Effective Reinforcement Learning for Reasoning in Language Models

Anonymous ACL submission

Abstract

Reinforcement learning (RL) has emerged as a promising strategy for improving the reasoning capabilities of language models (LMs) in domains such as mathematics and coding. However, most modern RL algorithms were designed to target robotics applications, which differ significantly from LM reasoning. We analyze RL algorithm design decisions for LM reasoning, for both accuracy and computational efficiency, focusing on relatively small models due to computational constraints. Our findings are: (i) on-policy RL significantly outperforms supervised fine-tuning (SFT), (ii) PPO-based off-policy updates increase accuracy instead of reduce variance, and (iii) removing KL divergence can lead to concise generations and higher accuracy. Furthermore, we find that a key bottleneck to computational efficiency is that the optimal batch sizes for inference and backpropagation are different. We propose a novel algorithm, DASH, that performs *preemptive sampling* (i.e., sample a large batch and accumulate gradient updates in small increments), and *gradient filtering* (i.e., drop samples with small advantage estimates). We show that DASH reduces training time by 83% compared to a standard implementation of GRPO without sacrificing accuracy. Our findings provide valuable insights on designing effective RL algorithms for LM reasoning.[1]

1 Introduction

Recent advancements have shown that reinforcement learning (RL) algorithms can significantly enhance the mathematical reasoning capabilities of language models (LMs) (DeepSeek-AI et al., 2025a; Qwen et al., 2025; Zeng et al., 2025). Despite these results, there has been little systematic understanding of how different RL design decisions contribute to their effectiveness in the LM reasoning setting. Many of these algorithms were originally designed for robotics, while LM reasoning exhibits qualitatively different learning patterns, meaning different design decisions may be more effective (Ahmadian et al., 2024); indeed, even the space of relevant design decisions may be different for LM reasoning compared to robotics. Our goal is to answer the following question:

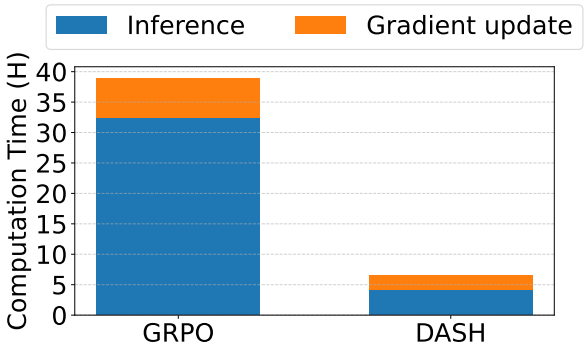


Figure 1: DASH can reduce running time by 83% compared to GRPO by using *preemptive sampling* (Section 3.4) and *gradient filtering* (Section 3.5).

How do we design effective RL algorithms for improving the reasoning capabilities of LMs?

Importantly, we are interested not only in the performance (i.e., the final accuracy), but also efficiency (i.e., how quickly the algorithm converges). Furthermore, we focus on relatively small models (0.5B, 1.5B, and 3B) where we can explore a variety of different RL algorithms.

We perform a systematic analysis of the different design decisions in an RL algorithm. We start by considering the two most prevalent types of algorithms: supervised fine-tuning (SFT) (Chen et al., 2023; Zeng et al., 2023), also known as behavior cloning, and on-policy RL (e.g., policy gradient (Sutton et al., 1999), PPO (Schulman et al., 2017a), GRPO (Shao et al., 2024), etc.). While SFT is much more efficient, we find it to be signifi-

[1]Our code is here: https://anonymous.4open.science/r/efficient_reasoning-A172/README.md.

cantly less effective at improving reasoning ability for the models we consider; this may be due to the inability for smaller models to effectively mimic the reasoning traces of larger models or humans. In contrast, we find that on-policy RL is highly effective at improving performance.

Next, we compare different kinds of on-policy RL algorithms. Compared to the original policy gradient (PG) algorithm, PPO is designed to improve stability by "freezing" the inference policy and taking multiple gradient steps. We find that while PPO achieves a slight increase in performance, it has significantly higher variance compared to PG, which is the opposite of conventional wisdom. In addition, PPO introduces a KL term to regularize the training policy towards the inference policy; perhaps surprisingly, we find that KL divergence leads to lengthier generations and often perform worse than without KL.

While on-policy RL is highly effective, existing algorithms are computationally expensive to run. Analyzing the performance bottlenecks of on-policy RL, we find that the sampling procedure is a key bottleneck. The key issue is that inference and training require significantly different batch sizes to make maximal use of computational resources. Thus, we find that it is much more effective to perform inference in a single large batch, and then accumulate gradient steps for this batch over multiple training steps. This strategy allows us to perform efficient sampling in conjunction with using the PG algorithm. Combined with strategies to filter out samples with small advantage estimates, we call the resulting algorithm Distributed-Aggregated Sampling Handler (DASH); compared to GRPO, it reduces on-policy training time by **83%** without sacrificing model performance (Figure 1). We open-source our DASH implementation to facilitate further research.

To summarize, our key findings are as follows:

- For models we consider, we find on-policy RL to be effective but not SFT (Section 4.2).
- We propose DASH, and find that it can accelerate on-policy training by **83%** without compromising accuracy (Section 4.3).
- We find that while PPO-style gradient updates can slightly improve accuracy, it can introduce instability into training (Section 4.4).
- We find that removing KL divergence can lead to more concise generations and higher accuracies (Section 4.5).

2 Related Work

LM Reasoning. Given the promising performance of language models (LMs), numerous studies have explored their application to mathematical problem solving (Hendrycks et al., 2021; Cobbe et al., 2021; Glazer et al., 2024), program synthesis (Austin et al., 2021; Puri et al., 2021), and other reasoning tasks. Since LMs often exhibit varying performance when directly prompted for these tasks, various methods have been proposed to explicitly elicit reasoning. For instance, Chain-of-Thought prompting (Wei et al., 2023) encourages LMs to generate intermediate reasoning steps before producing the final answer. Tree-of-Thought (Yao et al., 2023a) and Graph-of-Thought (Besta et al., 2024) extend this idea by imposing logical structure to organize the reasoning process. LM reasoning has also been enhanced through tool use (Yao et al., 2023b; Shinn et al., 2023). While these methods have proven effective in guiding LM reasoning and improving downstream task performance, they primarily focus on better prompt design rather than improving the models' inherent reasoning capabilities.

RL for LM reasoning. Recent efforts have focused on using RL to improve LM reasoning capabilities. In question-answering tasks, FireAct (Chen et al., 2023) and AgentTuning (Zeng et al., 2023) enhance reasoning capabilities by learning from demonstrations from humans or stronger models. These approaches are commonly referred to as *supervised fine-tuning* (SFT), or *behavior cloning* in the RL literature. However, several studies have found limits on the effectiveness of SFT, instead proposing to use on-policy RL (DeepSeek-AI et al., 2025a; Shao et al., 2024; Zeng et al., 2025). However, on-policy RL can be very computationally expensive, leading to a great deal of interest in improving efficiency. One shortcoming is that they require re-sampling generations after each model update, leading to sample inefficiency and prolonged training times. To mitigate this, DeepSeek-AI et al. (2025b) propose more efficient transformer architectures to accelerate pre-training, and Kwon et al. (2023a) introduce advanced memory management techniques to speed up sampling in post-training. Although current RL algorithms can leverage vLLM acceleration, the

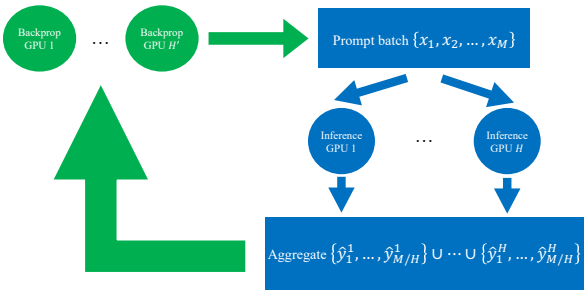


Figure 2: Illustration of preemptive sampling. We use H GPUs for inference and H' for backpropagation; they are shown in blue and green, respectively. Given a batch of M prompts $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$. The inference GPUs then generate corresponding responses $\{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_M\}$, which are aggregated across GPUs into CPU memory. When a backpropagation GPU requests generations for a prompt $\mathbf{x}_m$, the corresponding cached response $\mathbf{y}_m$ is retrieved and delivered. Since we are using groups for advantage estimation, each prompt $\mathbf{x}_m$ is duplicated to form groups, and all generations in the same group are sent to the backpropagation GPU upon request.

full potential of vLLM remains underutilized, leaving significant room for improving RL efficiency.

3 Effective RL for LM Reasoning

First, we describe basic design decisions of our RL algorithm rooted in the prior literature; these are based either on experiments from prior work or our own experiments. Specifically, we consider an LM π_θ with parameters θ, which takes in a user prompt $\mathbf{x}$ and generates a reasoning trace $\hat{\mathbf{y}}$, which we call a *trajectory*. We let $\hat{y}_t$ denote the tth token in trajectory $\hat{\mathbf{y}}$. For a training prompt $\mathbf{x}_n$, we can check whether a generated trajectory $\hat{\mathbf{y}}_n$ produces the correct answer, represented as a scalar reward $r_n = R(\mathbf{x}_n, \hat{\mathbf{y}}_n) \in \mathbb{R}$. We assume that r_n is for the entire trajectory; typically, it is a binary indicator of whether the final answer is correct.[2]

3.1 RL Strategy

The first decision is what kind of RL strategy to use. We consider two strategies: supervised fine-tuning (SFT) and on-policy RL. SFT is effectively the same as behavior cloning, a popular imitation learning algorithm. Given a prompt training set $D = \{\mathbf{x}_n\}_{n=1}^N$, SFT collects corresponding expert trajectories $\mathcal{Y} = \{\mathbf{y}_n\}_{n=1}^N$, either from a human or a stronger LM. Then, the LM is optimized via maximizing the log likelihood on $(D, \mathcal{Y})$:

$$\theta^* = \arg\max_\theta \sum_{n=1}^N \log \pi_\theta(\mathbf{y}_n \mid \mathbf{x}_n)$$

$$\pi_\theta(\mathbf{y}_n \mid \mathbf{x}_n) = \prod_{t=1}^T \pi_\theta(\hat{y}_{n,t} \mid \mathbf{x}_n, \hat{y}_{n,1}, ..., \hat{y}_{n,t-1})$$

Alternatively, on-policy RL learns from trajectories generated by the current LM π_θ. Given the a prompt set D, a typical on-policy RL algorithm optimizes the expected reward:

$$\pi_{\theta^*} = \arg\max_\theta J(\theta) \quad (1)$$

$$J(\theta) = \frac{1}{N} \sum_{x_n \in D} \mathbb{E}_{\hat{\mathbf{y}}_n \sim \pi_\theta(\cdot|\mathbf{x}_n)}[R(\mathbf{x}_n, \hat{\mathbf{y}}_n)].$$

While SFT has been shown to be effective in settings like (Muennighoff et al., 2025) where an already post-trained larger sized model is used (Qwen2.5-32B-Instruct), our experiments show that it can be ineffective when the gap between the expert and π_θ is too large (where we use small base models to ablate on the effect of RL post-training). For instance, an expert may take leaps of reasoning that are incomprehensible to the learner. Thus, DASH uses on-policy RL. Another alternative that has been studied in the literature is self-imitation (Oh et al., 2018), where the "expert" trajectories are obtained by performing search guided by π_θ, but results applying this strategy to LMs have so far been mixed (Shao et al., 2024).

3.2 Gradient Update Strategy

Next, we discuss the gradient update strategy. We consider both policy gradient (PG) (Sutton et al., 1999) and PPO (Schulman et al., 2017b) (which includes GRPO (Shao et al., 2024)). In general, we consider gradient approximations $\nabla_\theta J(\theta) \approx N^{-1} \sum_{n=1}^N J_n$ where J_n encodes the gradient approximation for the nth summand of $J(\theta)$. First, by the Policy Gradient Theorem, using

$$J_n^{\text{PG}} = \mathbb{E}_{\mathbf{y}_n \sim \pi_\theta(\cdot|\mathbf{x}_n)} \left[\frac{\nabla_\theta \pi_\theta(\hat{\mathbf{y}}_n \mid \mathbf{x}_n)}{\pi_\theta(\hat{\mathbf{y}}_n \mid \mathbf{x}_n)} A^{\pi_\theta}(\mathbf{x}_n, \hat{\mathbf{y}}_n) \right] \quad (2)$$

is exact, i.e., $\nabla_\theta J(\theta) = N^{-1} \sum_{n=1}^N J_n^{\text{PG}}$. Here, $A^{\pi_\theta}(\mathbf{x}_n, \hat{\mathbf{y}}_n)$ is the *advantage function*, which we discuss below. This update is truly on-policy since the trajectories $\hat{\mathbf{y}}$ must be sampled using the current policy π_θ. In robotics, PG can be unstable

[2] Recent work has found that process rewards (Wang et al., 2024) may not be effective in our setting due to the difficulty predicting whether a reasoning trace is on the right track (DeepSeek-AI et al., 2025a).

due to high variance when estimating the gradient $\nabla_\theta \pi_\theta(\hat{\mathbf{y}}_n \mid \mathbf{x}_n)$; as a consequence, π_θ can change rapidly across gradient steps, sometimes even becoming worse. PPO was devised to mitigate this instability. Specifically, they weaken the on-policy requirement, and "freeze" the data-generating policy $\pi_{\theta_{\text{old}}}$ for some number of gradient steps. The resulting update has the alternative form

$$J_n^{\text{PPO}} = \mathbb{E}_{\hat{\mathbf{y}}_n \sim \pi_{\theta_{\text{old}}}(\cdot|\mathbf{x}_n)} \left[\frac{\nabla_\theta \pi_\theta(\hat{\mathbf{y}}_n \mid \mathbf{x}_n)}{\pi_{\theta_{\text{old}}}(\hat{\mathbf{y}}_n \mid \mathbf{x}_n)} A^{\pi_{\theta_{\text{old}}}}(\mathbf{x}_n, \hat{\mathbf{y}}_n) \right],$$

where the differences compared to (2) are highlighted in red. Because this gradient is only valid when $\theta \approx \theta_{\text{old}}$, a KL regularization is imposed, to obtain $J_n^{\text{PPO-KL}} = J_n^{\text{PPO}} + \beta J_n^{\text{KL}}$, where

$$J_n^{\text{KL}} = \nabla_\theta D_{\text{KL}}(\pi_{\theta_{\text{base}}}(\cdot \mid \mathbf{x}_n) \parallel \pi_\theta(\cdot \mid \mathbf{x}_n))$$

Following Jaques et al. (2019); Ouyang et al. (2022), the KL divergence term is with respect to the original model $\pi_{\theta_{\text{base}}}$ instead of $\pi_{\theta_{\text{old}}}$ as in PPO.

Critically, in PPO, θ_{old} is updated to be θ every K steps, where K is a hyperparameter. To further improve stability, the gradient is often clipped. GRPO uses the same gradient update as PPO; early versions include a weight $1/\text{len}(\hat{\mathbf{y}}_n)$ on the nth term to normalize by the length of the trajectory, but this term was removed in later versions (Liu et al., 2025). Finally, we note that when $\theta = \theta_{\text{old}}$, this gradient update is equivalent to the PG update (2); this property holds even with gradient clipping.

Now, assume we have sampled a batch of M samples $\{(\mathbf{x}_m, \hat{\mathbf{y}}_m)\}_{m=1}^M$ from $\pi_{\theta_{\text{old}}}$, where initially $\theta = \theta_{\text{old}}$. If we take a single gradient step on all examples, then PPO coincides with PG. This is the strategy used by DASH. We consider two implementations of PPO that do not devolve into PG. First, we can take K gradient steps using all M samples, which we call *PPO-Multi* (or just *Multi*). Second, we can divide the M examples into K mini-batches of size M/K each, and take one gradient step on each mini-batch, which we call *PPO-Mini* (or just *Mini*). In our experiments, we find that DASH is more stable than Multi and Mini, suggesting that the added complexity of PPO-based off-policy gradient updates increases variance.

3.3 Advantage Estimation

A key challenge in RL is estimating the quantity $A^\pi(\hat{\mathbf{y}} \mid \mathbf{x})$, which is called the *advantage* (Sutton and Barto, 2018); it is defined to be $A^\pi(\hat{\mathbf{y}} \mid \mathbf{x}) = Q^\pi(\hat{\mathbf{y}} \mid \mathbf{x}) - V^\pi(\mathbf{x})$, where Q^π is the Q-function and V^π is the value function. Intuitively, it captures how the specific generation $\hat{\mathbf{y}}$ compares to a random sample $\hat{\mathbf{y}}' \sim \pi_\theta(\cdot \mid \mathbf{x})$. In general, A^π is not known and must be estimated from data. We consider three strategies: (i) training a model to predict A^π, (ii) a Monte Carlo estimate called the *single-path method*, and (iii) a Monte Carlo estimate introduced by GRPO. The first approach is to train a model to predict $Q^\pi(\hat{\mathbf{y}} \mid \mathbf{x})$, which can be used to compute V^π and A^π (Schulman et al., 2017a). This approach can reduce variance, but recent work has found that it is highly biased due to the difficulty in predicting Q^π for reasoning tasks (Liang et al., 2022). Thus, we focus on Monte Carlo approaches.

The most popular Monte Carlo approach is the *single-path method*, which uses the estimate

$$A^{\pi_\theta}(\hat{\mathbf{y}}_n \mid \mathbf{x}_n) \approx r_n - \frac{1}{N} \sum_{n'=1}^{N} r_{n'}, \qquad (3)$$

i.e., it is the centered reward; $b = N^{-1} \sum_{n'=1}^N r_{n'}$ is called the *baseline*. Intuitively, r_n is an estimate of the Q-function, and b is an estimate of the value function. A standard modification is to normalize by the standard deviation; this normalization can be useful when rewards tend to increase significantly as learning progresses, but our rewards are bounded so this cannot happen. Another modification is to leave out the reward for rollout n when estimating the value for rollout n, which reduces bias (Sutton and Barto, 2018); this modification can be important when N is small (e.g., $N = 2$) but only has a minor impact for larger N since the bias is small.

A shortcoming of the single-path method is that b is an estimate of the average value $N^{-1} \sum_{n'=1}^N V(\mathbf{x}_{n'})$ across all samples, whereas it ideally should estimate the value $V(\mathbf{x}_n)$. One alternative is the *vine method* (Kazemnejad et al., 2024; Schulman et al., 2017a), which uses a targeted sampling strategy to fix this issue; however, the vine method requires a large number of samples, making it computationally expensive. GRPO uses an advantage estimate that interpolates between the single-path and vine methods. It exploits the fact that in the reasoning setting, we typically train on multiple samples $\hat{\mathbf{y}}_n$ for a single user prompt $\mathbf{x}_n$. In our formulation, we can think of there being multiple $\mathbf{x}_n$ that are identical. Suppose that we partition N into groups $N_1, ..., N_K$, where $\mathbf{x}_n$ is the same for all $n \in N_k$. Then, it estimates the

advantage using the formula

$$A^{\pi_\theta}(\hat{\mathbf{y}}_n \mid \mathbf{x}_n) \approx r_n - \frac{1}{N_k} \sum_{n' \in N_k} r_{n'}, \quad (4)$$

where N_k is the group containing n. In other words, it replaces the baseline with a state-dependent baseline $b(\mathbf{x}_n) = N_k^{-1} \sum_{n' \in N_k} r_{n'}$; now, $b(\mathbf{x}_n)$ is an unbiased estimate of $V(\mathbf{x}_n)$. This strategy can be viewed as performing a vine estimate of the advantage at state $\mathbf{x}_n$, but not at any other state. DASH uses the GRPO advantage estimate in Section 4.

3.4 Preemptive Sampling

A key feature of RL for LMs is that inference typically occurs on specialized inference servers such as vLLM (Kwon et al., 2023b). Importantly, inference is typically much more memory efficient than backpropagation, meaning much larger batches are optimal for inference compared to backpropagation. Empirically, sampling takes up a much larger portion of training time than backpropagation if performed in small batches (Figure 1). Thus, we propose *preemptive sampling*, where we sample a large number of trajectories in one batch, and then perform backpropagation on these samples in smaller batches. Preemptive sampling can be further sped up by using multiple inference servers in parallel (Figure 2). In practice, our method can be used for both on-policy and off-policy sampling, depending on algorithmic design choices, as detailed in Section 3.2. Figure 2 illustrates preemptive sampling. DASH uses preemptive sampling.

3.5 Gradient Filtering

Finally, we propose to drop examples with small advantage estimates (which is equivalent to clipping small advantage values to zero, effectively dropping them from the gradient update). If the advantage estimate is small, then the contribution to the gradient is likely to be small (unless $\nabla_\theta \pi_\theta(\hat{\mathbf{y}}_n \mid \mathbf{x}_n)$ happens to be very large, which we find to be unlikely in practice). Intuitively, these are examples where the model either almost always gets the answer right (in which case there is nothing new to learn) or almost always gets it wrong (in which case the problem is currently too difficult to learn). In addition, even if we only drop advantages that are identically zero, this strategy can provide a speedup since backpropagation still takes time to compute the gradients $\nabla_\theta \pi_\theta(\hat{\mathbf{y}}_n \mid \mathbf{x}_n)$ before they are eventually multiplied by $A^{\pi_\theta}(\hat{\mathbf{y}}_n \mid \mathbf{x}_n) = 0$. DASH uses gradient filtering.

4 Experimental Results

We perform experiments showing that (i) on-policy RL significantly outperforms SFT (Section 4.2, (ii) DASH significantly reduces running time compared to standard GRPO (Section 4.3), (iii) PG gradient updates outperform PPO-based gradient updates (Section 4.4), and (iv) removing KL divergence can lead to concise generations and higher accuracies (Section 4.5).

4.1 Experimental Setup

We use Qwen2.5-{0.5B, 1.5B, 3B} models as our base models, all of which are not post-trained (i.e., no instruction tuning). We use the MATH dataset (Hendrycks et al., 2021), with the MATH-500 split (Lightman et al., 2023), which contains 12,000 examples for training and 500 examples for evaluation. We additionally use the GSM8K dataset (Cobbe et al., 2021) for out-of-distribution evaluation, which contains 1,319 examples. Finally, we also perform some experiments in the coding domain using the MBPP+ dataset (Liu et al., 2023), a 378-problem subset of verified problems from the MBPP dataset (Austin et al., 2021); we use 264 problems for training and 114 for evaluation. Additional details are provided in Table 7.

4.2 SFT vs. On-Policy RL

We compare three algorithms: (i) SFT with human-written reasoning traces, denoted *SFT-H*, (ii) SFT with reasoning traces from Qwen2.5-7B-Instruct, denoted *SFT-M*, and (iii) DASH. Results are shown in Table 1. As can be seen, DASH improves performance both in-distribution and out-of-distribution, demonstrating that on-policy algorithms can efficiently learn mathematical reasoning skills that generalize across datasets. On the other hand, neither SFT-H nor SFT-M improve performance, with SFT-H significantly degrading both in-distribution and out-of-distribution performance. Intuitively, the substantial performance degradation caused by SFT-H can be attributed to the fact that human reasoning often omits many intermediate steps, which is especially problematic for smaller LMs.

For coding, we train on human programs in MBPP+. Results are shown in Table 2. As can be seen, DASH outperforms SFT in most cases, demonstrating the the general effectiveness of on-policy RL at improving the reasoning capabilities of LMs. To the best of our knowledge, these are among the first results to show that on-policy RL

Method	Size (B)	MATH (%)	GSM8K (%)
Base	0.5	22.6	30.3
	1.5	48.0	58.8
	3.0	58.8	66.0
SFT-H	0.5	8.0	7.2
	1.5	17.2	32.8
	3.0	24.0	30.6
SFT-M	0.5	24.0	22.7
	1.5	46.2	46.0
	3.0	53.0	66.0
DASH	0.5	27.2	31.1
	1.5	54.8	56.0
	3.0	63.4	65.9

Table 1: Comparison of SFT to on-policy RL on math.

Method	Size (B)	pass1 (%)	pass@8 (%)
BASE	0.5	2.6	22.8
	1.5	7.1	60.5
SFT-H	0.5	8.77	29.0
	1.5	19.3	42.1
DASH	0.5	11.4	40.4
	1.5	23.7	63.2

Table 2: Comparison of SFT to on-policy RL on coding.

can improve code generation for smaller LMs.

4.3 DASH vs. GRPO

Next, we compare DASH to GRPO both in terms of accuracy and running time. Specifically, we train Qwen2.5-0.5B using both GRPO and DASH. We also use an ablation of DASH without gradient filtering, denoted *No-GF*. Results are shown in Table 3 and illustrated in Figure 1. As can be seen, DASH significantly reduces GRPO training time (from 39 hours to 6.6 hours) without any significant reduction in performance, highlighting the effectiveness of preemptive sampling and gradient filtering. Compared to No-GF, DASH reduces running time by 4% without any significant reduction in performance. The effectiveness of gradient filtering can be improved; see Appendix B. We additionally show results for coding in Table 4. For coding, we again see a significant speedup, though it is smaller since the generation length is much smaller so the gap in optimal inference and backpropagation batch sizes is smaller.

Method	Time (h)	MATH (%)	GSM8K (%)
BASE	N/A	22.6	30.3
GRPO	38.9	27.6	32.8
No-GF	6.9	27.4	31.6
DASH	6.6	27.2	31.1

Table 3: Comparing on-policy RL algorithms on Qwen2.5-0.5B for math.

Method	Time (m)	pass@1 (%)	pass@8 (%)
BASE	N/A	2.3	22.8
GRPO	35.3	11.4	49.1
No-GF	16.3	10.5	43.9
DASH	16.5	11.4	40.4

Table 4: Comparing on-policy RL algorithms using Qwen2.5-0.5B for coding.

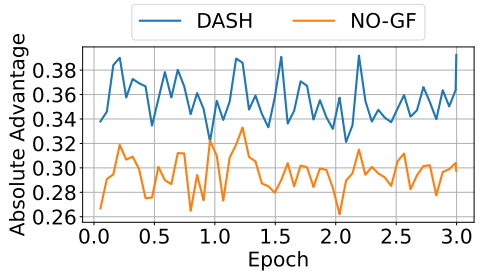
The impact of GF on training dynamics is illustrated in Figure 3. Specifically, as shown in Figure 3a, gradient filtering increases the average absolute advantage values, leading to more significant gradient updates; consequently, as shown in Figure 3b, forward and backward pass running times are reduced. Finally, since only samples inducing trivial gradient updates are filtered out, the training curves remain similar before and after applying gradient filtering, as shown in Figure 3c.

4.4 PG vs. PPO Gradient Updates

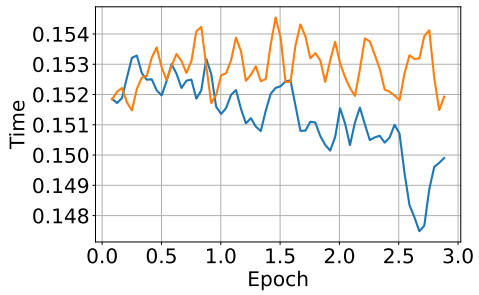
Next, we compare DASH to Multi and Mini. DASH uses a batch size of $M = 256$ (with $K = 1$), Multi uses $M = 256$ and $K = 3$, and Mini uses $M = 8$ so $K = 32$. Multi and Mini are slower than DASH; for fair comparison, we truncate their training times to match the wall-clock time of DASH. The results are shown in Table 5, and training curves are shown in Figure 4. As can be seen, Multi and Mini achieve faster initial performance improvements and have slightly higher accuracies; however, they have significantly more unstable training curves. Similar results for the 1.5B model are shown in Appendix B.

4.5 KL Divergence Regularization

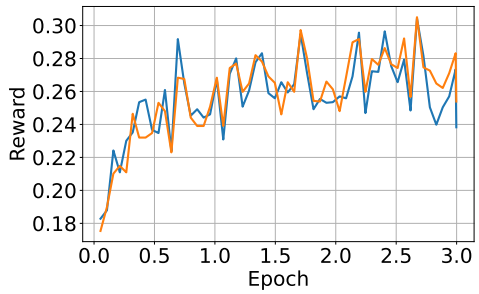
Next, we compare DASH to an ablation without the KL divergence term, denoted *No-KL*. Training rewards are shown in Figure 6a. As can be seen, removing KL divergence regularization generally leads to higher rewards during training; most likely, No-KL can focus on reward optimization without



(a) Average absolute advantage values.



(b) Loss computation time.



(c) Training reward recurves.

Figure 3: Comparison between DASH and No-GF for Qwen2.5-0.5B on math.

being constrained to stay close to the initial model. As shown in Table 6, No-KL leads to greater improvements in- and out-of-distribution upon the base model compared to DASH with KL (save one out-of-distribution result for the 3B model, which DASH also underperforms).

Furthermore, as shown in Figure 6b, we find that for No-KL, the average generation length is shorter, thereby reducing overall training time; this difference is also reflected in Table 6. We hypothesize that to compensate for KL divergence regularization, models must generate longer reasoning traces.

Finally, for the 3B model, we study how KL divergence regularization affects pass@k. We follow Chen et al. (2021) to evaluate pass@k in an unbiased way. Results are in Figure 5: No-KL performs best for small k, although the gap closes for larger k. Intuitively, RL concentrates probability mass and reduces generation diversity (Shypula et al., 2025; West and Potts, 2025).

Method	Size (B)	MATH (%)	GSM8K (%)
DASH	0.5	27.2	31.1
	1.5	53.0	58.9
Multi	0.5	28.8	31.5
	1.5	54.0	61.0
Mini	0.5	29.8	31.6
	1.5	36.0	8.1

Table 5: Comparing PG to PPO for math.

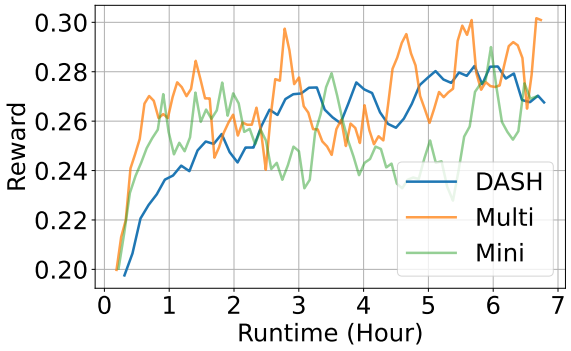


Figure 4: Training reward curves for PG vs. PPO on Qwen2.5-0.5B for math.

5 Conclusion

We have performed a careful empirical analysis of several of the key design decisions in RL algorithms for improving language model reasoning, with a particular focus on computationally constrained scenarios; these include SFT vs. on-policy RL, policy gradient vs. PPO, and whether KL divergence regularization is used. Furthermore, we identify the sampling strategy as the primary computational bottleneck in on-policy RL; to address these issues, we propose DASH, a novel algorithm using preemptive sampling and gradient filtering to improve efficiency. We demonstrate that DASH can reduce RL training time by 83% while maintaining performance. More broadly, we believe that systematizing the study of RL for language model reasoning is key to designing more effective RL algorithms in this domain, which differs significantly from robotics domains targeted by existing RL algorithms such as PPO. Our study is a first step in this direction.

Limitations. Due to computational constraints, we do not perform extensive hyperparameter tuning, instead adopting commonly used values; bet-

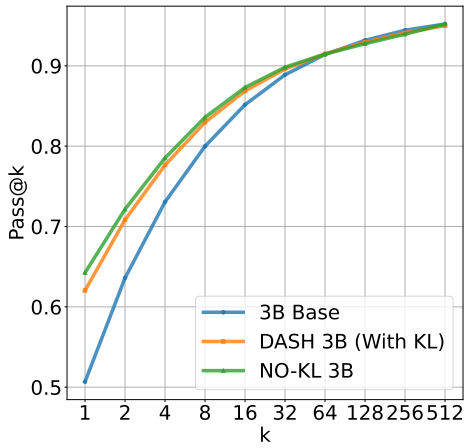
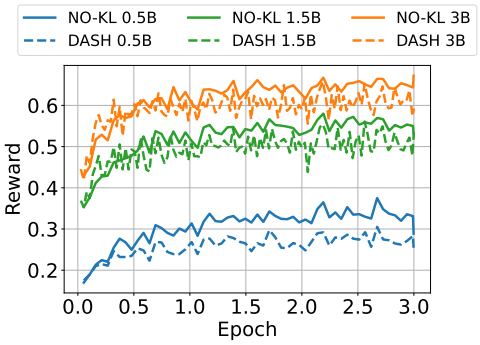


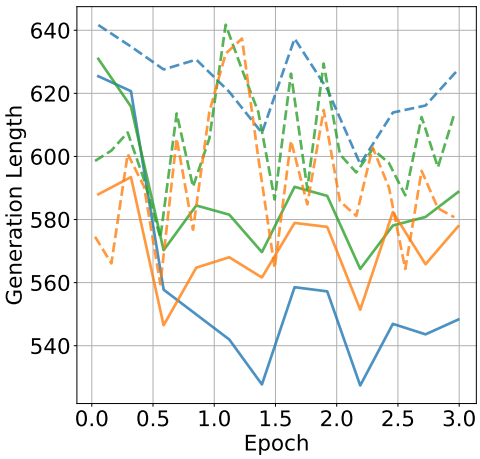
Figure 5: Impact of KL divergence regularization on pass@k for Qwen2.5-3B on math.

Method	Size (B)	Time (h)	MATH (%)	GSM8K (%)
Base	0.5	N/A	22.6	30.3
	1.5	N/A	48.0	58.8
	3.0	N/A	58.8	66.0
DASH (with KL)	0.5	6.6	27.2	31.1
	1.5	12.8	54.8	56.0
	3.0	22.6	63.4	65.9
No-KL	0.5	5.7	31.4	34.0
	1.5	10.3	56.8	62.1
	3.0	17.6	66.4	60.0

Table 6: Comparing No-KL to DASH and Base on math.



(a) Training reward curves.



(b) Generation length.

Figure 6: Comparing KL divergence regularization on math.

ter hyperparameter choices could further improve performance. Also, we focus on relatively small Qwen2.5 models, and our findings may not generalize to larger models or other architectures.

Ethics statement. Our paper aims to design better RL algorithms for reasoning in LMs; we do not foresee any ethical concerns beyond standard ethical issues with reasoning in LMs.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *Preprint*, arXiv:2402.14740.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *Preprint*, arXiv:2108.07732.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *Preprint*, arXiv:2310.05915.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang,

Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025b. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, Olli Järviniemi, Matthew Barnett, Robert Sandler, Matej Vrzala, Jaime Sevilla, Qiuyu Ren, Elizabeth Pratt, Lionel Levine, Grant Barkley, and 5 others. 2024. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *Preprint*, arXiv:2411.04872.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *Preprint*, arXiv:2103.03874.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. 2019. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *Preprint*, arXiv:1907.00456.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. 2024. Vineppo: Unlocking rl potential for llm reasoning through refined credit assignment. *Preprint*, arXiv:2410.01679.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023a. Efficient memory management for large language model serving with pagedattention. *Preprint*, arXiv:2309.06180.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023b. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Litian Liang, Yaosheng Xu, Stephen McAleer, Dailin Hu, Alexander Ihler, Pieter Abbeel, and Roy Fox. 2022. Reducing variance in temporal-difference value estimation via ensemble of deep networks. *Preprint*, arXiv:2209.07670.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *Preprint*, arXiv:2501.19393.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. 2018. Self-imitation learning. *Preprint*, arXiv:1806.05635.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. 2021. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *Preprint*, arXiv:2105.12655.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. *Preprint*, arXiv:1910.02054.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2017a. Trust region policy optimization. *Preprint*, arXiv:1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017b. Proximal policy optimization algorithms. *Preprint*, arXiv: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. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Alexander Shypula, Shuo Li, Botong Zhang, Vishakh Padmakumar, Kayo Yin, and Osbert Bastani. 2025. Evaluating the diversity and quality of llm generated content. *Preprint*, arXiv:2504.12522.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *Preprint*, arXiv:2312.08935.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Peter West and Christopher Potts. 2025. Base models beat aligned models at randomness and creativity. *Preprint*, arXiv:2505.00047.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Preprint*, arXiv:2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *Preprint*, arXiv:2310.12823.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. 2025. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *Preprint*, arXiv:2503.18892.

A Additional Experimental Setup

Math. All GRPO experiments are conducted using 6 Nvidia A6000 GPUs; we use 4 GPUs for backpropagation and 2 for inference across all three model sizes (Qwen2.5-{0.5B, 1.5B, 3B}) (in practice, the 0.5B model only needs 2 GPUs for backpropagation, but we still use 4 for consistency). Our implementation is based on Huggingface's GRPO Trainer; the hyperparameters are as follows:

- Learning rate: 1e-06 for the 0.5B model and 3e-06 for the 1.5B and 3B models
- Backpropagation batch size per GPU: 2 (so batch size is 8)
- # generations per prompt: 4 (resulting in 2 prompts backpropagated on in each step)
- Maximum completion length: 2048
- Inference batch size for DASH: 256 (128 per inference GPU); for comparing to Multi and Mini on Qwen2.5-1.5B, we use 1024 (512 per inference GPU)
- Gradient accumulation steps for DASH: 32 (so the effective batch size is 256); for comparing to Multi and Mini on Qwen2.5-1.5B, we use 128 (so the effective batch size is 1024)
- Gradient steps per batch for Multi: 3
- Batch size for Mini: 8 (equivalently, no gradient accumulation)
- Gradient filtering threshold: 0.1

All other parameters are set to the default of the Huggingface trainer; a summary of the GRPO hyperparamaters is in Table 7. To reduce memory footprint, we use DeepSpeed (Rajbhandari et al., 2020) ZeRO Stage 3 as well as CPU offload, gradient clipping, and mixed precision; our DeepSpeed configuration is shown in Figure 7.

Parameters for SFT are shown in Table 8. All SFT experiments use end-to-end fine-tuning instead of using parameter efficient methods such as LoRA (Hu et al., 2021). For model-generated reasoning traces, we use Qwen2.5-7B-Instruct as the teacher to keep the distribution of generations in the Qwen family. We use a temperature of 0.7 and filter out reasoning traces with the wrong answer. The resulting training set has 8,955 examples.

```
compute_environment: LOCAL_MACHINE
debug: false
deepspeed_config:
  gradient_clipping: 1.0
  offload_optimizer_device: cpu
  offload_param_device: cpu
  zero3_init_flag: false
  zero3_save_16bit_model: false
  zero_stage: 3
distributed_type: DEEPSPEED
downcast_bf16: 'no'
enable_cpu_affinity: false
machine_rank: 0
main_training_function: main
mixed_precision: bf16
num_machines: 1
num_processes: 4
rdzv_backend: static
same_network: true
tpu_env: []
tpu_use_cluster: false
tpu_use_sudo: false
use_cpu: false
```

Figure 7: DeepSpeed configuration.

Versions of python and key libraries are shown in Table 9. The dev version of trl was cloned directly from trl's GitHub repository on April 10, 2025.

Coding. All experiments with MBPP+ on coding using on-policy RL for Qwen2.5-0.5B were conducted on AWS EC2 g6.12xlarge instances with 48 vCPUs, 192 GiB memory, and 4 NVIDIA L4 Tensor Core GPUs with 96 GiB total GPU memory, with 2 GPUs dedicated to training and 2 to sampling. Experiments with Qwen2.5-1.5B and Qwen2.5-3B were conducted on AWS EC2 g6e.12xlarge instances with 48 vCPUs, 384 GiB memory, and 4 NVIDIA L40S Tensor Core GPUs with 192 GB total GPU memory, with 2 GPUs dedicated to training and 2 to sampling. All SFT experiments on MBPP+ were conducted using 2 NVIDIA A6000 GPUs. The hyperparameters for coding are the same as for math.

B Additional Experiment Results

We compare gradient filtering with larger batch sizes, finding that gradient filtering is more effective when the per device batch size is 4 (instead of 2). This experiment is only possible for the for Qwen2.5-0.5B on the math dataset using our compute. Results are shown in Table 10 and training curves are shown in Figure 9. The time reduction achieved is larger than before (10% instead of 4%). These results suggest that gradient filtering may become more effective with larger batch sizes.

Hyperparameter	Qwen2.5-0.5B	Qwen2.5-1.5B	Qwen2.5-3B
NVIDIA A6000 GPUs (training / sampling)	4 / 2 (2 / 2 using ZeRO)	4 / 2	4 / 2
Learning rate	1×10^{-6}	3×10^{-6}	3×10^{-6}
Epochs	3	3	3
Batch size per device	2	2	2
Generations per prompt	4	4	4
Max completion length (tokens)	2048	2048	2048
Gradient accumulation steps for DASH based runs	32	32	32
Gradient accumulation steps for Multi and Mini	32	128	N/A
Gradient steps per sampled batch for Multi	3	3	N/A
Gradient-filtering threshold	0.1	0.1	0.1
Normalize gradients by generation length?	No	No	No

Table 7: Experimental configuration and hyperparameters for on-policy RL on MATH.

Parameter	Value
Learning rate	2×10^{-5}
Epochs	3
Batch size per device	4
Gradient accumulation steps	2

Table 8: Experimental configuration and hyperparameters for SFT.

Package	Version
python	3.11.11
trl	0.17.0.dev0
vllm	0.8.1
pytorch	2.6.0

Table 9: Package versions.

We also show the comparison of Mini, Multi, and DASH on the 1.5B model (Figure 8). Conclusions are similar to Section 4.4. In this case we stabilize Multi by increasing the gradient accumulation step to 128, but the high instability of Mini leads to decrease in training rewards as well as accuracies as shown in Table 5.

Method	Time (h)	MATH (%)	GSM8K(%)
No-GF	5.1	31.8	31.3
DASH	4.6	28.4	30.9

Table 10: Comparing No-GF to DASH with per device batch of 4 instead of 2 for Qwen2.5-0.5B on math.

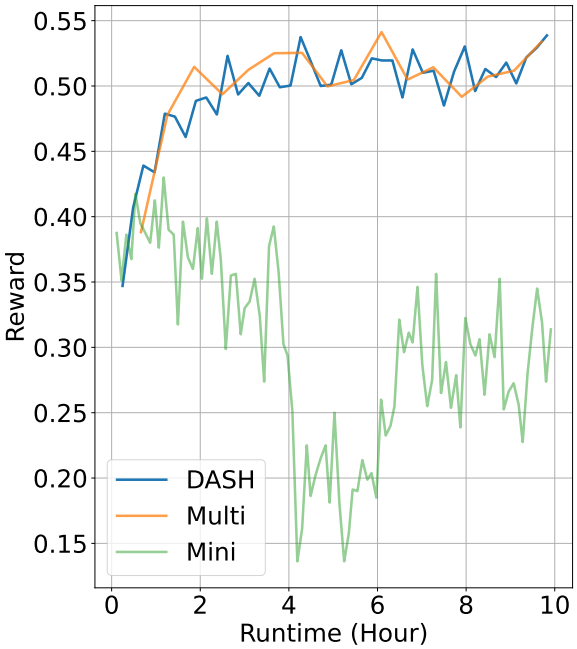
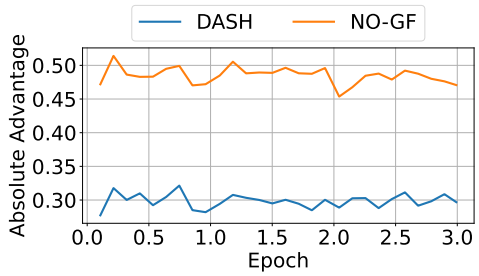
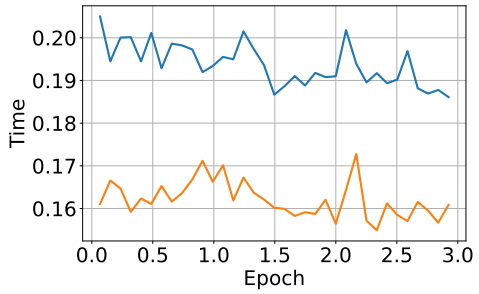


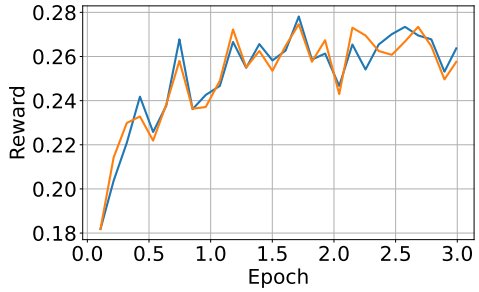
Figure 8: Training reward curves for PG vs. PPO on Qwen2.5-0.5B for math.



(a) Average absolute advantage values.



(b) Loss computation time.



(c) Training reward curves.

Figure 9: Comparison of No-GF and DASH with a per-device batch size of 4 for Qwen2.5-0.5B on math.