step}
```

Prompt 4: Self-judge prompt

System

You are a strict process verifier for a problem-solving response. You receive a private high-level solution plan. Use it as a roadmap when checking the candidate, but independently verify every claim and code fragment: a plan can omit details, and a different valid route is not an error. The trusted prefix is the immutable state accepted before the solver generated the new chunk. Inspect and number only the new chunk. Return JSON with exactly two fields: first `reasoning`, then `error_step`.

Use `reasoning` to compare the candidate with the problem, trusted prefix, and private plan. Then:

- Return 0 when the new chunk is still logically recoverable.
- Otherwise return the 1-based number of the first new-chunk step containing an error that cannot be repaired without revisiting that step.

Never select text from the trusted prefix. Never quote, reveal, or continue the private plan, and do not require the candidate to copy its route. An incomplete or length-truncated step is recoverable unless an earlier complete step is already fatally wrong. For programming problems, check algorithmic correctness, state consistency, edge cases, and whether code fragments remain compatible with the accepted prefix. Do not propose a correction.

User

```
{
  "problem": "{problem}",
  "private_high_level_solution_plan": "{private plan}",
  "trusted_prefix": "{accepted prefix}",
  "latest_qwen_finish_reason": "{finish reason}",
  "numbered_new_chunk_steps": [
    {"step_number": 1, "text": "{step 1}"},
    ...
  ]
}
```

Structured output

```
{"reasoning": "{brief verification}", "error_step": {0 or first error index}}
```

Prompt 5: Self-intervention continuation prompt

User

```
Problem:
{original problem}

Private high-level solution plan:
{private plan}

Use the private high-level solution plan only as silent guidance.
Generate your own solution to the problem. Do not mention, quote, or
reveal the plan.
```

Assistant prefill

```
{trusted prefix}{new-chunk text before the selected error step}
```

Prompt 6: Privileged context example

Problem: Two positive integers differ by 5 and their product is 88. What is the larger integer?

Private high-level solution plan (hidden from the student): 1. Let the smaller integer be x and represent the larger as $x + 5$. 2. Use the product condition to set up the equation $x(x + 5) = 88$. 3. Rearrange to a quadratic and factor it, keeping only the positive integer solution for x. 4. Add 5 to that value to obtain the larger integer.

D QUALITATIVE ROLLOUT EXAMPLES

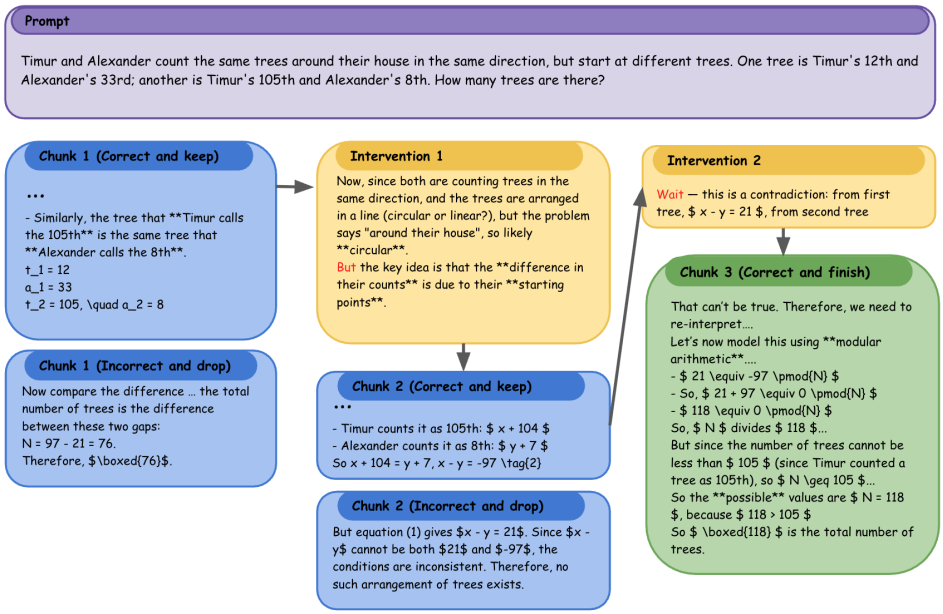


Figure 6: **Qualitative mathematics rollout with two local interventions.** The student first treats the two observed count gaps linearly and obtains the incorrect answer 76. The judge preserves the correctly extracted count pairs but rejects the subsequent linear subtraction because it ignores the wraparound of the circular sequence. Then, the first intervention redirects the reasoning toward offsets between the two starting positions. After that, the student derives the signed offsets $x - y = 21$ from the first shared tree and $x - y = -97$ from the second, but treats them as ordinary integer equalities and therefore concludes that no valid arrangement exists, overlooking that they need only agree modulo the unknown number of trees N. The judge rejects the no-solution conclusion and intervenes by highlighting the apparent contradiction between the two offsets. Afterward, the student reinterprets the offsets modulo N and derives the correct answer $N = 118$.

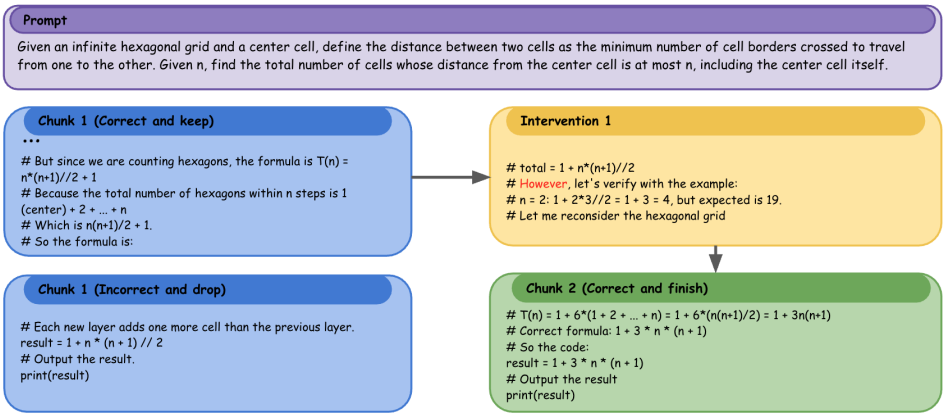


Figure 7: **Qualitative code rollout with one local intervention.** The student initially follows an incorrect triangular-number line of reasoning, suggesting the formula $1 + n(n + 1)/2$. The judge intervenes by checking the public example at $n = 2$, where the formula gives 4 instead of 19, and prompts the student to reconsider the hexagonal geometry. After the intervention, the student recognizes that the ring at distance k contains $6k$ cells, sums the rings, and implements the correct formula $1 + 3n(n + 1)$.

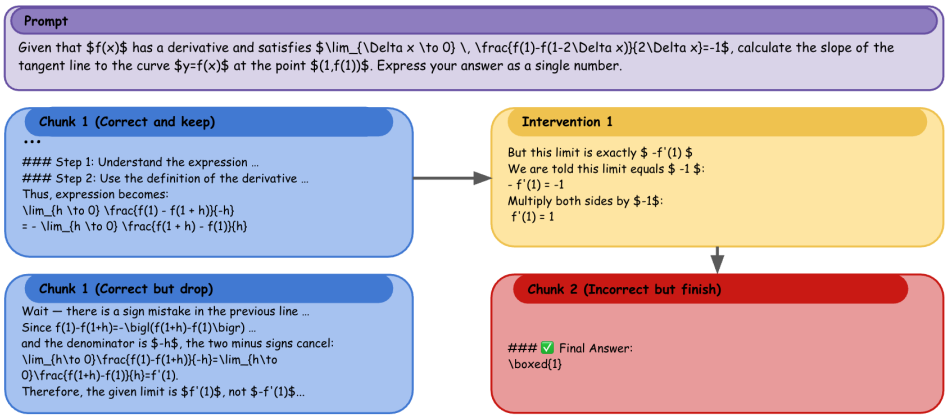


Figure 8: **A failure case of judge–intervention.** The student corrects its own sign error and reaches the correct result $f'(1) = -1$, but the judge mistakenly rejects this correction and provides an intervention with the wrong sign. The resumed student follows this faulty intervention and outputs incorrect answer 1.

This section presents qualitative rollouts illustrating both the benefits and limitations of local interventions. Figures 6 and 7 show how interventions can provide key insights or identify flaws in the student's reasoning, helping it reach a correct solution. Figure 8 illustrates the opposite possibility: an erroneous intervention can derail correct reasoning and lead to an incorrect final answer. These examples highlight a central design choice in MInTRL: interventions guide exploration without being treated as ground-truth supervision. Policy optimization is driven by the rule-based outcome reward of the completed trajectory, rather than by the presumed correctness of the intervention content.

E Final Checkpoint Evaluation

To complement the checkpoint-selected results in Table 1, we additionally evaluate all methods at the last checkpoints (500 steps). As shown in Table 3, MInTRL remains highly competitive under this fixed-checkpoint evaluation, consistently achieving strong performance across both model scales and domains.

Table 3: Performance on math and code at final checkpoint.

	Mathematics				Code			
Method	AIME25	AIME26	HMMT25	Math Avg.	LCB	HE+	MBPP+	Code Avg.
Qwen3-1.7B								
GRPO	22.08	16.46	14.69	17.74	21.74	64.63	54.65	47.01
OPD	23.33	22.60	15.21	20.38	24.13	63.66	55.70	47.83
MENTOR	28.44	27.08	16.04	23.85	33.27	62.94	56.00	50.74
SFT+GRPO	31.56	27.60	16.46	25.21	32.99	76.43	66.87	58.76
MInTRL-Proxy	35.10	31.25	20.94	29.10	32.84	73.95	60.09	55.63
MInTRL-Const	**40.73**	**41.56**	**24.06**	**35.45**	**37.85**	**80.81**	**67.20**	**61.95**
Qwen3-4B								
GRPO	60.83	60.31	36.15	52.43	50.06	80.32	67.13	65.83
OPD	42.50	47.60	28.85	39.65	35.11	77.31	63.98	58.80
MENTOR	63.02	63.96	**38.65**	55.21	**53.20**	86.38	75.83	71.80
SFT+GRPO	55.83	62.40	33.65	50.63	49.10	87.16	76.05	70.77
MInTRL-Proxy	55.10	56.35	32.19	47.88	52.18	85.94	75.87	71.33
MInTRL-Const	**65.31**	**64.17**	37.71	**55.73**	52.60	**88.36**	**76.60**	**72.52**

F Observable Reasoning Behavior

Analysis setup. We examine whether MInTRL affects not only solution quality but also observable reasoning behavior. We compare evaluation rollouts from GRPO, MInTRL-Proxy, and MInTRL-Const across math and code. For a fair comparison, we restrict the analysis to the first 512 narrative words of each response and measure the occurrence rates of reflection- and reasoning-related keywords within this fixed window. We report changes relative to GRPO in percentage points (pp).

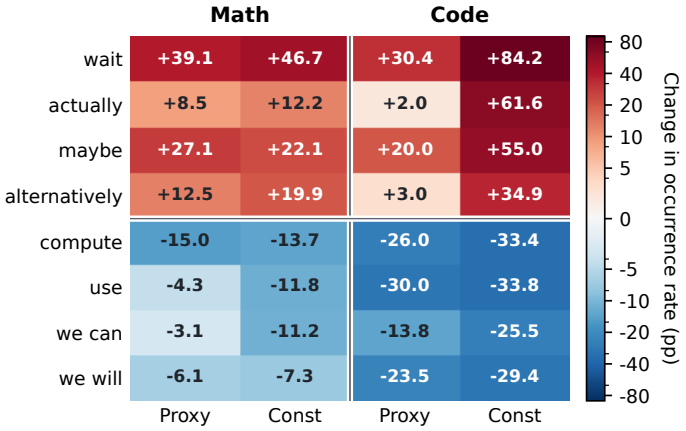


Figure 9: **Keyword shifts relative to GRPO.** Values show changes in keyword occurrence rates, measured in percentage points within the first 512 narrative words of evaluation responses. Reconsideration and branching markers (*wait*, *actually*, *maybe*, and *alternatively*) increase, whereas execution- and commitment-oriented markers (*compute*, *use*, *we can*, and *we will*) decrease.

Figure 9 reveals a consistent directional shift across both domains. Relative to GRPO, both MInTRL variants increase the occurrence of reconsideration and branching markers while reducing execution- and commitment-oriented markers. For example, under MInTRL-Const on code, the occurrence rates of *wait*, *actually*, and *maybe* increase by 55.0–84.2 pp relative to GRPO, while those of *compute*, *use*, *we can*, and *we will* decrease by 25.5–33.8 pp. MInTRL-Proxy exhibits the same directional pattern, with smaller changes on code. These shifts suggest that MInTRL encourages more frequent explicit reconsideration and branching before committing to an execution path. The smaller changes observed for MInTRL-Proxy on code are consistent with its stronger policy anchor, as discussed in Section 2.3. Qualitative examples in Appendix D further illustrate how local interventions introduce explicit reflection cues into rollouts. We interpret the keyword shifts as evidence of changes in observable reasoning style, rather than as a direct measure of reasoning quality.

G Robustness to the Judge–Intervention Model

Table 4: Performance with DeepSeek-V4-Flash as the judge–intervention model.

	Mathematics				Code			
Method	AIME25	AIME26	HMMT25	Avg.	LCB	HE+	MBPP+	Avg.
DeepSeek-V4-Flash	50.00	59.17	44.79	51.32	56.82	87.00	77.27	73.70
Qwen3-1.7B								
GRPO	22.08	16.46	14.69	17.74	21.74	64.63	54.65	47.01
MInTRL-Proxy	**41.46**	**43.54**	**25.52**	**36.84**	**35.17**	68.62	**56.48**	**53.42**
MInTRL-Const	39.17	40.10	25.00	34.76	32.91	**70.24**	55.70	52.95
Qwen3-4B								
GRPO	60.83	60.31	**36.98**	52.71	50.06	80.32	67.13	65.83
MInTRL-Proxy	61.46	**62.19**	35.00	52.88	**54.27**	**87.25**	**76.24**	**72.59**
MInTRL-Const	**63.44**	61.35	36.25	**53.68**	52.78	79.55	67.05	66.46

H Efficiency Analysis

We evaluate the training efficiency of MInTRL using resource-normalized wall-clock time, including the time spent on judge and intervention inference. In our experiments, GRPO and MENTOR each use 8 H100 GPUs, while MInTRL uses 10 (8 for training and 2 for judge–intervention inference) during the first 200 updates and 8 thereafter. To account for differences in GPU allocation, we normalize elapsed time to a 10-GPU reference by multiplying the duration of each interval by 0.8 when 8 GPUs are allocated and leaving it unchanged when 10 GPUs are allocated.

Efficiency relative to GRPO. Figure 10 compares training and evaluation reward against cumulative wall-clock time for GRPO and both MInTRL variants on Qwen3-1.7B mathematics training. We extend GRPO training to enable comparisons at matched wall-clock budgets. Both MInTRL variants achieve higher evaluation reward than GRPO at comparable budgets, with MInTRL-Const showing the largest gains. The evaluation gap persists even when GRPO is trained for longer than either MInTRL variant. These results show that the learning benefits of sparse interventions can outweigh their additional inference overhead.

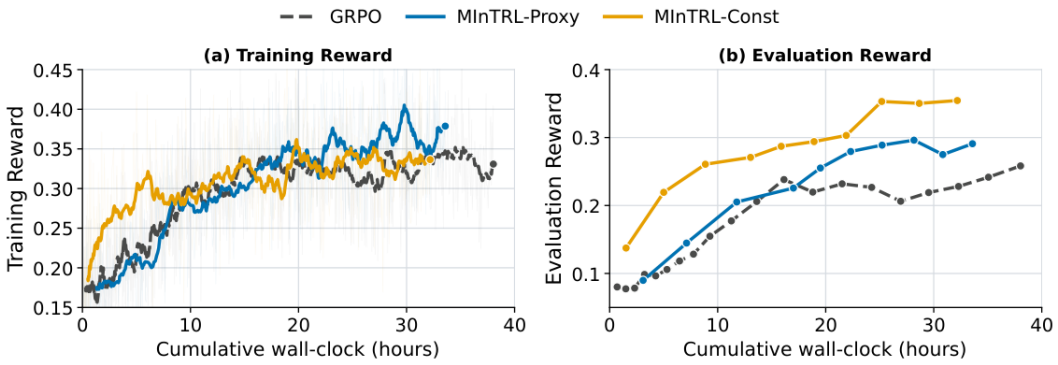


Figure 10: Wall-clock efficiency on Qwen3-1.7B mathematics training.

Efficiency relative to MENTOR. We compare MInTRL with MENTOR (Jiang et al., 2026) over the same number of policy updates during the shared intervention phase, excluding subsequent on-policy-only training. Both methods use the same student and expert models, 32 prompts per update, 8 on-policy and 8 expert-guided trajectories per prompt, a maximum response length of

Table 5: Wall-clock efficiency relative to MENTOR during the intervention phase. Times are normalized to 10-GPU-equivalent hours.

Student	Domain	MENTOR (h)	MInTRL-Const (h)	Speedup
Qwen3-1.7B	Math	17.77	6.44	2.76×
Qwen3-1.7B	Code	19.84	9.92	2.00×
Qwen3-4B	Math	23.22	11.07	2.10×
Qwen3-4B	Code	25.19	9.59	2.63×

16,384 tokens. Table 5 reports elapsed training time, with speedup defined as the ratio of MENTOR time to MInTRL-Const time.

As shown in Table 5, MInTRL-Const achieves a 2.00–$2.76\times$ speedup over MENTOR during the intervention phase, reducing wall-clock time by 50.0–63.8%. At the 4B scale, where the two methods achieve comparable final performance as in Table 1, the time savings are 52.3% on mathematics and 61.9% on code.

This efficiency advantage is consistent with the different mechanisms used to incorporate expert guidance. MENTOR mixes student and expert distributions at the token level and proposes speculative sampling to accelerate decoding. In contrast, MInTRL reviews student-generated chunks and introduces local corrections only when needed, allowing uninterrupted student decoding between review points and reducing the need for fine-grained coordination.

I Detailed Experimental Setups

Models and data. We use Qwen3-1.7B and Qwen3-4B in non-thinking mode as the student policies. Unless otherwise specified, we use Qwen3-4B-Instruct-2507 as the judge–intervention policy. For math, we train on a fixed set of prompts constructed by filtering AceReason-Nemotron to exclude overly simple problems. For code, we use the fixed DeepCoder-Preview training split. All experiments run for a total of 500 training steps, with student training performed on 8 x NVIDIA H100 80GB GPUs. For MInTRL, the judge–intervention model is served separately on two additional H100 GPUs and queried as needed during rollout generation.

Rollout construction. Each update samples a batch of 32 prompts and 16 trajectories per prompt. For MInTRL, these comprise 8 trajectories generated entirely by on-policy π_t and 8 semi-on-policy trajectories that permit intervention. MENTOR likewise uses 8 on-policy and 8 expert-guided trajectories per prompt. Other baselines use 16 on-policy trajectories per prompt. Both the policy and judge–intervention model sample with temperature 1.0 and top-p 1.0. Training responses are capped at 16,384 tokens, with a maximum prompt length of 2,048 tokens. The semi-on-policy rollout generation is controlled by three hyperparameters: 1). *chunk size*, which determines how many tokens π_t generates between reviews; 2). *number of judge reviews*, which limits how many times π_{JI} can inspect a trajectory; and 3). *intervention continuation length*, which bounds the number of tokens generated by π_{JI} at each intervention. We use $(512, 4, 64)$ for math and $(128, 4, 64)$ for code, respectively. The judge and intervention prompts used for the two domains are given in Appendix C.

Optimization and training schedule. We optimize the policy with AdamW using a learning rate of 1×10^{-6} and gradient clipping at 1.0, following the default veRL configuration. We use the mean reward of the 8 on-policy rollouts as the advantage baseline and optimize over both on-policy and semi-on-policy rollouts. For intervention tokens, MInTRL-Proxy uses their log probabilities under the current policy as anchors, while MInTRL-Const uses a fixed anchor $\kappa = -0.2$ for math and $\kappa = -0.1$ for code. For early stopping, we apply interventions only during the first 200 updates, followed by 300 updates of standard GRPO.

Hyperparameter selection. We tuned the rollout-construction hyperparameters by generating semi-on-policy rollouts with the base model. Specifically, we selected configurations that improved rule-based reward while keeping interventions sparse, thereby limiting the departure from the on-policy rollout distribution. For MInTRL-Const, we chose the domain-specific anchor κ based on

a coarse estimate of the mean log probability that Qwen3-4B-Instruct-2507 assigned to its own intervention-authored tokens in these pilot rollouts, resulting in $\kappa = -0.2$ for math and $\kappa = -0.1$ for code. We did not conduct an exhaustive or fine-grained hyperparameter search. Experiments using DeepSeek-V4-Flash as the judge–intervention policy reused the same rollout hyperparameters and κ values without recalibration. The improvements observed with these transferred settings suggest that the selected configurations remain effective with a different judge–intervention model, without additional judge-specific tuning.

Table 6: Core hyperparameters used in the main experiments.

Hyperparameter	Value
Batch size	32
Rollouts N_c / N_m	8 / 8
Maximum prompt length	2,048
Maximum response length T	16,384
Sampling temperature	1.0
Top-p	1.0
Math intervention configs	$(512, 4, 64)$
Code intervention configs	$(128, 4, 64)$
Intervention anchor κ	-0.2 (math), -0.1 (code)
Optimizer	AdamW
Learning rate	1×10^{-6}
Gradient clipping	1.0
β_1	∞
β_2	1×10^{-3}

Evaluation. We save checkpoints every 50 training steps and evaluate each using the same non-thinking decoding protocol. For each problem, we draw 32 independent samples with temperature 1.0, top-p 1.0, and a maximum response length of 32,768 tokens. We report mean sample correctness as Pass@1. In the main results, we report the checkpoint with the highest three-benchmark average for each method.

J LIMITATIONS

Experimental scope. Our evaluation focuses on mathematical reasoning and code generation with Qwen3 student models at the 1.7B and 4B scales. Extending the evaluation to additional student model families and long-horizon agentic tasks remains future work.

Computational trade-offs. MInTRL incurs additional inference costs for reviewing partial rollouts and generating local corrections compared with vanilla GRPO, even under self-intervention. These costs may be offset by more informative training trajectories that reduce the training needed to reach a target reward. Our wall-clock analysis demonstrates that sparse interventions can yield such efficiency gains. Substantial room remains for further improvement through more efficient rollout scheduling and adaptive intervention budgets.

Judge–intervention quality. As illustrated in Figure 8, the judge–intervention policy can occasionally reject correct reasoning or introduce erroneous corrections. Although MInTRL relies on rule-based outcome rewards and does not treat intervention content as ground-truth supervision, such errors can still disrupt promising reasoning and incur unnecessary inference costs. Improving the reliability of error detection and local correction could reduce unhelpful interventions and further improve both learning effectiveness and computational efficiency.