SAMPLE EFFICIENT GRPO VIA LEARNING TO REUSE TOKENS ACROSS TRAJECTORIES

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose FORK-RL: forking rollout trajectories in the middle and reusing the corresponding prefixes to boost RL post-training efficiency of LLMs in coding. GRPO estimates advantage based on group variety, hence the learning signal vanishes when all solutions in a group agree, whether they are all correct or wrong. As a result, such generated groups waste rollout budget and limit performance. Rather than increasing the sheer group size, FORK-RL resumes from the middle of existing trajectories to generate new solutions with different outcomes without additional rollout budget. We further introduce a learnable light-weight *score head* to identify the critical forking tokens. The score head decides where to fork according to the hidden states of random candidates. We train this head with a fiip reward when the new soluton leads to a different outcome. Our method not only endows the disagreement within groups, but also achieves finer grained credit assignment in the prefix-sharing trajectories. On competitive programming benchmarks, we achieve better performance than the compute-matched baseline: on CodeContests with a 4B policy model, a **17.8%** relative gain in pass@1 (24.5% against the baseline's 20.8%). Notably, our method reaches the baseline's training accuracy on **62%** fewer decoded tokens at 4B and **46%** fewer at 8B.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) enables large language models (LLMs) to achieve better performance by thinking more (Schulman et al., 2017; Guo et al., 2025; Shao et al., 2024; Wei et al., 2023). LLMs are prompted to think step-by-step to solve the problems, and are rewarded by a verifier judging on the answers' correctness. Trained in this way, a series of powerful large reasoning models (OpenAI et al., 2026; Guo et al., 2025) have emerged.

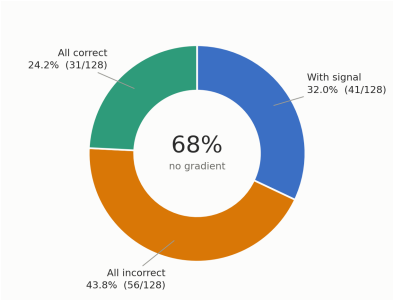

Figure 2: **Most groups carry no learning signal early in training.** The 128 groups of the first GRPO step with Qwen3-4B; only the blue slice has non-zero advantages.

Among multiple RLVR algorithms, GRPO (Shao et al., 2024) now becomes the most popular practice, which replaces the critic with a group related advantage calculation. However, GRPO could be rollout-ineffective. For each problem, GRPO samples a group of responses and normalizes the corresponding rewards to estimate advantages. Though such normalization reduces the need for a critic model Schulman et al. (2017), it requires variance of reward, which may not be realized in some cases. For instance, in early stages of training, all answers within some groups could be incorrect, and later in training all of them could be correct; in either case every advantage in the group is zero and the rollouts behind it teach the policy nothing. Figure 2 shows how large this waste is at the start of training: at the first step of a GRPO run with Qwen3-4B, only 32.0% (41/128) of the groups in the step carry any learning signal, while 43.8% (56/128) are all incorrect and 24.2% (31/128) are all correct, so 68% of the rollout budget of that step produces no gradient at all.

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.

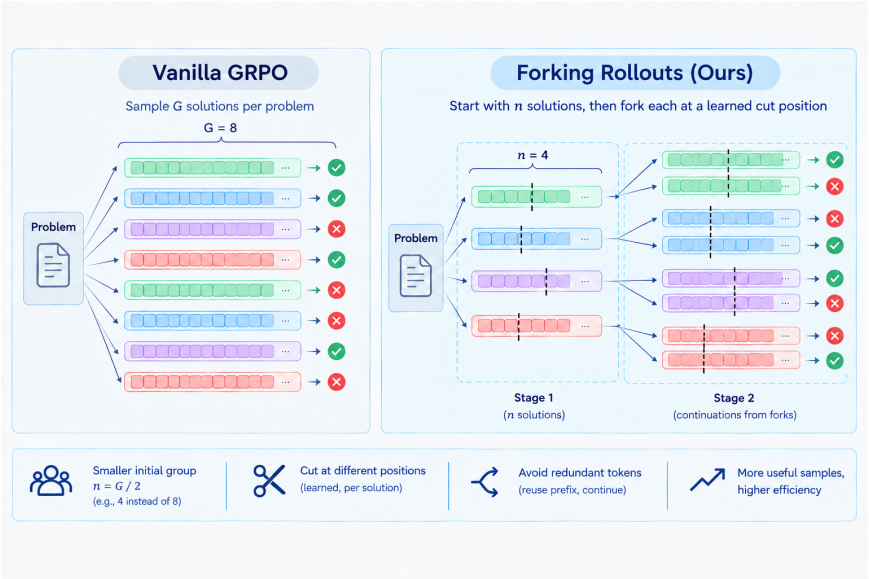

Figure 1: **GRPO versus forking rollouts.** Left: GRPO draws G complete solutions per problem independently, so when they all agree, whether all correct or all wrong, the within-group z-scores vanish and the step teaches the policy nothing. Right: FORK-RL draws a narrower group of n solutions (the initial rollouts, "Stage 1" in the figure), cuts each one at a position chosen by a learned score head, and continues from that prefix to a fresh ending ("Stage 2"). The second half of the group is therefore grown from parents whose verdicts are already known, from positions selected to reverse them, which is how a group that would have been degenerate acquires contrast. The continuation reuses its parent's previously generated prefix, so a group of size $2n$ costs less than $2n$ independent solutions to decode.

Naively scaling up the group size inflates the rollout budget without strictly guaranteeing outcome diversity. A growing body of literature explores dynamic training rollouts by branching trajectories at intermediate steps (Li et al.; Cao et al., 2026; Zhang et al., 2026b; Yang et al.). However, these methods rely on hand-crafted rules, like token entropy, rather than exploiting the policy model's hidden states. Existing learning-based methods focu only on failed trajectories to study the recoverability of early errors. While intuitive, this assumes the policy model already possessing the self-correction capacity. Moreover, focusing solely on failure recovery ignores the general exploration needed to actively flip outcomes and induce reward variance across all types of uniform groups.

Another line of works keeps the group size and spends more budget on selected problems, (Foster et al., 2026; Zhu et al., 2026; Yu et al., 2025; Luo et al., 2025). while this practice does not fully utilize the training problem set. Hence, a central problem remains: how could we increase the number of effective groups with fixed rollout budget and utilize the full set of training problems? In this paper, we propose FORK-RL, an algorithm that reduces the number of decoded token at training time while increasing effective group ratio (Figure 1): it takes a solution it has already graded, cuts it at one position, and grows a new ending from there. Sharing the prefix also buys *finer-grained credit assignment*. Two independent solutions differ from the first token on, so when one is right and the other wrong, the advantage that separates them is spread over everything they wrote. A parent and its continuation are identical up to the cut, so whatever separates their rewards was produced by the tokens after it, and the credit is aimed at the part of the solution that actually changed. This is also why we test on coding, where the prefix is most worth keeping: a program's early lines read the input, define a helper the rest of the solution calls, or set up the data structure the algorithm runs on, and they are rarely what decides pass or fail. A continuation inherits those lines instead of writing them again, and spends its tokens on the part of the program that actually determines the verdict.

Concretely, we sample a small group of n solutions per problem as usual (the *initial rollouts*), then pick a spot in each one, keep the prefix, and let the model write a new ending from there (a *continuation*). At the first new position we block the token the original used, so the ending has to differ instead of repeating the solution. Each new answer is checked by the same verifier, joins the

same GRPO group as its parent, and is trained with the same correctness reward, so a step builds a group of size $2n$ whose second half starts from solutions we have already graded. The prefix is previously generated content, reused rather than decoded again, which is what makes that second half cheap. Where to cut is learned too: a small *score head*, one trainable vector reading the model's own last-layer hidden state at each candidate, picks the position and is trained by REINFORCE (Williams, 1992) on whether the new ending *flipped* the parent's verdict. It adds no second model, no extra trained sequences, and no gradient into the language model. On competitive programming FORK-RL is ahead of a group-size-matched GRPO baseline at every shared validation point, and reaches the baseline's final training reward on 62% fewer decoded tokens at 4B and 46% fewer at 8B. Our contributions can be summarized as:

1. **We propose FORK-RL.** Instead of sampling every trajectory from scratch, we rebuild a group by truncating existing solutions and generating a new ending from its prefix. Reusing the prefix avoids redundant decoding.
2. **We train a lightweight score head to select the cut.** A single linear layer reads the policy model's hidden states at candidate positions and is trained via REINFORCE (Williams, 1992) based on whether the new ending flips the parent's verdict—requiring no auxiliary models and passing no gradients back to the policy.
3. **Together, they outperform baselines on competitive programming.** FORK-RL consistently outperforms a GRPO baseline across all validation checkpoints at both 4B and 8B scales. Notably, it matches the baseline's final training reward while consuming 62% fewer decoded tokens at 4B and 46% fewer at 8B. Beyond average score improvements, FORK-RL increases pass@8, successfully solving problems the baseline fails to solve in 8 attempts.

2 RELATED WORK

Where reasoning diverge. A line of work localizes the decisions steering a trajectory's outcome at successively coarser levels of abstraction. At the token level, reasoning tokens demonstrate a clear pattern that roughly 20% of them are high entropy and act as forking positions, and restricting policy-gradient updates to at these positions matches or exceeds full-gradient training (Wang et al., 2025). Similarly, decisions also occur at the step boundaries where models have low-confidence (Liu et al., 2025). Unlike the distribution-based analysis above, sentence-level importance can be realized counterfactually, that continuations are resampled from each sentence to measure the induced shift in the final-answer distribution (Bogdan et al., 2025). The most influential sentences are planning and uncertainty-management steps rather than the active-computation steps a forced-answer baseline would surface. At the representation level step-boundary activations occupy linearly separable step-specific subspaces (Sun et al., 2026), that correct and incorrect trajectories diverge geometrically only late, and that mid-reasoning correctness is predictable from trajectory features.

Dynamic Training Rollout. Instead of generating solutions from scratch for each training prompt, a separate literature explores spending rollout budget at special intermediate states and branching continuations. Existing branching methods relies on the policy model. TreePO (Li et al.) decodes in fixed-length segments and opens branches where local uncertainty is high; TreeAdv (Cao et al., 2026) branches at high-uncertainty decisions while sharing low-uncertainty tokens across rollouts. In the agentic scenario, branch can be created when post-tool-call entropy change exceeds a threshold (Dong et al., 2026; 2025). Recent work is actively exploring adaptive branching in either rule-based (Li et al., 2026) or learning-based methods. Since forking from earlier positions tend to affect more on the final results, hand-tuned depth bias $(t/T)^\gamma$ was introduced to alleviate such bias (Guo et al., 2026). RRPO and SRPO (Samanta et al., 2026) reset at a uniformly random step and at a step the model identifies by itself, respectively. RAIL (Zhang et al., 2026b) trains a contextual-bandit controller over twenty-six handcrafted rollout statistics, on candidates shortlisted by an entropy percentile. IGRPO (Zhang et al., 2026a) allocates tree-expansion budget by node-level informativeness. InT (Yang et al.) goes further and replaces the first erroneous step with a model-proposed alternative, localized by prompting with access to a reference solution. Neither of the existed literature has studied the exploration value of model's hidden representations during training. ThoughtProbe (Wang & Xu, 2025) ranks search-tree nodes with a classifier but is used as a test-time procedure. And every learning-based forking methods operate on failed trajectories only, so the quantity sought is always the recoverability of a failure rather than general exploration value.

An orthogonal line of work asks which training prompts deserve more sampling budget. DAPO (Yu et al., 2025) discards groups whose rollouts are uniformly successful/failed, and resamples until a disagreement appears. GRESO (Zheng et al., 2026) moves the filter before generation using the temporal consistency of prompt-level reward dynamics, and HIVE (Wu et al., 2026) adds a real-time entropy proxy after observing that the informative region shifts during training. VIP (Nguyen et al., 2026) models per-prompt success probability with a Gaussian process and solves a convex program minimizing expected gradient variance. CERO (Zong et al., 2026) maintains a Beta posterior and derives an $O(\sqrt{K})$ regret bound via a Fenchel-dual reformulation. VIGOR (Jiang et al., 2026) and Pilot-Commit (Kim et al., 2026) allocate progressively by measured group reward variance. The two axes are orthogonal.

3 Preliminaries: RL for LLM reasoning

Let π_θ be the policy, q a problem drawn from a dataset $\mathcal{D}$, and $o = (o^{(1)}, \ldots, o^{(|o|)})$ a sampled response. With a verifier that returns a binary reward $r(q, o) \in \{0, 1\}$, reinforcement learning with verifiable rewards maximizes

$$\mathcal{J}(\theta) = \mathbb{E}_{q\sim\mathcal{D},\, o\sim\pi_\theta(\cdot|q)}\big[\, r(q, o)\,\big]. \tag{1}$$

PPO (Schulman et al., 2017) optimizes this with a clipped surrogate over the importance ratio $\rho^{(t)}(\theta) = \pi_\theta(o^{(t)} \mid q, o^{<t}) / \pi_{\theta_{\text{old}}}(o^{(t)} \mid q, o^{<t})$ and a learned critic for the advantage. A critic is expensive to fit and, on a single terminal reward, largely redundant, so recent reasoning systems estimate the advantage from a group of samples instead.

GRPO. GRPO (Shao et al., 2024) samples a group of G responses $\{o_1, \ldots, o_G\} \sim \pi_{\theta_{\text{old}}}(\cdot \mid q)$ per problem, grades each one, and replaces the critic with the group's own statistics:

$$\hat{A}_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)}, \tag{2}$$

assigning the same $\hat{A}_i$ to every token of o_i and optimizing

$$\mathcal{L}(\theta) = -\,\mathbb{E}\Big[\tfrac{1}{G}\sum_{i=1}^{G}\tfrac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min\big(\rho_i^{(t)}(\theta)\hat{A}_i,\ \text{clip}(\rho_i^{(t)}(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_i\big)\Big]. \tag{3}$$

Following common practice, our runs and all baselines use GRPO with the two standard modifications from DAPO (Yu et al., 2025): a decoupled clipping range (ε_{low}, $\varepsilon_{\text{high}}$) and a token-level loss average, with the KL penalty to a reference model dropped. We do not use DAPO's resampling, and Equation 2 is untouched.

Degenerated groups. Equation 2 is the property this paper is built around. The estimator is *within-group*: if $r_1 = \cdots = r_G$, the numerator is zero for every member, so the group contributes no gradient no matter how many tokens it cost. DAPO's resampling is the standard admission that this is a real cost: it pays for those rollouts and then discards them. The normalizer also depends on G, so any method that changes the effective group size must hold G fixed against its baseline or admit that it is comparing two different estimators.

4 Methodology

In this section we explain how FORK-RL is implemented. Section 4.1 sets up *what forking in trajectories is*: what a forked trajectory is and what a forked group looks like. Section 4.2 answers *which tokens to fork*: which positions are candidates, how a small trainable vector picks one of them, and how that vector is trained from a reward of its own, paid when the new ending flips the verdict of the trajectory it came from. Section 4.3 covers *feeding forked trajectories into the actor*: how the prefix is reused, how we force the continuation to diverge, and how that forcing is undone again in the gradient. Nothing else changes. The loss, the verifier and the advantage stay the ones a GRPO step already uses (Section 3).

4.1 What is forking in trajectories

Fix a problem q and an initial trajectory $o = (o^{(1)}, \ldots, o^{(T)})$ sampled from $\pi_\theta(\cdot \mid q)$. A *fork point* is a position t; the *prefix* is $o_{<t}$. The continuation is sampled as $c \sim \pi_\theta(\cdot \mid q, o_{<t})$ by conditioning on

Algorithm 1 Building a forked group for one problem

Require: problem q; policy π_θ; verifier r; n initial trajectories per problem; score head v

Ensure: group of $2n$ graded trajectories

1: $o_1, \ldots, o_n \sim \pi_\theta(\cdot \mid q)$ ▷ initial trajectories, as in GRPO
2: **for** $i = 1$ **to** n **do**
3: draw candidate positions $t_1, \ldots, t_N$ of o_i and pick a fork point k_i with v ▷ Section 4.2
4: $c_i \sim \pi_\theta(\cdot \mid q, o_{i,<k_i})$, reusing the prefix $o_{i,<k_i}$ and avoiding $o_i^{(k_i)}$ at the first new position ▷ Section 4.3
5: **end for**
6: grade all $2n$ trajectories with r and give them the same group id
7: compute GRPO advantages over all $2n$ trajectories (Equation 2)
8: update π_θ with Equation 3, taking the ratio at each fork point against Equation 7
9: update v from the flip reward of Equation 5

$o_{<t}$ and decoding onward. The training group for q is then $\{o_1, \ldots, o_n\} \cup \{c_1, \ldots, c_n\}$. Both the initial trajectories and their continuations are graded by the same verifier and trained under the single correctness reward; there is no auxiliary loss on the policy model. Algorithm 1 summarizes how one forked group is built.

4.2 Which tokens to fork

Candidate token selection. For each of the trajectories sampled before forking, we draw a candidate set $t_1, \ldots, t_N$ by sampling K tokens randomly from a uniform distribution.

Choosing the forking token from candidate tokens with a score head. Let h_i be the policy model's last hidden state at the candidate token t_i, which is already computed by the forward pass. A light-weight trainable vector $v \in \mathbb{R}^d$ of the same size as the hidden state turns the candidate set into an N-dimensional distribution that we sample from:

$$\hat{h}_i = \frac{h_i}{\mathrm{rms}(h_i)}, \qquad s_i = \frac{\hat{h}_i^\top v}{T}, \qquad \pi^{\text{fork}} = \mathrm{softmax}(s), \qquad k \sim \pi^{\text{fork}}. \tag{4}$$

where k is the sampled forking token. The new trajectory resume at candidate k, thus reducing decoding burdens at training.

Training of the score head. We train the score head with REINFORCE (Williams, 1992) on a GRPO-style reward, and the reward is what the whole design turns on. We only reward a fork that flips the correctness of its parent trajectory. The flip reward is

$$r^{\text{fork}} = \mathbf{1}\big[\, r(q, c) \neq r(q, o) \,\big], \tag{5}$$

where o is the initial trajectory, $c \sim \pi_\theta(\cdot \mid q, o_{<k})$ its continuation, and $r(\cdot, \cdot)$ the same binary verifier that grades the policy. This r^{fork} encourages forking on the token where the outcome can be flipped compared to original trajectory. Motivated by the design of GRPO, we further use a GRPO-style advantage calculation on r^{fork}, normalizing it within the group of forks taken from the same problem, written $\hat{A}^{\text{fork}}$ below, to enable efficient training.

Thus, the gradient for training the score head is

$$\nabla_v J = \frac{1}{T} \sum_i c_i \, \hat{h}_i, \quad c_i = \hat{A}^{\text{fork}} \left(\mathbf{1}[t_i = k] - \pi_i^{\text{fork}} \right), \quad \hat{A}^{\text{fork}} = \frac{r^{\text{fork}} - \mathrm{mean}(r^{\text{fork}})}{\mathrm{std}(r^{\text{fork}})}, \tag{6}$$

where c_i is the weight the update puts on candidate t_i; the group-normalized flip advantage $\hat{A}^{\text{fork}}$ times the score-function term $\mathbf{1}[t_i = k] - \pi_i^{\text{fork}}$, which is positive at the candidate that was actually forked and negative at every other candidate. A fork that flipped the verdict therefore raises the score of the position it cut at and lowers the scores of the candidates passed over, and a fork that failed to flip does the opposite. The mean and standard deviation are taken over the flip rewards $r_1^{\text{fork}}, \ldots, r_n^{\text{fork}}$ of Equation 5 for the n forks taken on the same problem, so the score head is trained on exactly the group-normalized form of that reward.

4.3 Feeding forked trajectories into the actor

Reusing the prefix and regenerating. The prefix $o_{<k}$ is reused rather than decoded again, which is where the method's token saving comes from, and the model writes a new ending from there. We

make the model avoid the original's next token at the first decoded position, so that the continuation is forced to make a different *content* decision rather than simply carrying on with the original's continuation. Every run in this paper uses that setting.

Undoing the avoidance in the gradient. Avoiding that token means the continuation is not drawn from the policy. At the fork point the sampler draws from

$$\tilde{\pi}_{\theta_{\text{old}}}(a \mid q, o_{<k}) = \frac{\pi_{\theta_{\text{old}}}(a \mid q, o_{<k})\,\mathbf{1}[a \neq o^{(k)}]}{1 - \pi_{\theta_{\text{old}}}(o^{(k)} \mid q, o_{<k})}, \tag{7}$$

and only there; every later token comes from the unmodified policy. If we ignored this, the row would be credited as if the policy had steered away from $o^{(k)}$ on its own, when in fact we made it avoid that token. The surrogate is already importance weighted, so the fix is just to put the distribution we sampled from in the denominator. We apply the avoidance in the rollout engine, never in the loss, and take the ratio at the fork point against Equation 7,

$$\rho^{(k)}(\theta) = \frac{\pi_\theta(c^{(k)} \mid q, o_{<k})}{\tilde{\pi}_{\theta_{\text{old}}}(c^{(k)} \mid q, o_{<k})} = \frac{\pi_\theta(c^{(k)} \mid q, o_{<k})}{\pi_{\theta_{\text{old}}}(c^{(k)} \mid q, o_{<k})}\Big(1 - \pi_{\theta_{\text{old}}}(o^{(k)} \mid q, o_{<k})\Big), \tag{8}$$

which divides the avoidance's renormalizer back out. Every other position keeps the ordinary ratio. The avoidance only decides which continuation we see; the gradient is still for π_θ, the policy we deploy.

5 Experiment

5.1 Experimental Configuration

Training configuration. We train on TACO (Li et al., 2023), which is a commonly used training set for code generation. We run the experiment at two policy model sizes, Qwen3-4B and Qwen3-8B (Yang et al., 2025). We implement the score head as a vector to replace the language modeling head of the same model when forking. All experiments use verl (Sheng et al., 2025). Each step takes 128 problems with $n = 4$ initial samples per problem, and each initial sample is forked once, so a group contains 8 samples. We train with mini-batch 16 and actor learning rate 10^{-6}, use GRPO advantages with token-mean loss aggregation and decoupled clipping at $(0.2,\ 0.28)$ following DAPO (Yu et al., 2025), set the entropy and KL loss coefficients to 0, sample at temperature 1.0 for both the initial trajectories and their continuations, and cap prompts at 5120 and responses at 3072 tokens. We use GRPO with group size 8 and DAPO-style clipping as the baseline. All other settings remain the same to ensure fair comparison.

We use the single trainable vector of Section 4.2. We initialize it at $\sigma = 10^{-3}$, sample from it at temperature 1.0, and train it with a GRPO-style reward on the flip reward at learning rate 10^{-3}, warmed up linearly from 0 over the first 20 steps.

Evaluation configuration. We validate on four benchmarks: CodeContests (Li et al., 2022), HumanEval+ (Liu et al., 2023), MBPP+ (Liu et al., 2023), and LiveCodeBench (Jain et al., 2024). We score both experiments as pass@1 and pass@8 on every benchmark, at the shared starting checkpoint and at step 150, under each experiment's own sampling run. For training-rollout accuracy we score only the initial rollouts of the fork experiments, so both experiments are measured on the same quantity over the same rows.

5.2 Benchmark Results

We evaluate Fork-RL on coding tasks, where reusing an already generated program prefix avoids repeatedly decoding common setup and allocates more rollout tokens to the correctness-critical continuation. We assess its benefits from two perspectives: final task performance and training efficiency.

Higher accuracy across benchmarks. Table 1 reports pass@1 and pass@8 across four coding benchmarks for Qwen3-4B and Qwen3-8B. Starting from the same *Base* checkpoint, Fork-RL outperforms the group-size-matched baseline in all 16 model–benchmark–metric comparisons. The gains are largest on the more challenging contest-style benchmarks. With Qwen3-4B, pass@1 improves from 20.8% to 24.5% on CodeContests and from 45.2% to 46.9% on LiveCodeBench; with Qwen3-8B, the corresponding improvements are $22.8\% \rightarrow 25.9\%$ and $50.6\% \rightarrow 54.8\%$. On MBPP+

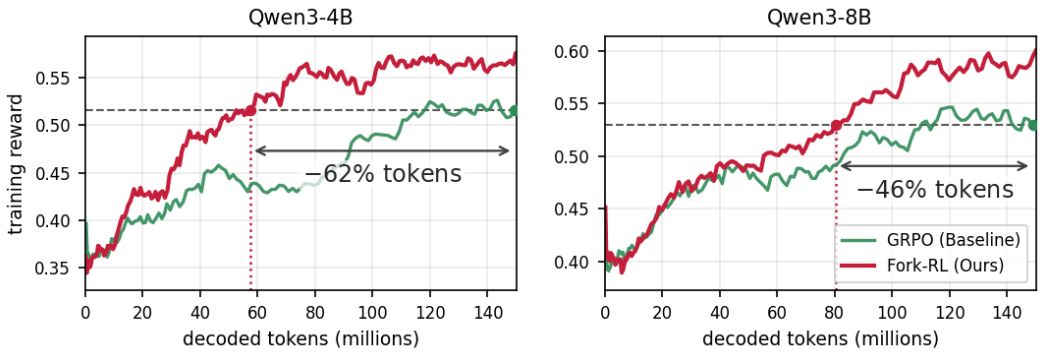

Figure 3: **FORK-RL learns more per decoded token.** Training reward against cumulative *decoded* rollout tokens at 4B and 8B. Red is FORK-RL, green its GRPO baseline; a curve above and to the left is learning more per token. The dashed rule is the baseline's reward at the end of that budget, the dotted line where FORK-RL first matches it, and the arrow the saving: 62% fewer decoded tokens at 4B and 46% at 8B.

Table 1: **Evaluation across four code benchmarks.** FORK-RL against its GRPO baseline, at both 4B and 8B, scored both as **pass@1** and **pass@8**. *Base* is the shared starting checkpoint. All entries are percentages; subscripts are differences from *Base* in percentage points, and bold marks the best within each size, metric and column. Benchmarks are MBPP+, HumanEval+ (HE+), CodeContests (CC) and LiveCodeBench (LCB); Avg is the unweighted mean over the four.

	MBPP+		HE+		CC		LCB		Avg	
Models	@1	@8	@1	@8	@1	@8	@1	@8	@1	@8
Qwen3-4B										
Base	68.3	72.1	77.1	82.6	8.3	18.2	26.8	38.9	45.1	52.9
GRPO (Baseline)	72.7$_{+4.4}$	80.4$_{+8.3}$	81.6$_{+4.5}$	90.9$_{+8.3}$	20.8$_{+12.5}$	38.2$_{+20.0}$	45.2$_{+18.4}$	60.5$_{+21.6}$	55.1	67.5
FORK-RL (Ours)	**74.6**$_{+6.3}$	**82.5**$_{+10.4}$	**83.0**$_{+5.9}$	**92.1**$_{+9.5}$	**24.5**$_{+16.2}$	**38.8**$_{+20.6}$	**46.9**$_{+20.1}$	**61.0**$_{+22.1}$	**57.3**	**68.6**
Qwen3-8B										
Base	71.9	76.4	81.0	86.0	10.1	18.0	30.7	44.4	48.4	56.2
GRPO (Baseline)	75.1$_{+3.2}$	82.8$_{+6.4}$	82.7$_{+1.7}$	92.7$_{+6.7}$	22.8$_{+12.7}$	39.4$_{+21.4}$	50.6$_{+19.9}$	64.8$_{+20.4}$	57.8	69.9
FORK-RL (Ours)	**76.5**$_{+4.6}$	**84.1**$_{+7.7}$	**83.8**$_{+2.8}$	**93.3**$_{+7.3}$	**25.9**$_{+15.8}$	**43.6**$_{+25.6}$	**54.8**$_{+24.1}$	**67.8**$_{+23.4}$	**60.3**	**72.2**

and HumanEval+, where both methods already exceed 70% pass@1, the gains remain consistent. Averaged across all four benchmarks, FORK-RL improves pass@1 by 2.2 points with Qwen3-4B and 2.5 points with Qwen3-8B.

At pass@8, FORK-RL again leads on every benchmark with both model sizes, with average gains of 1.1 points for Qwen3-4B and 2.3 points for Qwen3-8B. These results show that FORK-RL continues to improve model performance as the inference-time exploration budget increases.

Substantially improved token efficiency and higher final training reward. Figure 3 plots training reward against cumulative decoded rollout tokens, charging each method for the tokens it actually generates. With both Qwen3-4B and Qwen3-8B, the FORK-RL curve lies above and to the left of the baseline, achieving higher reward at a given decoding budget and reaching the same reward with fewer decoded tokens. At the reward attained by the baseline after 150M decoded tokens, FORK-RL reaches the same level using 62% fewer tokens with Qwen3-4B and 46% fewer with Qwen3-8B. Moreover, the reward advantage persists through the end of training, showing that FORK-RL improves not only token efficiency but also final model performance. The consistent token-efficiency gains from Qwen3-4B to Qwen3-8B further suggest FORK-RL has promising potential to scale to larger models.

The remaining subsections ablate the forking mechanism. We ask three questions: how the number of forking candidates affects the performance (Section 5.3), what forking actually buys (Section 5.4), and which trajectories are worth forking (Section 5.5).

Table 2: **Ablation study on number of candidate fork points.** Qwen3-4B pass@1 (avg@8) at step 100 on the four code benchmarks, for four runs that differ only in K, the number of candidate fork positions the head chooses from, plus the GRPO $G = 8$ control that never forks. Subscripts give the difference from that baseline in percentage points. $K = 1$ leaves the head no choice.

Experiment	MBPP+	HumanEval+	CodeContests	LiveCodeBench
Fork, $K = 8$	**73.3**$_{+1.1}$	**81.9**$_{+0.9}$	**21.4**$_{+4.1}$	**45.2**$_{+3.2}$
Fork, $K = 4$	72.8$_{+0.6}$	81.7$_{+0.7}$	20.9$_{+3.6}$	44.0$_{+2.0}$
Fork, $K = 2$	72.7$_{+0.5}$	81.1$_{+0.1}$	19.6$_{+2.3}$	43.4$_{+1.4}$
Fork, $K = 1$	72.4$_{+0.2}$	80.5$_{-0.5}$	18.5$_{+1.2}$	42.5$_{+0.5}$
GRPO $G = 8$	72.2	81.0	17.3	42.0

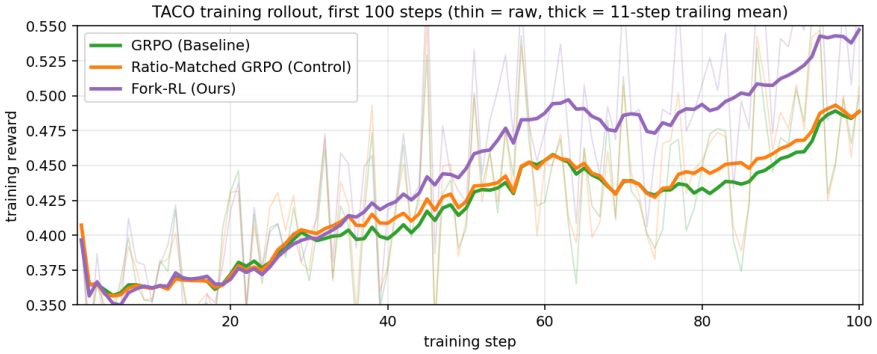

Figure 4: **FORK-RL versus GRPO with a same reward variance.** Here we compare fork RL to GRPO with a controlled reward variance on taco training set. For each group in the GRPO, the population of successful trajectories is forced to be the same as what it is in the corresponding FORK-RL group by rejection sampling. Hence, the improvement is isolated from the training reward variance. This shows the gain is mainly from the fine-grained credit assignment that forking enables.

5.3 FORKING CANDIDATE GRANULARITY

In each response we mark K positions at random and let the head choose one. A larger K gives the head more to choose from which, in theory, may result in better performance. But the performance is better for larger K if and only if the head can find a good cut among the candidates. If $K = 1$ the head has no choice left, and the cut is random. Table 2 compares four Qwen3-4B runs that differ only in K. More candidates is better, on all four benchmarks, with no exceptions.

With $K = 1$ the head has no choice left: the cut lands on one random spot, and that run is a tie with not forking at all. So cutting blindly buys nothing; having spots to choose between buys the gain. A separate A/B says the same at fixed K: replacing the head with a uniform draw, and changing nothing else, costs the whole margin on TACO training reward (0.497 against the no-fork baseline's 0.489, while the learned cut reaches 0.547), and the learned cut flips its parent's correctness 0.183 of the time against the random cut's 0.129, which is the disagreement the flip reward was built to manufacture.

This result indicates that the head is doing something useful, and that the gain not from simply selecting a random cut and fork it.

5.4 FINE-GRAINED CREDIT ASSIGNMENT BY FORKING

Forking changes two things at once. It changes *what is in the group*: the flip reward pushes continuations to disagree with their parents, so more groups has both correct and incorrect trajectories. And it changes *where the gradient lands*: a continuation shares its prefix with its parent and differs only after the cut, so their advantage difference is charged to the tokens after the cut. Either could explain the gain.



Figure 5: **Fork from the failed trajectories contribute the most.** We conduct another ablation which only forks the failed trajectories (red-dash). Forking only the failures has almost the same performance as forking both. This demonstrates forking is an efficient sampling strategy to create learning signal from a failed group.

To tell them apart we build a control that copies the correct-incorrect ratio but keeps the gradient where stock GRPO puts it: its trajectories are still sampled independently from the prompt, so no two of them share a prefix and nothing localises the credit. It draws 16 trajectories per problem instead of 8 and keeps the 8 that reproduce how many of the fork run's trajectories were correct on that problem at that step; the group it trains on is still exactly 8, and the mix is matched exactly on 93% of problems. It does not reproduce the gain (Figure 4): it ends at 0.489, on top of the plain baseline (0.489), while the fork ends at 0.547. So the fork does make the mix healthier, but an experiment handed that mix without the shared prefix learns no faster.

So the accuracy gain mainly comes from the fine grained credit assignment. The forked and the original trajectory differ only after the fork, so their advantage difference falls on that different part, while two independent samples differ everywhere and can localise nothing. That fine-grained credit assignment is what forking buys, and the head's job is to find the cuts where it is informative.

5.5 Efficient sampling by forking near-success trajectories

Forking a correct trajectory risks talking the model out of an answer it already had, so an obvious variant is to fork only the trajectories that were wrong. We run this experiment: continuations resume only from incorrect initial trajectories. Figure 5 plots the two experiments against each other. They are indistinguishable — 0.549 against 0.547 at step 100, with the curves on top of each other for the whole window — and both are well clear of GRPO (0.489). This indicates that the most performance gain in our method comes from fixing a all incorrect group and having fine-grained credit assignment.

6 Conclusion

In this paper, we explore forking as an efficient rollout strategy for RL post-training in LLMs. Recognizing that GRPO group learning signals vanish when rollouts agree—and that groups teach the most when split evenly—we propose forking to deliberately induce disagreement within groups. By resuming generation from a known parent trajectory, our approach reuses settled prefixes and cuts at tokens where the verdict is most likely to reverse.

To select critical forking points, we introduce a lightweight vector head operating on the policy model's hidden states without requiring auxiliary models, additional trained parameters, or gradient updates to the language model. On competitive programming tasks, this approach yields significant efficiency and accuracy gains: a group-size-matched baseline is led at every shared validation point at both 4B and 8B scales (pass@1 0.548 against 0.506), while matching baseline training reward on 62% fewer decoded tokens at 4B and 46% fewer at 8B. Ultimately, actively purchasing disagreement rather than relying on random sampling provides richer contrast, converting prefix reuse into superior task performance.

AI USE STATEMENT

LLMs were employed in a limited capacity for writing optimization. Specifically, the authors provided their own draft text to the LLM, which in turn suggested improvements such as corrections of grammatical errors, clearer phrasing, and removal of non-academic expressions. LLMs were also used to inspire possible titles for the paper. While the system provided suggestions, the final title was decided and refined by the authors and is not directly taken from any single LLM output. In addition, LLMs were used as coding assistants during the implementation phase. They provided code completion and debugging suggestions, but all final implementations, experimental design, and validation were carried out and verified by the authors. Importantly, LLMs were NOT used for generating research ideas, designing experiments, or searching and reviewing related work. All conceptual contributions and experimental designs were fully conceived and executed by the authors.

REPRODUCIBILITY STATEMENT

Section 4.2 gives the score head in closed form; the ablation's experiments differ in exactly the `fork_score` block. Every accuracy and token number we report is regenerated by scripts from the shipped per-step CSVs (accuracy, total tokens, mean prompt and response lengths, spliced-prefix token counts, head diagnostics), and every embedding-distance number from the per-pair distance dumps (both distances, both lengths, the fork offset, and the correctness of parent and continuation).

REFERENCES

Paul C. Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. Thought anchors: Which llm reasoning steps matter?, 2025. URL https://arxiv.org/abs/2506.19143.

Lang Cao, Hui Ruan, Yongqian Li, Peng Chao, Wu Ning, Haonan Song, Renhong Chen, and Yitong Li. Treeadv: Tree-structured advantage redistribution for group-based rl. *arXiv preprint arXiv:2601.03703*, 2026.

Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, et al. Agentic entropy-balanced policy optimization. *arXiv preprint arXiv:2510.14545*, 2025.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. In *International Conference on Learning Representations*, volume 2026, pp. 16981–17017, 2026.

Thomas Foster, Anya Sims, Johannes Forkel, Mattie Fellows, and Jakob Foerster. Learning to reason at the frontier of learnability, 2026. URL https://arxiv.org/abs/2502.12272.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi

Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Yiran Guo, Zhongjian Qiao, Yingqi Xie, Jie Liu, Dan Ye, Ruiqing Zhang, Shuang Qiu, and Lijie Xu. Deep dense exploration for llm reinforcement learning via pivot-driven resampling. *arXiv preprint arXiv:2602.14169*, 2026.

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

Heyang Jiang, Henry Liu, and Baharan Mirzasoleiman. Learning as reasoning unfolds: Progressive rollout allocation for efficient reinforcement learning. *arXiv preprint arXiv:2607.22002*, 2026.

Woojeong Kim, Ziyi Yang, Jing Nathan Yan, and Jialu Liu. Spend your rollouts where it counts: Rollout allocation for group-based rl post-training. *arXiv preprint arXiv:2605.26606*, 2026.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset, 2023. URL https://arxiv.org/abs/2312.14852.

Xintong Li, Sha Li, Yuwei Zhang, Changlong Yu, Rongmei Lin, Hongye Jin, Shuyi Guan, Xin Liu, Linwei Li, Qingyu Yin, et al. Process reward informed tree rollout for effective multi-turn rl. *arXiv preprint arXiv:2607.15610*, 2026.

Yizhi Li, Qingshui Gu, Zhoufutu Wen, Ziniu Li, Tianshun Xing, Shuyue Guo, Tianyu Zheng, Xin Zhou, Xingwei Qu, Wangchunshu Zhou, et al. Treepo: Bridging the gap of policy optimization and efficacy and inference efficiency with heuristic tree-based modeling, 2025. *URL https://arxiv.org/abs/2508.17445*.

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, December 2022. ISSN 1095-9203. doi: 10.1126/science.abq1158. URL http://dx.doi.org/10.1126/science.abq1158.

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, 2023. URL https://arxiv.org/abs/2305.01210.

Yuliang Liu, Junjie Lu, Zhaoling Chen, Chaofeng Qu, Jason Klein Liu, Chonghan Liu, Zefan Cai, Yunhui Xia, Li Zhao, Jiang Bian, Chuheng Zhang, Wei Shen, and Zhouhan Lin. Adaptivestep: Automatically dividing reasoning step through model confidence, 2025. URL https://arxiv.org/abs/2502.13943.

Michael Luo, Sijun Tan*and Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, and Jeffrey Luo. DeepScaleR: Surpassing O1-Preview with a 1.5B model by scaling RL. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005b 2025. Notion Blog.

Trung Hieu Nguyen, Bao Nguyen, Wenao Ma, Yuzhi Zhao, Ruifeng She, and Viet Anh Nguyen. Adaptive rollout allocation for online reinforcement learning with verifiable rewards. In *International Conference on Learning Representations*, volume 2026, pp. 136100–136132, 2026.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Bohan Zhang, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wenting Zhan, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2026. URL https://arxiv.org/abs/2412.16720.

Ankur Samanta, Akshayaa Magesh, Ayush Jain, Youliang Yu, Daniel Jiang, Kavosh Asadi, Kaveh Hassani, Paul Sajda, Jalaj Bhandari, and Yonathan Efroni. Credit assignment with resets in language model reasoning. *arXiv preprint arXiv:2605.25507*, 2026.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 1279–1297. ACM, March 2025. doi: 10.1145/3689031.3696075. URL http://dx.doi.org/10.1145/3689031.3696075.

Lihao Sun, Hang Dong, Bo Qiao, Qingwei Lin, Dongmei Zhang, and Saravan Rajmohan. Llm reasoning as trajectories: Step-specific representation geometry and correctness signals, 2026. URL https://arxiv.org/abs/2604.05655.

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

Zijian Wang and Chang Xu. Thoughtprobe: Classifier-guided llm thought space exploration via probing representations. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6029–6050, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. doi: 10.1007/BF00992696.

Jiahao Wu, Ning Lu, Shengcai Liu, Kun Wang, Yanting Yang, Bailong Lin, Chen Jason Zhang, Li Qing, and Ke Tang. Train at moving edge: Online-verified prompt selection for efficient rl training of large reasoning model. *arXiv preprint arXiv:2603.25184*, 2026.

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

Matthew YR Yang, Hao Bai, Ian Wu, Gene Yang, Amrith Setlur, and Aviral Kumar. Int: Self-proposed interventions enable credit assignment in llm reasoning, 2026. *URL https://arxiv.org/abs/2601.14209*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Yijun Zhang, Fan Xu, Jiaxin Ding, Yule Xie, Shiqing Gao, Xin Ding, Haoxiang Zhang, Luoyi Fu, and Xinbing Wang. Information gain-based rollout policy optimization: An adaptive tree-structured rollout approach for multi-turn llm agents. *arXiv preprint arXiv:2607.06223*, 2026a.

Zheyuan Zhang, Manqing Mao, Hong Wang, Zhuoer Wang, Samson Koelle, Jie Yuan, Yanjun Lin, James Feng, Nikki Lijing Kuang, Yanfang Ye, et al. Optimizing what policies learn from: Recoverability-aware rollout intervention learning. *arXiv preprint arXiv:2608.05080*, 2026b.

Haizhong Zheng, Yang Zhou, Brian Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts. *Advances in Neural Information Processing Systems*, 38:124321–124346, 2026.

Tianshu Zhu, Wenyu Zhang, Xiaoying Zuo, Lun Tian, Haotian Zhao, Yucheng Zeng, Jingnan Gu, Daxiang Dong, Jianmin Wu, Dawei Yin, and Dou Shen. Rollout pass-rate control: Steering binary-reward rl toward its most informative regime, 2026. URL `https://arxiv.org/abs/2605.05112`.

Yiming Zong, Yige Wang, and Jiashuo Jiang. Cross-epoch adaptive rollout optimization for rl post-training. *arXiv preprint arXiv:2606.05606*, 2026.

A Additional tables and figures

The vector learns, hard. From an essentially uniform softmax over roughly 48 positions, the probability the head puts on the cut it picks rises $0.029 \to 0.819 \to 0.941$ at steps 1, 50 and 200, its selection entropy falls $3.72 \to 0.12$ nats, and its weight norm grows from 0.073 to 0.919. By step 200 it is putting 0.94 on one position out of 36 where a uniform draw would put 0.028, a $34\times$ concentration, from 2,560 parameters and a REINFORCE signal with no baseline.

Cutting damages code more than reasoning. On code the continuation is consistently worse than its parent, with continuations at $0.208 \to 0.473$ against their parents' $0.361 \to 0.516$, so cutting a short program and regenerating damages it more than the same surgery damages a chain of reasoning. The experiment wins on token-matched accuracy anyway, because the continuations are cheap.

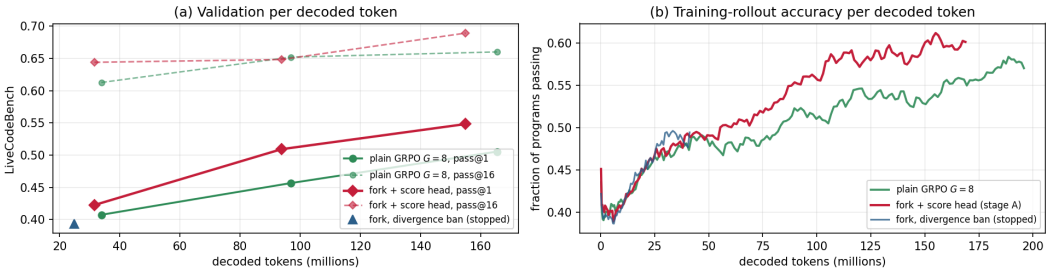

Figure 6: The Qwen3-8B coding pair at full size: the second panel of the bottom row of Figure 3, with pass@16 and the rollout curve added. *(a)* LiveCodeBench pass@1 (solid) and pass@16 (dashed); the triangle is the divergence-ban variant, stopped at step 71. *(b)* Training-rollout accuracy, initial rollouts only for the fork experiment so both curves measure the same quantity on the same rows. The experiments are inseparable for the first $\sim$50M decoded tokens and then part, the same shape the 4B pair shows.

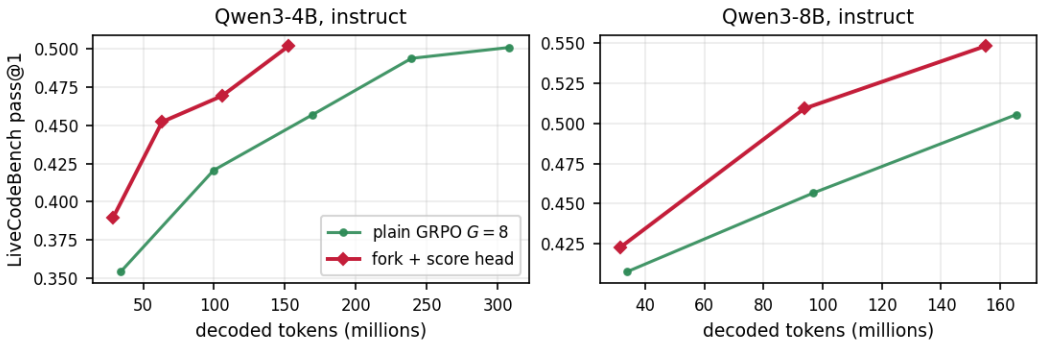

Figure 7: Figure 3 on *validation* accuracy instead of training rollouts, showing LiveCodeBench pass@1 against the same decoded-token axis, with each run drawn to the end of its own budget rather than to Figure 3's shared window. Validation is far sparser (every 50 steps on the coding runs), so these panels are a few markers rather than curves, which is why the training-rollout view is the one in the main text. The two views agree.

B How far a continuation travels

A fork is only worth its tokens if the continuation is a different trajectory, not a restatement of its parent. We encode every trajectory with bge-m3 (mean-pooled, L_2-normalized; distance is $1 - \cos$) and compare two populations at six checkpoints spanning the run. *Fork pairs* (2,814) put each initial trajectory against its own continuation, measured both over the whole text and over the post-fork suffix alone, since the continuation replays its parent verbatim up to the cut, so the whole-text distance counts shared text the fork never had the option to change. *Independent pairs* (6,144) put rollout j against rollout $j{+}4$ inside each 8-sample baseline group, which measures the spread plain sampling produces and sets the bar a fork has to clear.

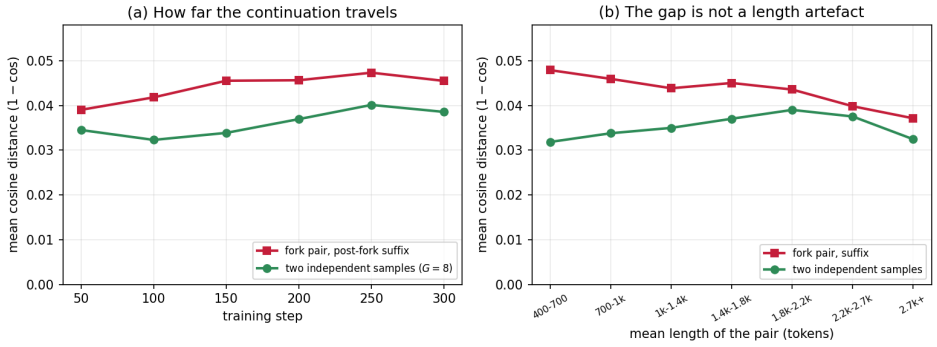

Figure 8: *(a)* Distance per checkpoint: the post-fork suffix of a fork pair sits above two independent samples at every checkpoint. *(b)* The same two quantities inside shared length bins, where the ordering is unchanged. The whole-trajectory fork distance, which the shared prefix holds *below* independent sampling, is reported in the text; flip rate against suffix-distance quintile is in Table 5.

A continuation diverges more than a fresh sample, once you look at the part it wrote. Over the whole text a fork pair is *closer* than two independent samples, 0.0274 against 0.0361, which is what the shared prefix buys, and is the wrong number to read as diversity. Restricted to the post-fork suffix the ordering flips: 0.0440 against 0.0361, and the fork suffix is above independent sampling at all six checkpoints (Figure 8a). Mean pooling drifts long texts toward their own centroid and the fork suffixes are the shortest population here, so we redo the comparison inside shared length bins, weighting each by the smaller experiment's count; the gaps survive at essentially full size,

-0.0085 and $+0.0076$ against raw values of -0.0087 and $+0.0079$ (Figure 8b). Resuming from a mid-trajectory prefix with the parent's next token banned produces a continuation that ends up *further* from its parent than an independent sample would, at a fraction of the decoding cost.

Provenance. These dumps share this paper's rollout machinery (same resume-after-delimiter rule, skip-header walk, divergence ban, n and data), but come from a different fork run, whose dumps were not retained. What they measure is a property of forking at a delimiter and of the flip reward rather than of one selector.

Table 3: Coding port. Training-rollout accuracy (11-step trailing mean) at shared decoded-token budgets. The naive total is the logged response-token count before the cached-prefix correction. Subscripts give the FORK-RL row's difference from the control above it, green for a gain and red for a loss.

	Rollout accuracy at matched decoded tokens			
Experiment	**40M**	**80M**	**120M**	**152.6M**
GRPO, $G = 8$	0.433	0.437	0.525	0.506
FORK-RL	**0.486**$_{+0.053}$	**0.552**$_{+0.115}$	**0.568**$_{+0.043}$	**0.569**$_{+0.063}$

Totals: fork 200 *steps,* 152.6*M decoded (*186.7*M before the cached-prefix correction); GRPO* 299 *steps,* 374.8*M decoded.*

C QWEN3-1.7B

This is the smallest policy model we ran. We include it because it is the cleanest picture in the paper of the failure mode Section 5.3 diagnoses: a head that succeeds at its own objective and stops converting it.

Table 4: Qwen3-1.7B on TACO/LiveCodeBench against a group-size-matched GRPO $G = 8$ control from the same checkpoint, on a training-step axis, LiveCodeBench pass@1. The pair is a wash. Entries marked n/a are points the shorter run never reached. Subscripts give the FORK-RL row's difference from the control above it, green for a gain and red for a loss.

	Training step				
Experiment	**50**	**100**	**150**	**200**	**250**
GRPO, $G = 8$	0.232	**0.303**	n/a	n/a	n/a
FORK-RL	**0.241**$_{+0.009}$	0.290$_{-0.013}$	n/a	n/a	n/a

The head saturates while its reward stays flat. The 2,048-parameter vector does exactly what it did at 4B: the probability it places on the cut it picks rises $0.176 \to 0.741 \to 0.873$ at steps 1, 50 and 250, against a uniform draw's 0.145 falling to 0.076 as the candidate set grows from 6.9 to 13.1 positions. But its own reward does not follow: the flip fraction is 0.111 at step 1, 0.215 at step 100 and 0.175 at step 250, i.e. flat from step 50 onward while the selector concentrates by another factor of ten. A selector that commits harder and harder without flipping more is committing to something constant within the group, and that is the point at which the experiment stops leading. Read alongside Table 5, where flip rate rises with distance while net accuracy gain does not, this is the paper's central negative result at the smallest size we ran it.

Code: a wash. On TACO/LiveCodeBench the two 1.7B experiments are within noise of each other at both shared validation points ($+0.009$ then -0.013 pass@1; pass@16 0.389/0.455 against 0.377/0.460), with the fork experiment slightly ahead on training rollouts (0.356 against 0.348 initial-rollout solve rate at step 100). The fork mechanism itself worked, and candidates per response grew $5.3 \to 17.8$ as responses lengthened, so the head was never short of positions to choose between. Both runs were stopped early for compute, the control at step 100 and the fork experiment at 141, which is why Table 4 carries only two points.

Table 5: Post-fork suffix distance against what the fork achieved, pooled over all six checkpoints (2,814 pairs, quintiles of 562–563). Flip rate climbs with distance inside both strata, so the relationship is not a by-product of hard problems being both more divergent and more volatile. The net accuracy gain does not climb with it: wrong→right and right→wrong rise together.

		Flip rate		Correctness		
Quintile	**mean dist.**	**A correct**	**A wrong**	**wrong→right**	**right→wrong**	**net**
Q1 (closest)	0.024	0.076	0.159	0.053	0.060	−0.007
Q2	0.033	0.136	0.177	0.062	0.069	−0.007
Q3	0.040	0.127	0.216	0.078	0.092	−0.014
Q4	0.050	0.208	0.272	0.126	0.115	+0.011
Q5 (farthest)	0.073	0.227	0.250	0.123	0.117	+0.005

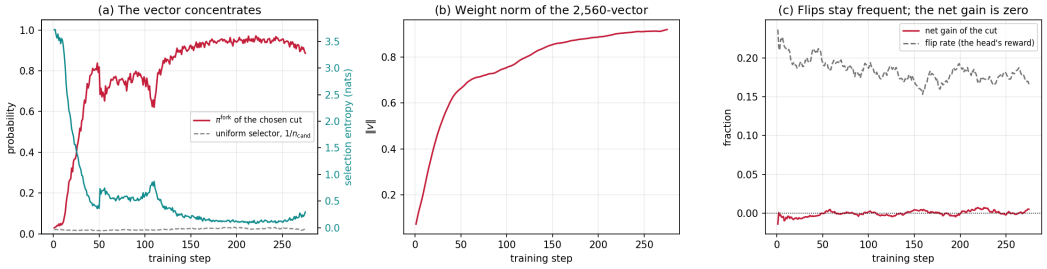

Figure 9: The head over its run. *(a)* The 2,560-vector concentrates: the probability it assigns to the cut it picks rises from 0.03 to 0.94 while its selection entropy falls from 3.72 to 0.12 nats, against a uniform selector's $1/n_{\text{cand}} \approx 0.03$. *(b)* Its weight norm grows by an order of magnitude and then saturates. *(c)* The problem: the flip rate stays high all run while the *net* accuracy gain of the cut, the continuation minus its parent, never leaves the neighbourhood of zero.

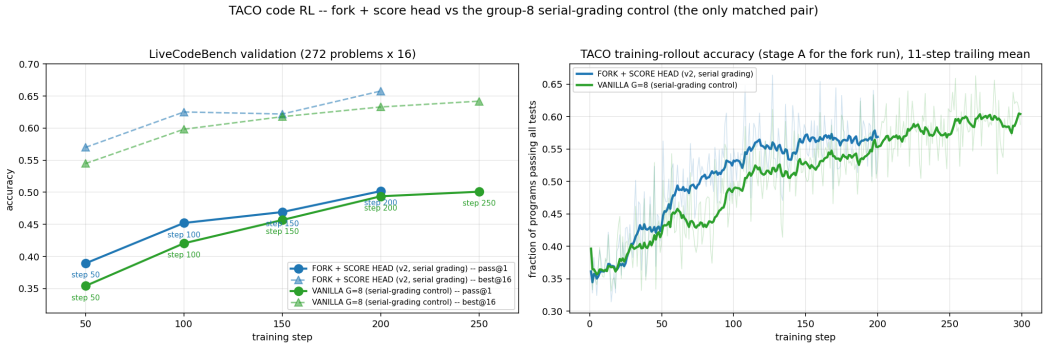

Figure 10: The coding pair of Section 5 on a *training-step* axis: the same number of gradient updates on the same problems for both experiments, differing only in how each group was assembled. The curves are intertwined early, after which the fork experiment leads at every shared validation point, closing to +0.008 pass@1 by the end, having spent 152.6M decoded tokens against the control's 239.3M to get there. Charging the same two runs for what they actually decoded widens the gap.