# LLMs CAN LEARN TO REASON VIA OFF-POLICY RL

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

Reinforcement learning (RL) approaches for Large Language Models (LLMs) frequently use on-policy algorithms, such as PPO or GRPO. However, policy lag from distributed training architectures and differences between the training and inference policies break this assumption, making the data off-policy by design. To rectify this, prior work has focused on making this off-policy data appear more on-policy, either via importance sampling (IS), or by more closely aligning the training and inference policies by explicitly modifying the inference engine. In this work, we embrace off-policyness and propose a novel off-policy RL algorithm that does not require these modifications: *Optimal Advantage-based Policy Optimization with Lagged Inference policy* (OAPL). We show that OAPL is significantly more stable and performant than GRPO with importance sampling across several math reasoning benchmarks when using stale data. We also show that OAPL can match the performance of a publicly available coding model, DeepCoder, on LiveCodeBench, while using 3x fewer generations during training. We further empirically demonstrate that models trained via OAPL improve test time scaling under the Pass@k metric. OAPL allows for efficient, effective post-training even with lags of more than 400 gradient steps between the training and inference policies, 100x more off-policy than prior approaches.

## 1 INTRODUCTION

End-to-end optimization of Large Language Models (LLMs) via Reinforcement Learning (RL) unlocks LLMs' reasoning capabilities. DeepSeek-R1 (Guo et al., 2025) is one representative work demonstrating that reasoning capabilities emerge naturally via large-scale RL optimization. Since DeepSeek-R1, the literature has mainly focused on improving the training stability of Group Relative Policy Optimization (Shao et al., 2024) (GRPO) – the RL method that powers the post-training of DeepSeek-R1.

A central reason training stability is hard to achieve in practice is that modern RL post-training infrastructures are often not truly on-policy. In particular, the trainer (e.g. a HuggingFace model (Wolf et al., 2020)) and the inference engine (e.g. a vLLM model (Kwon et al., 2023)) may produce different log-probabilities for the same sequence, even when both models have the same weights. This mismatch can arise because of differences between the trainer and inference engine kernel implementations (Yao et al., 2025; Liu et al., 2025b; Team et al., 2025b), or because of asynchronous training pipelines, where the inference engine may contain an older version of the trainer's weights (Fu et al., 2025).

This discrepancy in log-probabilities makes practical policy-gradient training effectively **off-policy** because the data used to optimize the current policy is not generated by that policy. In contrast, classic policy gradient methods (e.g., REINFORCE (Williams, 1992)), from which modern policy optimization methods like GRPO and its predecessor PPO (Schulman et al., 2017) are derived, work under the assumption that sampling is **on-policy**, meaning that the data is generated from the current policy being optimized.

Most improvements to GRPO thus focus on making it as on-policy as possible despite the gap between the trainer and the inference engine. Existing work generally addresses this problem either by introducing additional importance weights (Zheng et al., 2025a; Yao et al., 2025; Fu et al., 2025)



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.

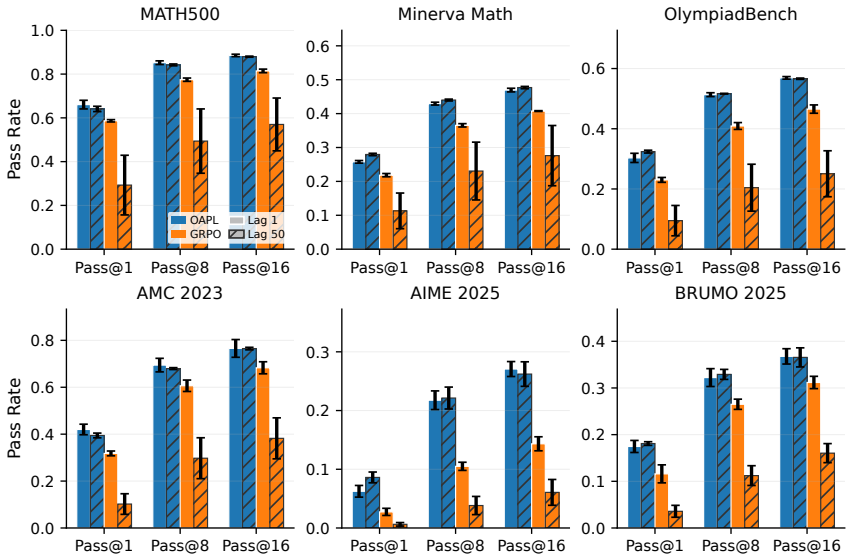



Figure 1: OAPL and GRPO on math reasoning benchmarks. Bars show the average performance after 1000 steps of training over 3 seeds, with error bars indicating standard error. Striped bars show performance when training with a policy lag of 50 ($L = 50$). We report Pass@1, Pass@8, and Pass@16 on MATH500, Minerva Math, OlympiadBench, AMC 2023, AIME 2025, and BRUMO 2025

or by modifying the inference engine to reduce its mismatch with the trainer (Qi et al., 2025; Wasti et al., 2025).

While both families demonstrate promising results, we argue that neither is ideal. In the first approach, adding importance weights to the GRPO objective introduces extra variance to the RL loss function. The second approach makes the inference engine slower and does not fully close the gap between the inference engine and the trainer in asynchronous RL training. In this work, we ask the following questions for RL post-training of LLMs:

**Are on-policy algorithms necessary for RL post-training?**
**Can we develop simple and scalable off-policy RL algorithms?**

We find that being on-policy is not necessary for RL post-training, and we propose an easy to implement and effective off-policy post-training algorithm: *Optimal Advantage-based Policy Optimization with Lagged Inference policy*, abbreviated OAPL. We treat the mismatch between the trainer and inference engine policies as a KL-regularized RL problem, where the KL term explicitly prevents the training policy from moving too far away from the inference policy. Leveraging the closed-form solution of KL-regularized RL, we derive a squared regression objective that trains on rollouts from a lagged inference policy, eliminating the need for on-policy sampling. OAPL then uses that objective in an iterative procedure that very infrequently syncs the trainer and inference policy, enabling training that is significantly more off-policy than other approaches. OAPL fully embraces off-policy training without any importance weighting ratios. Our view that on-policy learning is not necessary for RL post-training is consistent with classical RL results, where on-policy policy gradient methods such as PPO and REINFORCE are often less sample efficient than off-policy algorithms such as DDPG and SAC (Lillicrap et al., 2015; Haarnoja et al., 2018) on traditional robotics control and video game benchmarks.

Empirically, we find that OAPL can outperform a GRPO-based baseline on six math reasoning benchmarks (MATH500, OlympiadBench, MinervaMath, AMC2023, AIME2025, and BRUMO2025)

across multiple Pass@k metrics[1] (see Figure 1). On LiveCodeBench v5, across various Pass@k metrics, our approach can match or outperform DeepCoder (Luo et al., 2025a), which is trained via GRPO with additional heuristics including clip-high, overlong filtering, etc, while using approximately **one third the number of generations for training**. Notably, in our code generation experiments, the policy lag (off-policyness) can be as large as 400 gradient updates without the need for any importance sampling. OAPL also stably improves the Pass@k test-time scaling metrics for k ranging from 1 to 16. Overall, we demonstrate that being on-policy is not necessary and embracing off-policy learning can result in stable, effective, and efficient training for reasoning LLMs.

## 2 BACKGROUND

In modern RL post-training, there are generally two types of policies: the trainer $\pi$ and the inference engine $\pi_{\text{vllm}}$[2]. The trainer $\pi$ is used to compute gradient updates given generated sequences, while the inference engine $\pi_{\text{vllm}}$ is used for fast generation. However, even when $\pi$ and $\pi_{\text{vllm}}$ share the same weights, they can output different log-probabilities given the same sequence of tokens. This inherent difference in the log-probabilities from $\pi$ and $\pi_{\text{vllm}}$ breaks the on-policy assumption of policy gradient-based methods. For example, Liu et al. (2025b) measure the KL divergence between the inference engine and trainer, and found that sudden increases in that divergence contributed to training instability and policy collapse in GRPO. The gap between the inference engine and trainer can be further enlarged in an asynchronous RL training framework (e.g., $\pi_{\text{vllm}}$ could be multiple gradient steps behind the trainer $\pi$).

One common way to handle off-policy rollouts in LLM post-training—and the primary baseline we compare against—is standard importance sampling (IS), applied either at the token level (Fu et al., 2025) or the sequence level (Zheng et al., 2025a). Given any prefix $x$ and next token $a$ sampled from $\pi_{\text{vllm}}(\cdot|x)$, IS computes the likelihood ratio $\frac{\pi(a|x)}{\pi_{\text{vllm}}(a|x)}$ and uses it to reweight the GRPO loss function before averaging across a batch of examples. These likelihood ratios aim to correct the mismatch caused by action $a$ being generated from $\pi_{\text{vllm}}$ instead of $\pi$. For example, we can formulate GRPO as an importance weighted loss function:

$$\mathbb{E}_{\{y_i\}_{i=1}^{G}\sim\pi_{\text{vllm}}(\cdot|x)}\left[\frac{1}{G}\sum_{y\in y_1\ldots y_G}\frac{1}{|y|}\sum_{t=1}^{|y|}\underbrace{\frac{\pi_{\text{old}}(y_t|x,y_{<t})}{\pi_{\text{vllm}}(y_t|x,y_{<t})}}_{\text{IS ratio}}\cdot\min\left\{r_tA_t,\text{clip}(r_t,1-\epsilon,1+\epsilon)A_t\right\}\right]$$

where $\pi_{\text{old}}$ is the previous iteration of the trainer, $r_t = \frac{\pi(y_t|x,y_{<t})}{\pi_{\text{old}}(y_t|x,y_{<t})}$ is the PPO-style likelihood ratio, and $A_t$ is the normalized advantage. Fu et al. (2025) introduced this loss function for their asynchronous RL training framework, where the data generation policy $\pi_{\text{vllm}}$ could lag behind the current training policy $\pi$. The additional token-level ratio $\frac{\pi_{\text{old}}(y_t|x,y_{<t})}{\pi_{\text{vllm}}(y_t|x,y_{<t})}$ reweights tokens $y_t$ sampled from $\pi_{\text{vllm}}$ as if they were generated by $\pi_{\text{old}}$.

However, IS can become unreliable when the behavior and target policies differ substantially, motivating a great deal of prior work on variance-reduction techniques (Munos et al., 2016; Mahmood & Sutton, 2015; Mahmood et al., 2015; Hallak et al., 2015; Geist & Scherrer, 2013). Empirically, prior work has tried many additional heuristics such as clipping IS ratios, explicitly deleting tokens from the GRPO objective whose IS ratio is too large or too small, or throwing away entire rollouts that are too off-policy. While these heuristics stabilize the GRPO training specifically, they deviate more and more from the principles behind classic policy gradient theory. Since these heuristics are specifically designed for and tested under the GRPO loss, it is unclear how they generalize beyond the very specific GRPO loss function. In this work, instead of focusing on modifying GRPO's loss, we take a different route, and design a new RL training objective that works naturally with off-policy data.

[1]To compute these metrics, we sample 32 independent rollouts per prompt. Pass@k is then computed using the unbiased estimator from (Chen et al., 2021). For our code generation experiments, we sample 20 independent rollouts per prompt and use the same estimator.

[2]We use vLLM as our example inference engine throughout the text

# 3 METHOD: OAPL

We introduce *Optimal Advantage-based Policy Optimization with Lagged Inference policy* (OAPL), a principled off-policy objective that remains stable under substantial policy lag. Unlike prior approaches that require inference engine customization or GRPO variants augmented with extra ratios, clipping operators, or deletion of stale tokens/sequences, we embrace the off-policy nature of RL post-training and design a simple, fully off-policy RL algorithm.

## 3.1 OFF-POLICY LOSS FUNCTION

We first introduce our off-policy policy optimization objective, motivated by the KL-regularized RL formulation. Consider the following objective:

$$\max_{\pi} \mathbb{E}_{x,y\sim\pi(\cdot|x)} r(x,y) - \beta \mathrm{KL}(\pi||\pi_{\text{vllm}})$$

The goal of this objective is to maximize the reward $r$ while at the same time minimizing the KL to the inference policy $\pi_{\text{vllm}}$.[3] It is well known that the optimal policy $\pi^\star$ and the optimal value function $V^\star$ of the above KL-regularized RL formulation have the closed-form expressions $\pi^\star(y|x) \propto \pi_{\text{vllm}}(y|x)\exp(r(x,y)/\beta)$ and $V^\star(x) = \beta \ln \mathbb{E}y \sim \pi_{\text{vllm}}(\cdot|x)[\exp(r(x,y)/\beta)]$. Rearranging terms, we can express the relationship between $\pi^\star$ and $V^\star$ as follows:

$$\beta \ln \frac{\pi^\star(y|x)}{\pi_{\text{vllm}}(y|x)} = \underbrace{r(x,y) - V^\star(x)}_{\text{optimal advantage } A^\star}, \forall x, y.$$

Crucially, the expectation defining $V^\star$ is taken under the sampling policy $\pi_{\text{vllm}}$, not $\pi^\star$. Thus, given $x$ and a group of $G$ rollouts $\{y_1, \ldots, y_G\}$ sampled from $\pi_{\text{vllm}}(\cdot|x)$, Brantley et al. (2025) proposes estimating $V^\star$ by:

$$\hat{V}^\star(x) = \beta \ln \frac{1}{G} \sum_{i=1}^{G} \exp(r(x,y_i)/\beta). \tag{1}$$

The estimator $\hat{V}^\star$ can be accurate under mild assumptions on the sampling distribution $\pi_{\text{vllm}}$. In particular, for a binary reward, if $\pi_{\text{vllm}}$ has a non-zero probability of solving $x$, then $\hat{V}^\star(x)$ converges to $V^\star(x)$ as $G$ increases (Brantley et al., 2025; Zhou et al., 2025). The role of $\beta$ here is *smoothing*: when $\beta \to 0$, we have $\hat{V}^\star(x) = \max_i r(x,y_i)$, and when $\beta \to \infty$, $\hat{V}^\star(x) = \sum_i r(x,y_i)/G$ becomes the average which is an unbiased estimate of the average reward of the current inference policy $\pi_{\text{vllm}}$.

Given $\hat{V}^\star$, we can estimate the optimal advantage $A^\star(x,y)$ as $r(x,y) - \hat{V}^\star(x)$. We adopt the $A^\star$PO objective from Brantley et al. (2025) and define the following policy optimization objective:

$$\min_{\pi} \sum_{x} \sum_{i=1}^{G} \left( \beta \ln \frac{\pi(y_i|x)}{\pi_{\text{vllm}}(y_i|x)} - (r(x,y_i) - \hat{V}^\star(x)) \right)^2 \tag{2}$$

When $\hat{V}^\star = V^\star$, Eq. 2 is minimized by the KL-regularized optimum $\pi^\star$, regardless of the sampling distribution of $y$ (e.g., it holds for rollouts drawn from $\pi_{\text{vllm}}$). While our loss function is motivated by $A^\star$PO, $A^\star$PO was designed to be an on-policy algorithm, i.e., it formulates the above optimization under an on-policy dataset generated from $\pi$. We instead rely on the objective's unique minimizer, and use the off-policy data and log-probabilities from the inference engine directly.

As motivated by the original $A^\star$PO paper, estimating $\hat{V}^\star$ from groups of rollouts allows us to avoid making extra assumptions, such as $V^\star$ being approximated by a constant (Zhu et al., 2024) or having to use another neural network to model $V^\star$, which can be computationally expensive (Richemond et al., 2024).

## 3.2 OAPL: THE OFF-POLICY RL ALGORITHM

We convert Eq. 2 into a practical post-training pipeline with a lagged inference engine. This yields *Optimal Advantage-Based Policy Optimization with Lagged Inference Policy* (Algorithm 1),

[3] Note $\pi_{\text{vllm}}$ is not necessarily the reference policy. We use $\pi_{\text{vllm}}$ to denote the current inference policy which could share the weights from the trainer. We emphasize that this is not the usual KL regularization to $\pi_{\text{ref}}$.

**Algorithm 1** Optimal Advantage-Based Policy Optimization with Lagged Inference Policy (OAPL)

**Require:** Policy model $\pi$, Inference engine $\pi_{\text{vllm}}$, Data buffer $\mathcal{D}$, Policy lag $L$
  Synchronize $\pi$ and $\pi_{\text{vllm}}$
  **for** $t = 1 \to T$ **do**
    Sample a batch $\{x, \{y_i\}_{i=1}^{G}\}$ from $\pi_{\text{vllm}}$ and store it in $\mathcal{D}$; ▷ # Data generation (can be async)
    Optimize policy $\pi$ using data from $\mathcal{D}$ via gradient descent on Eq. 2; ▷ # Off-policy update (can be async)
    **if** $t \bmod L = 0$ **then**
      Synchronize $\pi_{\text{vllm}}$ with $\pi$ and clear $\mathcal{D}$; ▷ # Update inference engine
    **end if**
  **end for**

abbreviated OAPL. OAPL begins by synchronizing $\pi$ and $\pi_{\text{vllm}}$ to share the same weights. The inference engine, using $\pi_{\text{vllm}}$, then begins asynchronously generating data, and adding it to the buffer $\mathcal{D}$. Concurrently, the trainer begins updating the policy $\pi$ by minimizing Eq. 2, using data sampled from $\mathcal{D}$. Every $L$ iterations of the trainer, with $L$ being a hyperparameter, the algorithm synchronizes $\pi$ and $\pi_{\text{vllm}}$'s weights. Between synchronizations, the algorithm operates off-policy: $\pi_{\text{vllm}}$ both generates the data and serves as the KL reference in Eq. 2. Due to its fully off-policy nature, OAPL can run completely asynchronously between the two synchronization steps of $\pi$ and $\pi_{\text{vllm}}$.

We clear the buffer $\mathcal{D}$ whenever we synchronize $\pi_{\text{vllm}}$ with $\pi$ to make sure that $\mathcal{D}$ only contains data from a single $\pi_{\text{vllm}}$. This is to ensure that the estimator $\hat{V}^\star$ and hence the advantage is always computed with data from only one sampling distribution $\pi_{\text{vllm}}$. Because OAPL does not rely on importance ratios or clipping operations, the resulting update reduces to a simple least-squares regression loss that remains stable even under substantial policy lag.

**Comparison to GRPO** Following PPO's original design, GRPO uses a clipping operator on $\frac{\pi(y|x)}{\pi_{\text{old}}(y|x)}$ to prevent $\pi$ from deviating too far from $\pi_{\text{old}}$ – the trainer from the previous iteration. This is motivated by conservative policy iteration (Kakade & Langford, 2002). However, clipping is not always effective at preventing $\pi$ from deviating from $\pi_{\text{old}}$. When beginning with $\pi = \pi_{\text{old}}$, the computation of the first gradient update using the GRPO loss does not cause any clipping. Thus if the first gradient is large, one step of gradient descent could already take $\pi$ far away from $\pi_{\text{old}}$, and the clipping operator cannot pull $\pi$ back to $\pi_{\text{old}}$. This is a known problem of PPO/GRPO's loss functions (Hsu et al., 2020). In contrast, OAPL incorporates KL regularization to $\pi_{\text{vllm}}$ into the optimization objective, completely abandons the concept of $\pi_{\text{old}}$, and directly uses the log-probabilities from the sampling distribution $\pi_{\text{vllm}}$. Thus, in each iteration, OAPL directly encourages the trainer $\pi$ to stay close to $\pi_{\text{vllm}}$ while optimizing the reward.

**Comparison to $A^\star$PO** $A^\star$PO was originally designed as an on-policy RL algorithm. It defines $V^\star$ relative to a fixed reference policy $\pi_{\text{ref}}$ and uses $\ln \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}$ inside the regression loss, while training on fresh rollouts from the current policy $\pi$. The reference policy $\pi_{\text{ref}}$ is therefore distinct from the policy used to generate the training data and remains fixed throughout training. In contrast, OAPL uses the lagged inference policy $\pi_{\text{vllm}}$ both to generate the rollouts and as the reference policy inside the regression objective. Because the same policy defines both the sampling distribution and the corresponding $V^\star$, samples generated by an older $\pi_{\text{vllm}}$ can be used directly without importance weighting. OAPL periodically synchronizes $\pi_{\text{vllm}}$ with the trainer, yielding an iterative off-policy procedure in which the behavior policy and its associated KL-regularized objective are updated together.

## 4 Related Work

**Off-policy RL Post-Training** Approaches for dealing with off-policy sampling in RL post-training can broadly be divided into those that avoid importance sampling, and those that apply importance sampling or a related variation of it.

Examples of methods that avoid importance sampling include Melo et al. (2025), who estimate Fisher information for token masking, or Arnal et al. (2025), who bias their objective function for performance improvement guarantees. OAPL similarly avoids the added variance of importance sampling, but does not require additional estimation procedures while remaining unbiased. The most closely related works to ours in this category use squared regression losses for either on- or off-policy training, e.g. REBEL (Gao et al., 2024a), REFUEL (Gao et al., 2024b), AGRO (Tang et al., 2025), Kimi K2 (Team et al., 2025a), or TBA (Bartoldson et al., 2025). However, these approaches do not estimate $V^*$ as OAPL does, replacing it instead with group-relative baselines for variance reduction, similar to the RLOO estimator (Kool et al., 2019). This replacement introduces some bias, as the minimum is no longer the true KL-regularized optimal policy (Xu et al., 2026).

Approaches that rely on importance sampling, or just the importance ratio $\frac{\pi(y|x)}{\pi_{\text{vllm}}(y|x)}$, vary in exactly how they apply it. For instance, DeepSeek-v3.2 (Liu et al., 2025a) deletes rollouts whose likelihood is small under $\pi$, and Zhao et al. (2025) and Zheng et al. (2025b) delete tokens whose token-level ratio is too large or too small. Roux et al. (2025) and Su et al. (2026) construct objective functions to bound the gradients of tokens with large importance ratios. By avoiding importance sampling, OAPL avoids having to delete samples or tokens that could be useful for learning, and does not incur bias or additional tuning costs by adding clipping to ratios or gradients.

**Off-Policy RL in Asynchronous Settings** Work on asynchronous and large-scale RL training has also dealt with off-policy sampling. Recent work on scaling up RL from human feedback systems (Noukhovitch et al., 2025; Khatri et al., 2025), for instance, has used truncated importance sampling for this issue. Outside of the language modeling context, methods for scaling up policy gradient algorithms and leveraging off-policy data have also used some form of (usually truncated) importance sampling (Espeholt et al., 2018; Munos et al., 2016; Wang et al., 2017; Mahmood & Sutton, 2015), or have constrained their data generation to avoid collecting data that is too off-policy (OpenAI et al., 2019). Other approaches avoid importance sampling entirely by learning a Q-function (Haarnoja et al., 2018; Mnih et al., 2015; 2016). OAPL similarly requires no importance sampling, and can actually be understood as a value learning approach which uses $\ln \frac{\pi}{\pi_{\text{vllm}}}$ as a function approximator to estimate the optimal advantage $A^\star$ directly.

## 5 Experimental Setup

We evaluate OAPL on mathematical problem solving and code generation, focusing on stability during asynchronous training and performance measured by Pass@k. For both the math and code generation settings, as in Brantley et al. (2025), we use two separate betas-$\beta_1, \beta_2$, in Equations 1 and 2, respectively, rather than a single $\beta$. This allows for additional freedom in choosing hyperparameters. Additional details about the training setup and hyperparameters for the experiments can be found in Appendix A.

### 5.1 Math Experimental Setup

For our math experiments, we use the random subset of Deepscaler (Luo et al., 2025b) from Wang et al. (2025) as our training dataset and MATH500 (Lightman et al., 2023), Minerva Math (Lewkowycz et al., 2022), OlympiadBench (He et al., 2024), AMC2023 (Mathematical Association of America, 2023), AIME2025 (Mathematical Association of America, 2025), and BRUMO2025 (Brown University Math Olympiad Committee, 2025) as our evaluation sets. We compare OAPL to GRPO with additional truncated importance sampling that accounts for the log-probability difference between the inference engine and the trainer (Yao et al., 2025). For both approaches, we implement asynchronous optimization, which means that the inference engine can generate data while we optimize the trainer. We run GRPO+TIS and OAPL with $L = 1$ and $L = 50$, meaning our data are either on-policy or generated by a policy 50 steps older than the current trainer. We use Qwen2.5-Math-1.5B as our base model for both algorithms.

### 5.2 Code Generation Experimental Setup

For the code generation experiments, we use a highly off-policy two-stage training process to replicate the performance of DeepCoder (Luo et al., 2025a), a publicly available coding model trained via

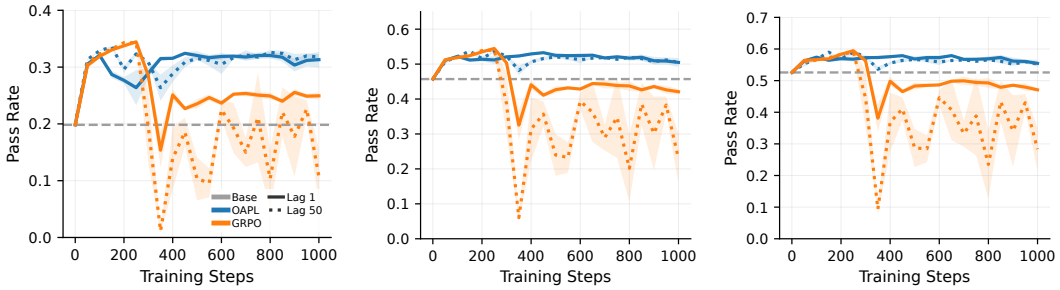



Figure 2: Training curves on math benchmarks. Curves show mean accuracy across six benchmarks (MATH500, Minerva Math, OlympiadBench, AMC2023, AIME2025, BRUMO2025) and shaded regions denote standard error. The gray dashed line is base model performance. (**Left**) Pass@1, (**Middle**) Pass@8, and (**Right**) Pass@16. OAPL converges to high accuracy and remains stable, while GRPO collapses partway through training.

GRPO with several additional heuristics. Beginning with the base model, DeepSeek-R1-Distill-Qwen-14B, we generate an offline dataset of 8 responses for every prompt in DeepCoder's training dataset. To focus training on feasible problems, we additionally filter out all prompts where the model generated no correct responses. We then train the base model on this dataset with OAPL for 1 epoch without synchronizing the trainer and inference engines. Using the resulting model, we generate a new offline dataset from a random subset of 4000 prompts (due to resource constraints), and continue training for an additional four epochs on this dataset. This is equivalent to running OAPL with $L$ set to 1 epoch (approximately 400 gradient updates) and total iterations $T = 2$. The maximum generation length for both rounds of training is 32K tokens. For evaluation, we follow DeepCoder's LiveCodeBench (Jain et al., 2024) setup, using the same subset of 279 LiveCodeBench problems, and evaluating with a maximum generation length of 64K. We evaluate all four checkpoints from each epoch of the second round of training for OAPL, and report results for the best performing checkpoint.

## 6 Experimental Results

We evaluate OAPL along three axes: final accuracy on standard reasoning benchmarks, training dynamics and stability under asynchronous rollouts, and test-time scaling as measured by Pass@k. We first study competition math, where we can track learning curves and dynamics over training, and then turn to code generation, where we evaluate robustness under *extreme policy lag* and compare against the GRPO-trained DeepCoder model.

### 6.1 Results on Competition Math

**Performance on benchmarks** Figure 1 demonstrates that OAPL outperforms the GRPO baseline on all six benchmarks across Pass@k for various $k$. Figure 2 additionally shows the performance across training, averaged over all six benchmarks. Overall, we see that OAPL learns more stably than and outperforms GRPO, regardless of policy staleness. OAPL also handles the high lag case ($L = 50$) with no performance degradation, while GRPO's performance and stability worsen significantly. We also observe that, for OAPL, training on the Pass@1 reward (i.e., the outcome reward alone) improves the Pass@k for $k > 1$ up to $k = 16$, but not for much larger values of $k$ (see Figure 9). Our code generation experiments do not additionally show this phenomenon, but that may be due to the smaller value of $k$ used in that evaluation.

**Training Dynamics** Figure 3 (left) shows the estimated KL divergence and effective sample size (ESS) for GRPO+TIS and OAPL (with a policy lag of 50) across training. We see that GRPO+TIS begins to deviate significantly from the behavior policy as training progresses, while OAPL maintains comparatively small KL values and a high ESS. GRPO+TIS's large KL values help explain its training instability, as the data from the behavior policy is off-policy relative to the trainer. Details of the KL divergence and ESS computation are in Appendix A.2

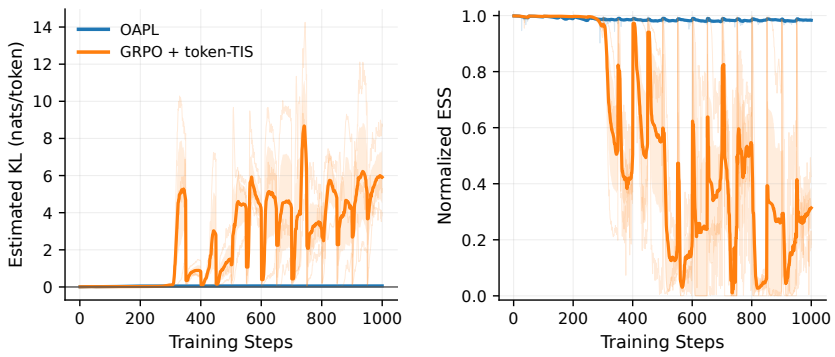



Figure 3: Training dynamics in math reasoning with $L = 50$. (**Left**) $KL(\pi_{vllm}||\pi)$ for both OAPL and GRPO. (**Right**) Effective sample size (ESS) over training for OAPL. Values shown are averaged across all six math benchmarks and three seeds. The shaded regions show standard errors. Details of the metric computation are in Section A.2.

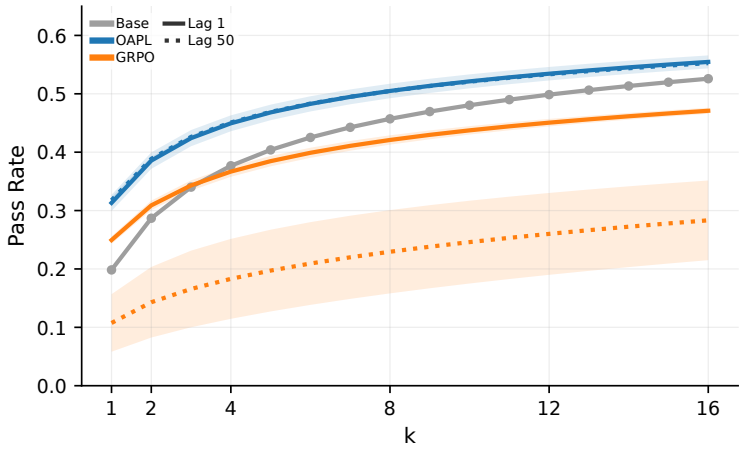



Figure 4: Pass@k scaling behaviors of OAPL and GRPO+TIS on math reasoning. Pass@k values are averaged across all math benchmarks. Individual Pass@k plots are available in Figure 6. We observe that RL training increases Pass@k for $k$ ranging from 1 to 16. OAPL improves scaling relative to GRPO+TIS and the base model.

**Scaling $k$ in Pass@k** Does OAPL's improved stability lead to better scaling behavior under Pass@k? We evaluate the final checkpoint of each method using Pass@k for $k$ ranging from 1 to 16 in Figure 4. Averaged across benchmarks, OAPL achieves the highest Pass@k throughout this range, increasing from approximately 31% at $k = 1$ to 55% at $k = 16$. Moreover, the curves for lags 1 and 50 nearly overlap, indicating that the gains across sampling budgets are largely preserved even when training uses more stale data. Whereas GRPO+TIS exhibits a different pattern. At lag 1, it improves Pass1 over the base model, but the base model out performs it around $k = 4$ and reaches approximately 52% at $k = 16$, compared with 47% for GRPO+TIS. Increasing the lag to 50 further degrades GRPO+TIS across the entire range, with Pass16 falling to approximately 28%. This demonstrates that OAPL scales better than GRPO+TIS on average and on every benchmark individually (see Figure 6).

### 6.2 Results on Code Generation

We evaluate whether OAPL remains effective under extreme off-policyness using the two-stage offline rollout procedure described in Section 5.2, and compare against DeepCoder (Luo et al., 2025a) on LiveCodeBench.

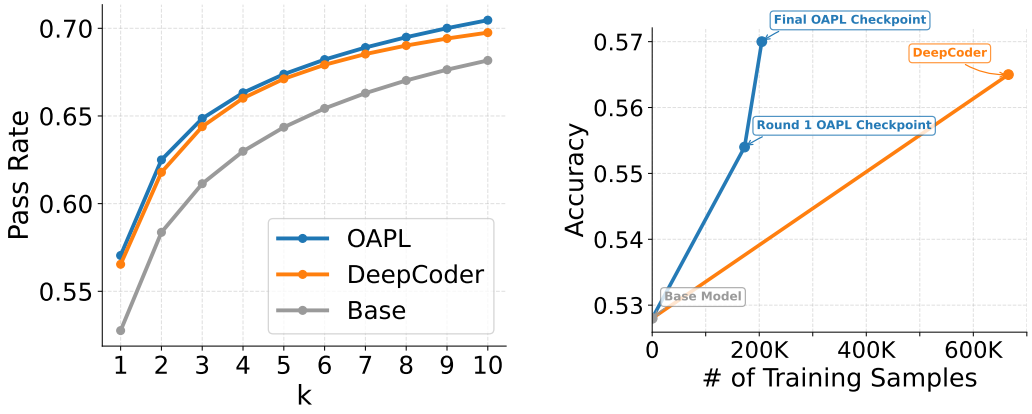



Figure 5: Code generation results on LiveCodeBench. (**Left**) Pass@k scaling for OAPL, DeepCoder, and the shared base model. (**Right**) Sample efficiency: Pass@1 Accuracy versus the number of training generations, highlighting that OAPL matches DeepCoder while using substantially fewer samples. All metrics are computed from 20 rollouts per prompt using the same evaluation protocol as DeepCoder.

**Pass@k performance.** Figure 5 (Left) shows the Pass@k performance on LiveCodeBench for DeepCoder, our OAPL-trained replication model, and the base model used for both (Deepseek-R1-Distill-Qwen-14B) [4]. Pass@k increases with $k$ for all models. Across the entire range of $k$, the OAPL-trained model matches or slightly outperforms DeepCoder. Comparing to the scaling curve from the base model, we see that RL training (both OAPL and the GRPO variant used for DeepCoder) improves pass@k for large $k$.

**Sample efficiency.** Training with OAPL is also significantly more sample efficient than the original DeepCoder training pipeline. Figure 5 (Right) shows OAPL and DeepCoder's Pass@1 performance as a function of total training samples. DeepCoder used approximately 650K samples during training[5]. In contrast, training with OAPL required only $\sim$200K samples. This represents an approximately 3x reduction in the number of samples required, while achieving equal or better performance. This comparison does slightly inflate the actual total computational cost of DeepCoder, as the first part of their training (160 steps) is limited to 16K length generations, and switches to 32K later. But even if we count the 16K generations as 'half' a sample for a fairer accounting, the total is approximately 580K samples, and OAPL still provides significant sample efficiency gains.

## 7 Conclusion and Future Work

Our work demonstrates that a simple off-policy RL method can be more effective than GRPO, an on-policy RL method for LLM post-training. Off-policy methods enable fully asynchronous training and allow algorithms to reuse previously sampled data, often yielding superior computational and sample efficiency. Our experimental results show that we can achieve equivalent or better performance to GRPO on competition math and code generation tasks, in addition to significant increases in sample efficiency from training off-policy. We are excited to continue exploring off-policy training, including training value functions in an off-policy manner for better credit assignment, and leveraging additional offline data (e.g., human data) for more efficient RL.

[4]Note that our DeepCoder accuracy is lower than the originally reported 60.6% accuracy. We were unable to replicate the 60.6% result despite our best efforts to match their sampling parameters and software environment. See here for an open issue where others have had difficulty reproducing the result as well. We also use 20 samples per prompt when computing all Pass@k to reduce the potential randomness in evaluation.

[5]Based on the training runs released here, DeepCoder was trained for 650 steps, with 1024 samples generated per step.

## 8 AI USE STATEMENT

In this work we used generative AI tools for some code generation for the experiments. All generated code was manually reviewed for correctness by the lead author.

## REFERENCES


Charles Arnal, Gaëtan Narozniak, Vivien Cabannes, Yunhao Tang, et al. Asymmetric reinforce for off-policy reinforcement learning: Balancing positive and negative rewards, 2025. URL https://arxiv.org/abs/2506.20520.

Brian Bartoldson, Siddarth Venkatraman, James Diffenderfer, Moksh Jain, Tal Ben-Nun, Seanie Lee, Minsu Kim, Johan Obando-Ceron, Yoshua Bengio, and Bhavya Kailkhura. Trajectory balance with asynchrony: Decoupling exploration and learning for fast, scalable llm post-training, 2025. URL https://arxiv.org/abs/2503.18929.

Kianté Brantley, Mingyu Chen, Zhaolin Gao, Jason D Lee, et al. Accelerating rl for llm reasoning with optimal advantage regression. *arXiv preprint arXiv:2505.20686*, 2025.

Brown University Math Olympiad Committee. *Proceedings of the Inaugural Brown University Math Olympiad (BrUMO 2025)*. Brown University Department of Applied Mathematics, Providence, RI, 2025. Official Tournament Packet.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018. URL https://arxiv.org/abs/1802.01561.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025.

Zhaolin Gao, Jonathan Chang, Wenhao Zhan, Owen Oertell, Gokul Swamy, Kianté Brantley, Thorsten Joachims, Drew Bagnell, Jason D Lee, and Wen Sun. Rebel: Reinforcement learning via regressing relative rewards. *Advances in Neural Information Processing Systems*, 37:52354–52400, 2024a.

Zhaolin Gao, Wenhao Zhan, Jonathan D Chang, Gokul Swamy, Kianté Brantley, Jason D Lee, and Wen Sun. Regressing the relative future: Efficient policy optimization for multi-turn rlhf. *arXiv preprint arXiv:2410.04612*, 2024b.

Matthieu Geist and Bruno Scherrer. Off-policy learning with eligibility traces: A survey, 2013. URL https://arxiv.org/abs/1304.3999.

Daya Guo, Dejian Yang, Haowei Zhang, and Junxiao Song. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, Sep 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Assaf Hallak, Aviv Tamar, Remi Munos, and Shie Mannor. Generalized emphatic temporal difference learning: Bias-variance analysis, 2015. URL https://arxiv.org/abs/1509.05172.

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

Chloe Ching-Yun Hsu, Celestine Mendler-Dünner, and Moritz Hardt. Revisiting design choices in proximal policy optimization. *arXiv preprint arXiv:2009.10897*, 2020.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, et al. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the nineteenth international conference on machine learning*, pp. 267–274, 2002.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Sai Surya Duvvuri, Manzil Zaheer, Inderjit S. Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for llms, 2025. URL https://arxiv.org/abs/2510.13786.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! In *DeepRLStructPred@ICLR*, 2019. URL https://api.semanticscholar.org/CorpusID:198489118.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, et al. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

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

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025a.

Jiacai Liu, Yingru Li, Yuqian Fu, Jiawei Wang, et al. When speed kills stability: Demystifying RL collapse from the training-inference mismatch, September 2025b. URL https://richardli.xyz/rl-collapse.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, et al. Deepcoder: A fully open-source 14b coder at o3-mini level. https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51, 2025a. Notion Blog.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, et al. 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-19681902c1468005bed8ca303013a4e2, 2025b. Notion Blog.

A. Rupam Mahmood and Richard S. Sutton. Off-policy learning based on weighted importance sampling with linear computational complexity. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI'15, pp. 552–561, Arlington, Virginia, USA, 2015. AUAI Press. ISBN 9780996643108.

A. Rupam Mahmood, Huizhen Yu, Martha White, and Richard S. Sutton. Emphatic temporal-difference learning, 2015. URL https://arxiv.org/abs/1507.01569.

Mathematical Association of America. *American Mathematics Competitions (AMC 10/12)*. Mathematical Association of America, Washington, DC, 2023. Official Test Booklet.

Mathematical Association of America. *American Invitational Mathematics Examination (AIME)*. Mathematical Association of America, Washington, DC, 2025. Official Competition Booklet.

Luckeciano C. Melo, Alessandro Abate, and Yarin Gal. Stabilizing policy gradients for sample-efficient reinforcement learning in llm reasoning, 2025. URL https://arxiv.org/abs/2510.00819.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, et al. Asynchronous methods for deep reinforcement learning, 2016. URL https://arxiv.org/abs/1602.01783.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning, 2016. URL https://arxiv.org/abs/1606.02647.

Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, et al. Asynchronous rlhf: Faster and more efficient off-policy rl for language models, 2025. URL https://arxiv.org/abs/2410.18252.

OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, et al. Dota 2 with large scale deep reinforcement learning, 2019. URL https://arxiv.org/abs/1912.06680.

Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, et al. Defeating the training-inference mismatch via fp16, 2025. URL https://arxiv.org/abs/2510.26788.

Pierre Harvey Richemond, Yunhao Tang, Daniel Guo, Daniele Calandriello, et al. Offline regularised reinforcement learning for large language models alignment, 2024. URL https://arxiv.org/abs/2405.19107.

Nicolas Le Roux, Marc G. Bellemare, Jonathan Lebensold, Arnaud Bergeron, et al. Tapered off-policy reinforce: Stable and efficient reinforcement learning for llms, 2025. URL https://arxiv.org/abs/2503.14286.

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. *arXiv preprint arXiv: 2409.19256*, 2024.

Zhenpeng Su, Leiyu Pan, Xue Bai, Dening Liu, et al. Klear-reasoner: Advancing reasoning capability via gradient-preserving clipping policy optimization, 2026. URL https://arxiv.org/abs/2508.07629.

Yunhao Tang, Taco Cohen, David W. Zhang, Michal Valko, and Rémi Munos. Rl-finetuning llms from on- and off-policy data with a single algorithm, 2025. URL https://arxiv.org/abs/2503.19612.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, et al. Kimi k2: Open agentic intelligence, 2025a. URL https://arxiv.org/abs/2507.20534.

Ling Team, Bin Han, Caizhi Tang, Chen Liang, Donghao Zhang, et al. Every attention matters: An efficient hybrid architecture for long-context reasoning, 2025b. URL https://arxiv.org/abs/2510.19338.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Reinforcement learning for reasoning in large language models with one training example, 2025. URL https://arxiv.org/abs/2504.20571.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay, 2017. URL https://arxiv.org/abs/1611.01224.

Bram Wasti, Wentao Ye, Teja Rao, Michael Goin, et al. No more train-inference mismatch: Bitwise consistent on-policy reinforcement learning with vllm and torchtitan, 2025. URL https://blog.vllm.ai/2025/11/10/bitwise-consistent-train-inference.html.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, et al. Huggingface's transformers: State-of-the-art natural language processing, 2020. URL https://arxiv.org/abs/1910.03771.

Zhenghao Xu, Qin Lu, Changlong Yu, and Tuo Zhao. Approximation of log-partition function in policy mirror descent induces implicit regularization for llm post-training, 2026. URL https://arxiv.org/abs/2602.05933.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Your efficient rl framework secretly brings you off-policy rl training, August 2025. URL https://fengyao.notion.site/off-policy-rl.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, et al. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model?, 2025. URL https://arxiv.org/abs/2504.13837.

Xin Zhao, Yongkang Liu, Kuan Xu, Jia Guo, et al. Small leak can sink a great ship–boost rl training on moe with icepop!, Sep 2025. URL https://ringtech.notion.site/icepop.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025a. URL https://arxiv.org/abs/2507.18071.

Haizhong Zheng, Jiawei Zhao, and Beidi Chen. Prosperity before collapse: How far can off-policy rl reach with stale data on llms?, 2025b. URL https://arxiv.org/abs/2510.01161.

Jin Peng Zhou, Kaiwen Wang, Jonathan Chang, Zhaolin Gao, et al. Q♯: Provably optimal distributional rl for llm post-training. *arXiv preprint arXiv:2502.20548*, 2025.

Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, et al. Starling-7b: Improving helpfulness and harmlessness with rlaif. In *First Conference on Language Modeling*, 2024.

# A EXPERIMENTAL DETAILS

We include the detailed hyperparameters of the algorithms here. We use Verl (Sheng et al., 2024) for our experiments. Following the default recipe there, we include a small KL penalty to the reference base for GRPO as a stabilizer, and a default entropy bonus for both GRPO and OAPL. We use a sampling temperature of 0.7, and top p value of 0.8, following the recommended parameters for Qwen2.5-Math-1.5B-Instruct (the closest to the base model we use) in the Qwen2.5-Math repository.

## A.1 GRPO WITH TRUNCATED IMPORTANCE SAMPLING

Following recent work (Yao et al., 2025) we run GRPO with truncated importance sampling (GRPO+TIS), where the importance ratio is truncated at some maximum value $c$ as a variance reduction mechanism. Let $r_{vllm}(x, y_{<i}, y_i) = \frac{\pi_{old}(y_i|x,y_{<i})}{\pi_{vllm}(y_i,x,y_{<i})}$ be the importance ratio correcting for the off-policyness of the data. The GRPO+TIS loss can then be written as follows:

$$\mathbb{E}_{\{y_i\}_{i=1}^{G}\sim\pi_{\text{vllm}}(\cdot|x)}\Bigg[\frac{1}{G}\sum_{y\in\{y_1,\ldots,y_G\}}\frac{1}{|y|}\sum_{t=1}^{|y|} \min\big(r_{\text{vllm}}(x,y_{<t},y_t),c\big) \cdot \min\{r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)A_t\}\Bigg].$$

This is identical to the GRPO loss given in Eq. 2, with the addition of the truncating operation $min(r_{vllm}, c)$. We set $c = 2.0$ in all experiments.

## A.2 KL DIVERGENCE AND EFFECTIVE SAMPLE SIZE ESTIMATION

For Figure 3, we estimate the KL divergence $KL(\pi_{vllm}||\pi_{old})$ at each training step as the average difference in log-probabilities per token:

$$\widehat{\text{KL}}(\pi_{\text{vllm}}\|\pi_t) = \frac{1}{B}\sum_{i=1}^{B}\frac{1}{|T_i|}\sum_{t=1}^{T_i} \log\frac{\pi_{\text{vllm}}(y_{i,t} \mid x_i, y_{i,<t})}{\pi_{\text{old}}(y_{i,t} \mid x_i, y_{i,<t})}.$$

where $B$ is the batch size and $T_i$ is the number of tokens in sequence $i$ in a batch. For effective sample size (ESS), we first compute the importance ratio $r = \frac{\pi_{old}}{\pi_{vllm}}$ for all tokens. ESS is then defined as:

$$\text{ESS} = \frac{\left(\sum_{i=1}^{B}\sum_{t=1}^{T_i} r_{i,t}\right)^2}{B\sum_{i=1}^{B} T_i \sum_{t=1}^{T_i} r_{i,t}^2}.$$

where $B$ and $T_i$ is defined as in the KL divergence estimation.

## A.3 MATH TRAINING HYPERPARAMETERS

Table 1 shows the hyperparameters we use for the optimizer for both methods. We did not tune the optimizer for the math task. Table 2 shows the method-specific hyperparameters, and Table 3 shows the shared hyperparameters of both approaches.

## A.4 CODE GENERATION TRAINING HYPERPARAMETERS

Tables 4, 5, and 6 show the optimizer, OAPL-specific, and training hyperparameters, respectively, for our code generation experiments.

| Parameter | Value |
|---|---|
| Optimizer | AdamW |
| Learning rate | $1 \times 10^{-6}$ |
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.999 |
| Weight decay | 0.01 |
| Gradient clipping type | Norm |
| Clipping threshold | 1.0 |
| Entropy Coefficient | 0.001 |

Table 1: Optimizer and loss hyperparameters used for both OAPL and GRPO for the math task. Note that $\beta_1$ and $\beta_2$ here are for AdamW, not OAPL.

| Parameter | OAPL | GRPO |
|---|---|---|
| $\beta_1$ | 1 | – |
| $\beta_2$ | $1 \times 10^{-3}$ | – |
| Clip ratio | – | 0.2 |
| $\pi_{ref}$ KL Coefficient | – | 0.001 |
| $IS$ Truncation $c$ | – | 2.0 |

Table 2: Method-specific hyperparameters for OAPL and GRPO on math task.

| Category | Parameter | Value |
|---|---|---|
| Training | Generations per prompt (G) | 8 |
| | Batches per update | 1 |
| | Global train batch size | 128 |
| Evaluation | Temperature | 0.7 |
| | Top-$p$ | 0.8 |
| | Max tokens | 3072 |
| Generation | Temperature | 0.7 |
| | Top-$p$ | 0.8 |
| | Max tokens | 3072 |

Table 3: Shared training, evaluation, and generation hyperparameters for OAPL and GRPO.

| Parameter | Value |
|---|---|
| Optimizer | AdamW |
| Learning Rate | $1 \times 10^{-6}$ |
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.999 |
| Weight decay | $1 \times 10^{-}2$ |
| Gradient clipping type | Norm |
| Gradient clipping threshold | 1.0 |

Table 4: Optimizer hyperparameters for code generation experiments.

| Parameter | Value |
|---|---|
| $\beta_1$ | 1 |
| $\beta_2$ | $1 \times 10^{-3}$ |
| $L$ | 418 |

Table 5: OAPL hyperparameters for code generation experiments.

| Category | Parameter | Value |
|---|---|---|
| Training | Generations per prompt (G) | 8 |
| | Batches per update | 1 |
| | Global train batch size | 256 |
| Evaluation | Temperature | 0.6 |
| | Top-$p$ | 0.95 |
| | Max tokens | 65536 |
| Generation | Temperature | 1.0 |
| | Top-$p$ | 1.0 |
| | Max tokens | 32000 |

Table 6: Training, evaluation, and generation hyperparameters for code generation experiments.

# B Additional Benchmark Results

## B.1 Individual Pass@k Plots

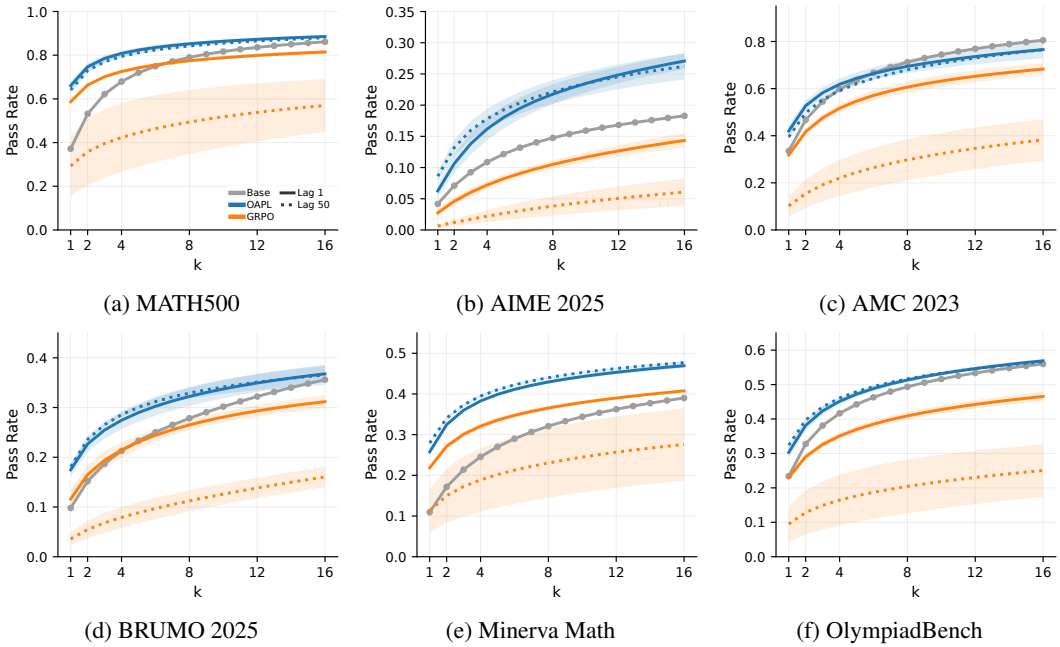
Figure 6 shows the individual Pass@k results for each math reasoning benchmark. We can see that OAPL improves Pass@k over the base model across all 6 benchmarks, while GRPO often underperforms the base model due to training collapse.

## B.2 Maximum Performance Results

Our primary results use the final checkpoint after 1000 steps of training. We also report here similar results taking the maximum performing checkpoint for each method across training. We find that GRPO+TIS often achieves comparable performance under this checkpoint choice, but destabilizes later in training. OAPL achieves stable performance regardless of the number of training steps.

## B.3 Pass@k for Large $k$

We also evaluate Pass@k for larger values of $k$ up to 128 on all math reasoning tasks. Figure 9 shows these results, and we see evidence of RL training not improving performance for large $k$, similar to results reported in Yue et al. (2025). However, we do not see similar behavior in our code experiments (see Figure 5), implying that this may be an artifact of the choice of base model or task.




(a) MATH500 (b) AIME 2025 (c) AMC 2023

(d) BRUMO 2025 (e) Minerva Math (f) OlympiadBench

Figure 6: Scaling behaviors of OAPL and GRPO for Pass@k on individual math benchmarks. We observe RL training increases Pass@k for all $k$ up to 16. OAPL improves scaling relative to GRPO and the base model.

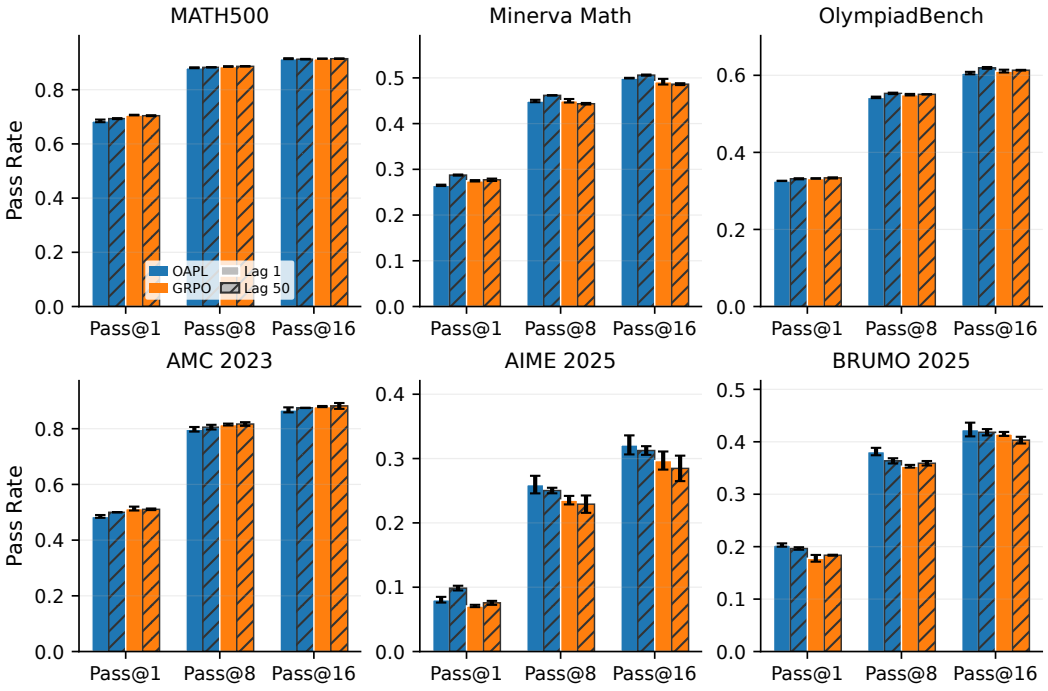
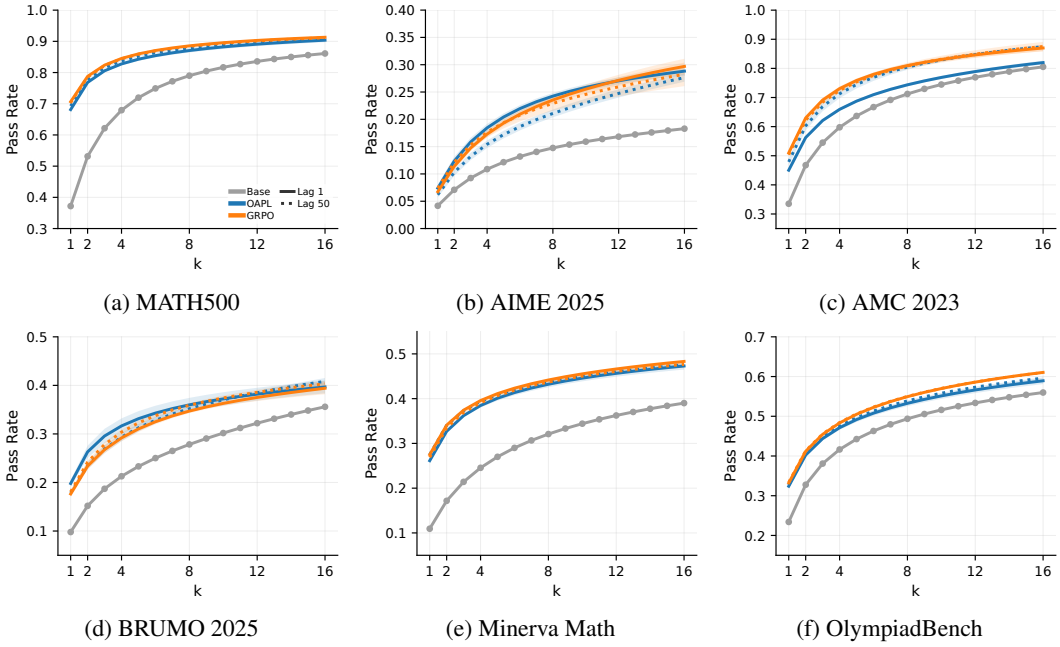


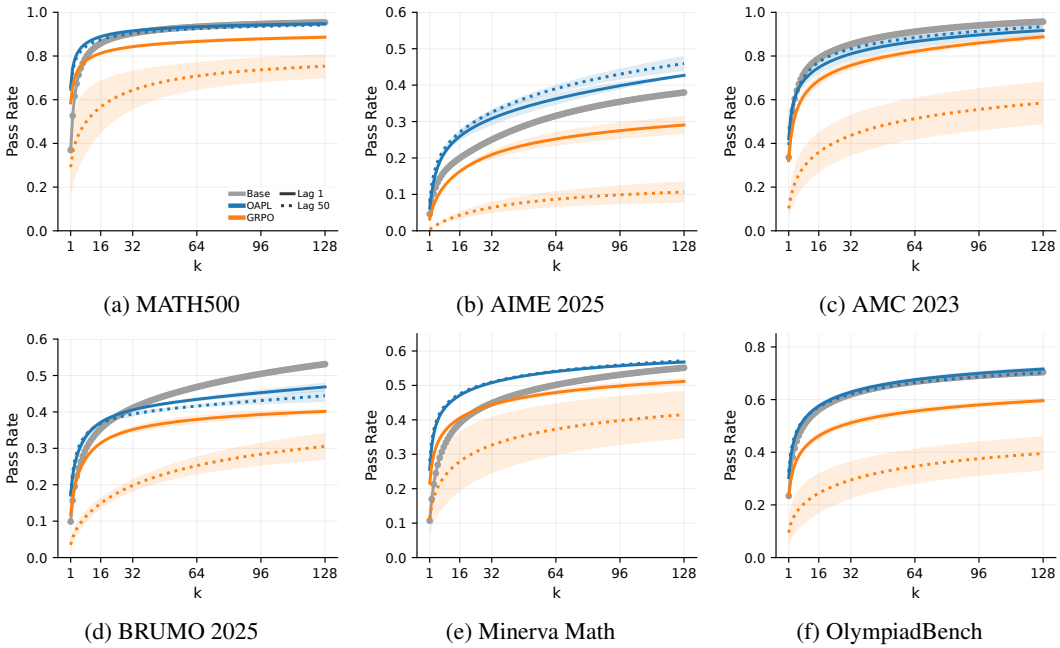

Figure 7: OAPL and GRPO on math reasoning benchmarks. Bars show the maximum performance across 1000 steps of training, averaged over 3 seeds, with error bars indicating standard error. Striped bars show performance when training with a policy lag of 50 ($L = 50$). We report Pass@1, Pass@8, and Pass@16 on MATH500, Minerva Math, OlympiadBench, AMC 2023, AIME 2025, and BRUMO 2025




(a) MATH500 (b) AIME 2025 (c) AMC 2023

(d) BRUMO 2025 (e) Minerva Math (f) OlympiadBench

Figure 8: Scaling behaviors of OAPL and GRPO for Pass@k on individual math benchmarks using max performing checkpoints across training. We observe RL training increases Pass@k for all $k$ up to 16.




(a) MATH500 (b) AIME 2025 (c) AMC 2023

(d) BRUMO 2025 (e) Minerva Math (f) OlympiadBench

Figure 9: Scaling behaviors of OAPL and GRPO for Pass@k on individual math benchmarks after 1000 steps of training for $k$ up to 128.