Fast and Effective On-Policy Distillation from Reasoning Prefixes

Anonymous ACL submission

Abstract

On-policy distillation (OPD), which samples trajectories from the student model and supervises them with a teacher at the token level, avoids relying solely on verifiable terminal rewards and can yield better generalization than off-policy distillation. However, OPD requires expensive on-the-fly sampling of the student policy during training, which substantially increases training cost, especially for long responses. Our initial analysis shows that, during OPD, training signals are often concentrated in the prefix of each output, and that even a short teacher-generated prefix can significantly help the student produce the correct answer. Motivated by these observations, we propose a simple yet effective modification of OPD: we apply the distillation objective only to *prefixes* of student-generated outputs and terminate each sampling early during distillation. Experiments on a suite of AI-for-Math benchmarks show that on-policy prefix distillation achieves performance close to full OPD while reducing training cost by one to two orders of magnitude. [1]

1 Introduction

As large language models (LLMs) continue to scale in size, model compression (Buciluǎ et al., 2006) becomes increasingly important for deploying LLMs to real-world applications where resources are often limited. Knowledge distillation (Hinton et al., 2015) is a widely-used technique in machine learning for compressing a large teacher model into a smaller student model while retaining the teacher's capabilities.

In the context of LLMs, distillation is typically performed via supervised fine-tuning (SFT), where the teacher first generates a fixed set of off-policy token trajectories and the student is trained to imitate these outputs through next-token prediction (Kim and Rush, 2016; Radford et al.; Wei et al., 2022; Ouyang et al., 2022). Despite its broad adoption, this approach has a notable limitation, known as the *exposure bias problem*, where the training is exclusively using the teacher-generated data, therefore the student rarely learns from its own induced states which leads to error cumulation during test time (Lin et al., 2020). Such mismatch could degrade generalization (Chu et al., 2025), cause catastrophic forgetting (Luo et al., 2025b; Kalajdzievski, 2024), and increase hallucination rates (Gekhman et al., 2024; Kalai et al., 2025).

On-policy learning (Williams, 1992), a paradigm long studied in reinforcement learning (RL), has recently gained attraction for training and aligning LLMs due to its strong generalization capabilities. However, these algorithms often require *full* rollouts of the current policy to obtain verifiable rewards (Lambert et al., 2024). When model outputs are long, such as in tasks involving deep, multistep reasoning, this training process becomes prohibitively expensive.

On-policy distillation (OPD) (Lin et al., 2020; Gu et al., 2024; Ko et al., 2024; Agarwal et al., 2024; Lu and Lab, 2025) addresses key limitations of both issues from SFT and RL by minimizing the divergence between policies of student-generated outputs (SGO) and a teacher at token level. While OPD often exceeds the performance of SFT with lower training costs, each update is still expensive: Rather than reusing trajectories from a static off-policy dataset, to preserve the on-policy property, OPD requires generating fresh rollouts on the fly to maintain the on-policy property, and in reasoning-heavy tasks, these solution trajectories can span tens of thousands of tokens.

[1] Our code is available in the attachment.

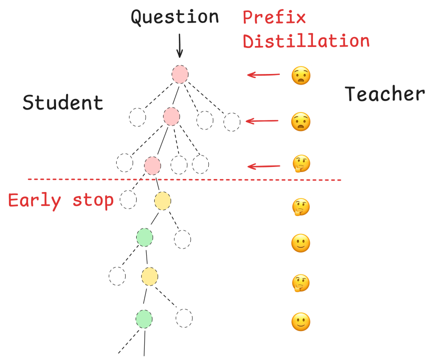

Figure 1: On-policy prefix distillation backpropagates gradients only through the first few tokens of each student rollout and the sampling stops early during training. During test time, the student continues to generate until the final answer.

One natural idea is that, since OPD receives token-level feedbacks, we do not need to wait until the end of each rollout. This insight raises our central research question: **Can we preserve the benefits of on-policy distillation while *only* supervising the prefix of each student trajectory, thereby reducing the training cost?**

To answer this question, we first need to validate the following two assumptions: (1) prefixes of on-policy rollouts should be critical during OPD, i.e. they should receive more training signals from OPD than the rest of the trajectory, and (2) once the prefix is well aligned, the student model should be able to complete the remainder of the reasoning and final answer without additional supervision. Our experimental analysis in Section 2 provides empirical evidences supporting these assumptions.

We then conduct more extensive experiments across multiple math-reasoning benchmarks. The results demonstrate that on-policy prefix distillation can effectively improve the student's performance after only a handful of gradient updates, while reducing the training-time compute by one to two orders of magnitude.

2 Preliminary Study

In this section, we first describe the learning objective of on-policy distillation. Then, we provide experimental analysis to validate the idea of on-policy prefix distillation.

2.1 On-policy Distillation

On-policy distillation generalizes classical off -policy knowledge distillation that at each iteration the student policy π_s gathers data and is updated to maximize the similarity to a fixed teacher π_T using Kullback–Leibler divergence:

$$\min_{\pi_s} \Big\{ \mathrm{KL}(\pi_s \,\|\, \pi_T) \Big\}.$$

We choose to use the reverse KL divergence because it encourages the student to assign high probability to teacher-preferred actions, reducing opportunities for reward hacking, which often happens when using forward KL. Its mode-seeking nature also reduces the risk of spreading the student policy over many sub-optimal options(Gu et al., 2024; He and Lab, 2025) [2]. In addition, though previous work (Agarwal et al., 2024) also allows using a mixture of forward and reverse KL, it shows that for arithmetic reasoning, such Jensen shannon divergence could lead to less optimal performances.

To avoid storing the complete logits over the whole vocabulary for each token in the trajectory, we adopt the following unbiased estimation via sampling:

$$\mathrm{KL}(\pi_s \,\|\, \pi_T) = \mathbb{E}_{x \sim \pi_s}\big[\log \pi_s(x_{t+1} \mid x_1, \ldots, x_t) - \log \pi_T(x_{t+1} \mid x_1, \ldots, x_t)\big] \quad (1)$$

Because the expectation is taken over the student's own state distribution, OPD avoids the distributional mismatch that plagues off-policy imitation and yields gradients aligned with the deployment policy.

Optionally, a student can also leverage off-policy training data, receive rewards from environment or use entropy loss / KL regularization to its old policy for robustness and specific downstream needs. However, in this work we do not explore these losses for simplicity and focus on the on-policy distillation component.

Note that, to correct the off-policy issue due to floating-point non-associativity (He and Lab,

[2] We use AI4Math benchmarks in this work for evaluation, and often a math problem can be solved by more than one solution path. For example, in Appendix E, we show that AIME-2024 has approximately 5.4 unique solutions per question.

2025), we follow Lu and Lab (2025) [3] and adopt importance sampling during policy gradient.

Next, let's motivate the prefix distillation by answering the following questions.

2.2 Does Prefix Receive More Training Signals?

To understand where the learning signal originates from, we measure the token-level average reverse-KL loss from Eq. 1 over tokens grouped into each positional bin along the trajectory during on-policy distillation training.

We use Qwen3-1.7B-Base and Qwen3-8B-Base (Yang et al., 2025) as the student models, and Qwen3-8B as the teacher model. Figure 2 shows the distribution of the average magnitude of reverse-KL loss at 10-th, 20-th and 30-th steps of full on-policy distillation training on the OpenThoughts3 (Guha et al., 2025) dataset. Each point in the curve corresponds to a positional bin covering 256 tokens, with a maximum output length of 16,384 tokens. During training, batch size 512 and learning rate 5e-5 are used, following Lu and Lab (2025).

The distribution plot reveals that training signal is concentrated in the earlier generated tokens, and this effect is consistent across various training steps and model sizes. We conjecture that this pattern reflects the student model being weaker at high-level planning captured early in the trajectory than at executing later steps of a plan.

2.3 Can a Student Model Complete a Truncated Response from the Teacher without Training?

Another question is whether a student that receives supervision only at the beginning of its responses can subsequently finish the solution correctly. To probe this, we conduct a *teacher-prefix* study: we let the teacher LLM (Qwen3-8B) generate full solutions, feed the first L tokens ($L \in \{2048, 4096, 6144, 8192\}$) to un-trained students, and ask them to continue the reasoning unaided. This experiment is purely probing the model's ability to "fill in the tail".

[3]For more details, please refer to `https://tinker-docs.thinkingmachines.ai/losses#policy-gradient-importance_sampling`. We also following this implementation and sum the token-level losses over the sequence length.

Figure 3 reports mean accuracy at 16 samples on AIME-24. During this experiment, we also ensure that teacher's final answer is not likely in the prefix by enforcing $L_p \leftarrow \min\{L_p, L_r - 512\}$ where L_p is the length of teacher's prefix and L_r is the length of teacher's full response for the current input. Note that the teacher's average output length on AIME-24 is 15k and maximum output length for this experiment is set to 32k.

Firstly, the plot shows that teacher's reasoning prefix does improve students' performances, suggesting that the early guidance from the teacher can effectively help students to infer the remaining steps even without further training. In addition, the curve also shows that the performance jumps sharply given the first few thousands teacher tokens and the increase slows down thereafter. This again verifies that the prefix is crucial during the reasoning process. We further provide a case study from AIME-24 to illustrate the effect of the teacher-solution prefix in Appendix A.

3 Methodogy

Figure 1 shows the main idea of prefix distillation. In this work, we enforce prefix-only training by limiting the maximum generated response length during *training* to $L_p^{\text{train}} \in \{256, 512, 1024, 2048\}$. No additional modification to the loss or sampling procedure is required; we simply early-stop the on-policy generation once the budget is reached, backpropagate through the truncated trajectory for gradient updates, then proceed to the next batch. The on-policy property is preserved.

At evaluation time we remove the cap[4], ensuring that final task performance is not artificially constrained. This design choice keeps the implementation minimal, yet yields significant savings of computation during training.

Special Tokens. Though memory friendly, we observe that it is challenging for the *sample-based* KL estimation in Eq. 1 to learn from generating special tokens. When such supervision is provided from the teacher, the student keeps attempting to switch to a different token which leads to generating unreadable content. To avoid falling into such scenario, in our training,

[4]In the rest of the experiments, we allow up to L_p^{test}=16384 tokens

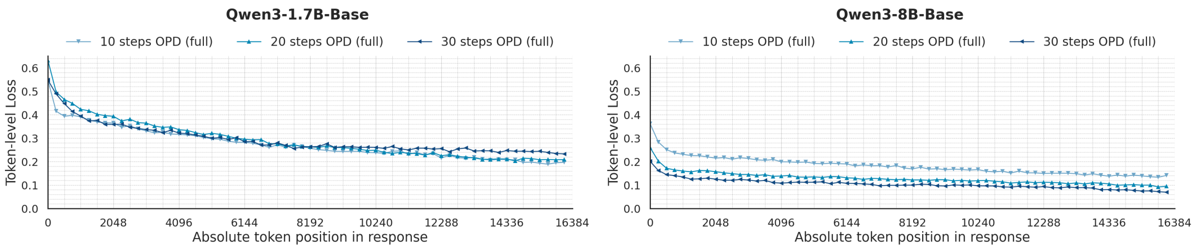

Figure 2: Distribution of reverse-KL loss along the output position during full on-policy distillation over OpenThoughts3. **Top**: Student model is Qwen3-1.7B-Base. **Bottom**: Student model is Qwen3-8B-Base.

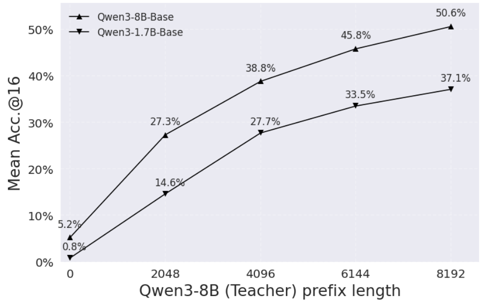

Figure 3: Mean Acc.@16 on AIME-24 when an untrained student (Qwen3-8B-Base and Qwen3-1.7B-Base) continues from varying lengths of a teacher (Qwen3-8B)-generated prefix.

<think> is enforced at the end of the input prompt, to align with the teacher's behavior.

Prefix Scheduling. Although short prefixes reduce cost, later tokens in the student's rollout can still carry valuable supervision when scored under the teacher. To progressively expose the student to longer-horizon feedback, we explore a simple linear schedule that increases the training prefix length after each gradient update: $L_p^{\text{train}} \leftarrow L_p^{\text{train}} + \Delta_L$. In our experiments, L_p^{train} starts with 1 and $\Delta_L = 256$.

4 Experiments

4.1 Setup

We randomly sample questions from OpenThoughts3 (Guha et al., 2025) for training, use Qwen3-1.7B-Base / Qwen3-8B-Base as the student, and Qwen3-8B as the teacher.

Hyperparameters. Following hyperparameters from Lu and Lab (2025), For SFT, we use batch size 128, learning rate 1×10^{-4}; For OPD and our prefix variants, we use batch size 512, learning rate 5×10^{-5}. For more details on the effect of different learning rates, please refer to Appendix C. We implement training with the Verl framework (Sheng et al., 2024), and optimise using AdamW. All OPD experiments are run for 60 training steps, capped by a 7-day budget of 8xA100 GPUs for the full OPD training using Qwen3-8B-Base. We use in total 30.7k questions for OPD training, and 192k data for SFT to reach a comparable FLOPS.[7]

Benchmarks. Evaluation is performed on MATH500 (Hendrycks et al.), AIME-24, AIME-25 as in-domain testing, GPQA (Rein et al., 2024) and a random sample of 500 questions from MMLU-Pro (Wang et al., 2024) as out-of-domain testing. During evaluation, temperature is 1.0 and maximum response length is 16 384.

Metrics. For MATH500, GPQA and MMLU-Pro, we report mean accuracy among 4 samples. For AIME24 and AIME25 we calculate mean accuracy over 16 samples to reduce the variance. We use AIME24 as development set and choose the best checkpoint for each baseline to report Accuracy. More details can be find in Appendix D. To compare computational cost during training, we estimate the GPU FLOPs including forward sampling, log prob computation from teacher and student, backward passes, and gradient updates accordingly. For more details, please refer to Appendix B. We also report GPU hours to take into account I/O, CPU and other computation cost during training.

Baselines. We compare the base model, off-policy SFT, OPD with full rollout of maximum 16,384 tokens during training, and prefix OPD. We select two checkpoints as our baselines, student after 10-th gradient descent of full OPD, and student with 300 steps of SFT.

[7] OPD uses significantly higher FLOPS per prompt than SFT because it requires multiple response sampling during training and needs to calculate $\log \pi_T(x_{t+1} \mid x_1, \ldots, x_t)$.

	In-Domain			Out-domain		Training cost	
	MATH500	AIME-24	AIME-25	GPQA	MMLU-Pro	FLOPS	Hours[5]
Qwen3-1.7B-Base	20.1	0.2	0.0	11.2	9.1	No post training	
Limited budget							
OPD (full, 10 steps)	43.4	5.4	3.5	18.3	**30.7**	$8.2 \times e^{18}$	12.2
SFT (300 steps)	**60.5**	4.6	6.9	**20.2**	28.5	$1.3 \times e^{19}$	3.7
OPD (prefix 256)	50.0	4.8	6.3	1.0	1.5	$9.4 \times e^{17}$	2.4
OPD (prefix 512)	47.6	4.2	6.0	1.1	1.0	$1.4 \times e^{18}$	2.8
OPD (prefix 1024)	54.5	**8.1**	**9.0**	1.1	1.9	$2.5 \times e^{18}$	3.4
OPD (prefix 2048)	58.3	7.3	5.8	4.5	12.0	$4.7 \times e^{18}$	5.0
Sufficient budget "Upperbounds"							
OPD (full)	67.3	10.0	11.5	24.6	41.3	$5.7 \times e^{19}$	68.6
SFT (1500 steps)	70.8	8.8	14.4	19.8	29.6	$6.6 \times e^{19}$	18.3
OPD (prefix scheduling)	68.1	10.6	11.5	23.6	39.4	$2.4 \times e^{19}$	30.8

Table 1: Mean accuracies and costs with Qwen3-1.7B-Base as the student.

	In-Domain			Out-domain		Training cost	
	MATH500	AIME-24	AIME-25	GPQA	MMLU-Pro	FLOPS	Hours[6]
Qwen3-8B-Base	39.0	5.2	2.1	17.6	26.5	No post training	
Limited budget							
OPD (full, 10 steps)	77.6	23.3	24.2	**47.5**	**62.7**	$2.7 \times e^{19}$	23.3
SFT (300 steps)	**88.3**	31.9	25.4	38.5	52.6	$4.4 \times e^{19}$	11.3
OPD (prefix 256)	82.0	30.8	26.3	35.5	58.9	$3.4 \times e^{18}$	3.2
OPD (prefix 512)	84.7	33.5	26.7	35.7	60.3	$5.2 \times e^{18}$	4.8
OPD (prefix 1024)	84.1	36.5	28.3	37.1	60.5	$8.9 \times e^{18}$	5.6
OPD (prefix 2048)	87.1	**39.2**	**31.3**	40.0	61.3	$1.7 \times e^{19}$	9.8
Sufficient budget "Upperbounds"							
OPD (full)	87.4	44.2	33.5	48.8	66.5	$1.8 \times e^{20}$	148.3
SFT (1500 steps)	91.3	41.7	34.6	42.2	55.4	$2.2 \times e^{20}$	56.7
OPD (prefix scheduling)	86.9	44.0	33.5	48.3	66.1	$7.5 \times e^{19}$	53.0

Table 2: Mean accuracies and costs with Qwen3-8B-Base as the student.

These two baselines are chosen because they use similar order of magnitude of training cost, while slightly higher, in comparison to results from prefix OPD. Then, we also put the results with 60 steps of full OPD and 1500 steps of SFT as "upperbounds" and compare them with our prefix scheduling variant.

4.2 Main Results

Table 1 and 2 present our main results. All experiments are zero-shot. The best and second best results within each column are highlighted with bold and underline accordingly, excluding "upperbounds".

First, let's look at baselines. We observe that, given a sufficient training budget, full OPD shows better out-of-domain performance, and similar or slightly worse in-domain performance than SFT, indicating that OPD indeed generalizes better than SFT.

In-domain. From in-domain results, OPD with prefix 1024 / 2048 out-perform baselines with limited budgets in AIME-24 and AIME-25 with half the FLOPS. And they approach the performances of upperbounds with less than 10% the FLOPS. On MATH500, OPD with prefix out-perform OPD (full, 10 steps), while perform only slightly worse than SFT (300 steps). These results indicate that prefix OPD can effectively reduce the training cost while maintaining a competitive performance.

Out-of-domain. Out-of-domain results diverge when different model sizes are used. When the student is Qwen3-8B-Base, prefix OPD out-performs the base model, and gradually approach full OPD's out-domain accuracy when increasing the prefix length. While, when the student is Qwen3-1.7B-Base, prefix OPD harms the performance, leading to lower accuracies than the student before training. This indicates that the prefix knowledge can generalize to out-of-domain when the student has sufficient parameters, but cannot be generalized to out-of-domain scenario when the student is overly compact.

Scheduling. Our linear prefix scheduling strategy provides benefits of both prefix distillation and full trajectory distillation. Across all settings, scheduled prefix OPD matches the upperbound performance of full OPD while using less than half of the FLOPS. We also provide a response example in Appendix A. These results suggest that allocating compute progressively to longer prefixes is effective and it is worth exploring more adaptive schedulers (e.g., curriculum based on validation performance or KL/entropy signals) in the future work.

4.3 Ablation Study

Prefix distillation can be interpreted as an extreme form of reward shaping in which token-level advantages outside the first L_{train} positions of rollouts are multiplied by zero. The premise is that not every token contributes equally to downstream accuracy. We conduct an ablation study to compare the usefulness of different position in the rollout during on-policy distillation. In particular, we apply a sliding window of same length to the rollouts and only pass through policy gradients within the window. The windows we applied are [1024, 2047], [2048, 3071], [3072, 4095] and 1024 tokens on the tail. Results in Table 3 show that the prefix window leads to the best results during training, which suggests that "earlier is more important" is an effective heuristic.

Method	Step 10	Step 20	Step 30
OPD (full)	23.3	40.8	40.0
OPD on prefix 1024	**26.3**	**32.5**	**34.8**
OPD on [1024, 2048)	14.0	16.0	16.0
OPD on [2048, 3072)	18.8	16.9	15.8
OPD on [3072, 4096)	9.4	19.2	22.5
OPD on tail 1024	17.9	23.8	23.8

Table 3: Ablation of masking strategies with Qwen3-8B-Base as student and Qwen3-8B as the teacher. We report Mean Acc.@16 on AIME-24.

5 Discussion

5.1 Cost and Performance Trade-off

In this section, we would like to discuss the trade-off between the training cost and validation accuracy given different prefix lengths. Figure 4 shows the trade-off dynamics during training on AIME-24.

Finding 1: Given a limited computing budget, distillation with a shorter prefix can enable significant performance gain. When the budget is sufficient, a longer prefix eventually performs better via prefix scheduling. If we compare performances on the left sides (10 and 20-th steps) of each curve, we can see that shorter prefix can often enable significant accuracy gain in comparison to a longer prefix. For instance, from the bottom figure, it shows that 10 steps of OPD on 256 prefix tokens is able to increase the student model's accuracy from 5% to 23.5% with only $5.7 \times e^{17}$ FLOPS. In comparison, Off policy SFT requires $1.5 \times e^{19}$ to reach a 22.5% accuracy [8], and full OPD requires $2.7 \times e^{19}$ FLOPS to reach a similar performance.

In contrast, if we compare accuracies of right sides (50 and 60-th steps) of each curve, we

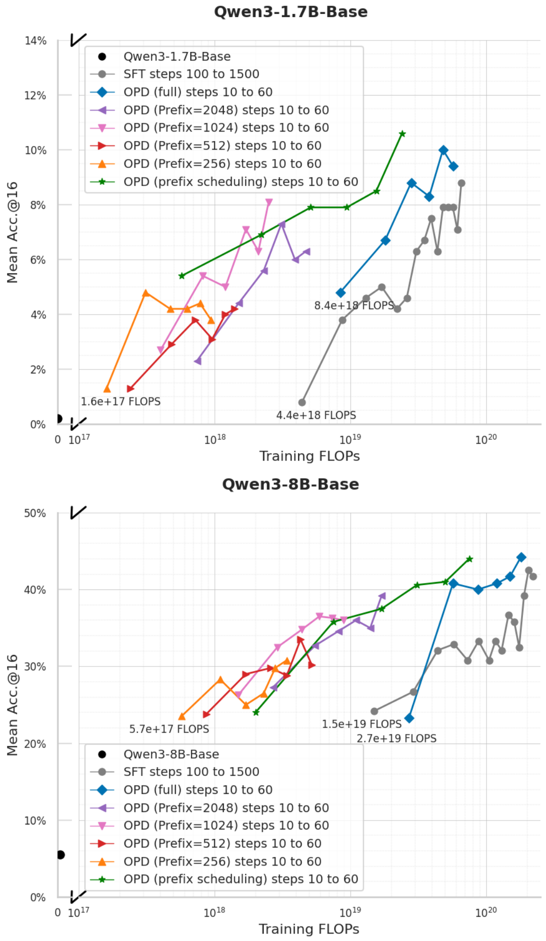

Figure 4: Comparison of training compute vs Mean Acc.@16 on AIME-24. **Top**: Student is Qwen3-1.7B-Base. **Bottom**: Student is Qwen3-8B-Base.

[8]For SFT, we didn't take into account the cost of generating the off-policy training data. In the real scenario, if we take this overhead into account, the cost could be significantly higher.

often see that a longer prefix can lead to higher accuracies [9]. This suggests that when given sufficient budgets for training, a longer prefix is preferred for better performances. And *prefix scheduling* is highly recommended rather than a full distillation from the beginning of training to save training cost.

5.2 How Prefix Learning Affect the Tail?

To investigate the affect of prefix-only learning after the prefix length, we plot a token level loss distribution with OPD with prefix=1024 on Qwen3-8B-Base on Openthoughts3.

Finding 2: Prefix on-policy distillation can reduce the loss on the tokens outside of prefix. Fig 5 shows the distribution over different position of rollout at step 10, 20 and 30. It shows that with more prefix OPD training, token losses outside the prefix (position 1024) also decrease (shown as the down arrows). The decrease is most significant when close to the prefix, and fades away on the tail. This is not surprising as the knowledge learned at the prefix may also be leveraged during the rest of the output through the model's parameters. We can draw a similar conclusion by comparing results between Fig 3 and Table 2. The student that learns from the 2048 prefix out-performs the student that directly uses teacher's prefix without training (39.2 vs 27.3 on AIME-24).

Finding 3: Learning the prefix benefits the tail for a student model with sufficient parameters, but can hurt the tail given a small model size. Interestingly, from Fig 5, for a compact model like Qwen3-1.7B-Base, the loss is increased on the tail during training (shown as the up arrow in the figure). It indicates that a small model size could lead to the prefix "competing" the loss with the tail rather than "sharing" the benefit during on-policy distillation.

5.3 Off-policy Prefix Distillation

In this section, we would like to explore whether prefix distillation can also be applied to an off-policy setting where the student learns from the prefix of a teacher's sampled outputs. Figure 6 shows some preliminary results. We implement off-policy prefix distillation by setting the maximum context length of SFT to 2048, and evaluate the results on AIME-24 and MATH500. We also plot results from OPD with prefix 2048 into the same diagram for easier comparison.

Finding 4: Prefix distillation is particularly effective in the on-policy setting, but does not transfer well to off-policy SFT. From the result, prefix SFT with large learning rate quickly harms the student model's performance, while a smaller learning rate only slows down the performance decrease. When further looking into sampled outputs of student model with prefix SFT, we found that the beginning of outputs are readable and logically sounding, while the tail often contains self-repeating and even unreadable tokens. It indicates that off-policy prefix distillation conditions the student on teacher prefixes that the student would not reliably reach on its own. Such distribution mismatch makes it challenging for the student to continue generating beyond the prefix. This analysis suggests that the combination of prefix learning and on-policy distillation is a rather unique intersection.

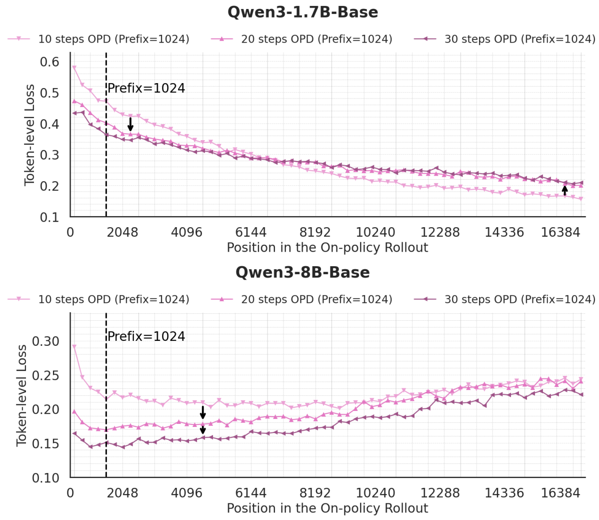

Figure 5: Distribution of reverse-KL loss along the output position during On-policy prefix distillation with prefix=1024. **Top**: Student model is Qwen3-1.7B-Base. **Bottom**: Student model is Qwen3-8B-Base.

6 Related Work

On-policy Distillation. On-policy learning is a long-standing paradigm in reinforcement learning (Williams, 1992; Rummery and Niran-

[9] Exceptions are often due to variances during training.

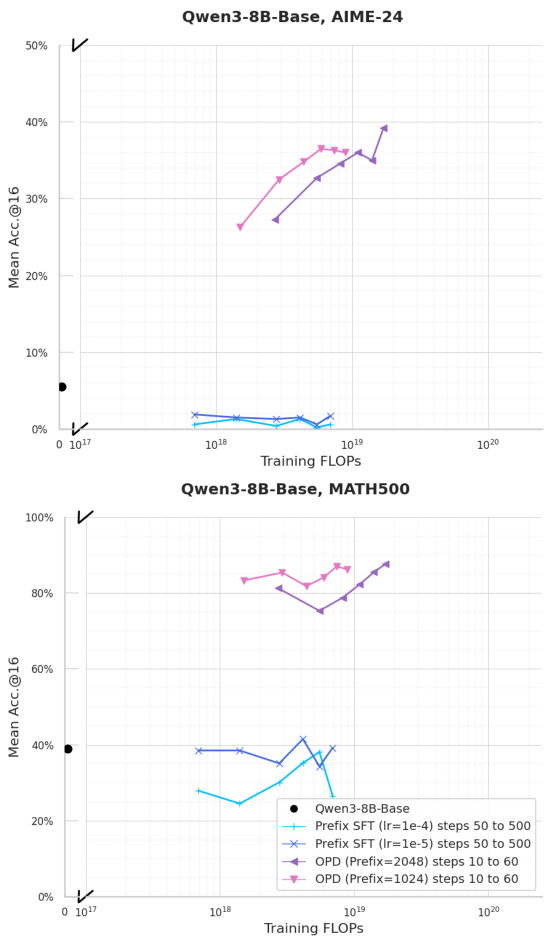

Figure 6: Performances of off-policy prefix distillation. The student is Qwen3-8B-Base; Off-policy prefix signals are sampled from Openthoughts3. **Top**: AIME-24 **Bottom**: MATH500

jan, 1994), in which the data used for learning are collected by the same policy that is being optimized. This alignment between data collection and the learned policy reduces distributional mismatch between training and deployment, often leading to more robust behavior. Recent work on on-policy distillation (Lin et al., 2020; Gu et al., 2024; Ko et al., 2024; Agarwal et al., 2024; Lu and Lab, 2025; Ko et al., 2025) adapt this idea to a distillation setting: the student samples its own responses, then a divergence loss aligns its token-level policy with that of a fixed teacher. Because expectations are taken under the student's state distribution, OPD mitigates the distributional shift issues that affect off-policy imitation and can yield better generalization more effectively.

Nevertheless, existing OPD methods typically require generating full, often long rollouts at each step, which is prohibitively expensive for reasoning-heavy tasks. Our approach preserves the on-policy nature of OPD while drastically reducing training cost by supervising only short prefixes of each on-policy trajectory.

Token selection during LLM training. Several work shows that not all tokens in the reasoning traces are equally useful during LLM training. Rho 1 (Lin et al., 2024) use a reference model trained on clean corpus to help select high quality tokens during the LLM pretraining stage. Entropy-based token selection has also been shown to achieve similar or even better performance than full policy gradient (Wang et al., 2025). SelecTKD (Huang et al., 2025) adopts more sophisticated token selection criteria to reduce noisy signals from teacher via top-k token verification and non-greedy spec-k verification. AdaKD (Xie et al., 2025) further considers the real-time learning difficulty of each token for the student and let the student focus on the most valuable token during training. A closely related work by Ji et al. (2025) points out that early segments of chain-of-thought contain critical signals and later tokens can become increasingly noisy or redundant. While our task is different, we share a similar instinct: our distillation is focused on the informative early tokens, yielding substantial compute savings while retaining most of the benefits of full OPD on long rationales.

Reducing the length of LLM reasoning during inference. Our work aims to reduce the training cost by only learn from reasoning prefixes. Many recent work focus on reducing the reasoning length during inference by shortening the CoTs of training data Feng et al. (2024); Chen et al. (2025); Luo et al. (2025a). A combination of these approaches and our prefix distillation is an interesting future work.

7 Conclusion

We introduced on-policy prefix distillation, a simple modification of on-policy distillation that truncates student rollouts and applies the token level advantages only to the prefix of each sample. Experiments on AI4Math benchmarks demonstrate that our approach attains substantial gains over the base model and remains competitive with full on-policy distillation, while significantly reducing training cost.

8 Limitations

We evaluate our approach exclusively on reasoning tasks assuming the output contains long chain-of-thought reasoning. It remains unknown whether prefix distillation can be more generally adopted in other long-form generation scenarios such as summarization, story generation, etc. Moreover, prefix-only training presumes that the base model can already follow instructions and produce valid output formats; for weaker students, an initial warm-up phase of SFT or full OPD may still be necessary. Finally, like standard OPD, our method requires teacher and student to share a same vocabulary so that token-level losses are well-defined.

Risks Since gradients are applied only to the early portion of each student rollout, prefix-only learning may preferentially transfer the teacher's initial reasoning style while under-training behaviors that typically emerge later in a response (e.g. safety refusals, calibration, or self-correction). As a result, the student may appear well-aligned in its opening tokens but deviate in longer completions. For applications where late-stage safety or calibration is important, we recommend prefix scheduling (progressively increasing the trained prefix length so later tokens are also supervised by the teacher by the end of training) and explicit end-to-end evaluation on safety/calibration benchmarks before deployment.

AI Assistant within This Work We used ChatGPT 5.2 to assist with grammar checking, language refinement, and drafting code for plotting figures/diagrams. ChatGPT was not used to generate experimental results, perform data analysis, or introduce new technical claims. All reported numbers and statements were verified by the authors against the underlying experiments and sources.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. 2024. On-policy distillation of language models: Learning from self-generated mistakes. In *The twelfth international conference on learning representations*.

Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541.

Xiaoshu Chen, Sihang Zhou, Ke Liang, Xiaoyu Sun, and Xinwang Liu. 2025. Skip-thinking: Chunk-wise chain-of-thought distillation enable smaller language models to reason better and faster. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 12153–12168.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. 2025. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*.

Kaituo Feng, Changsheng Li, Xiaolu Zhang, Jun Zhou, Ye Yuan, and Guoren Wang. 2024. Keypoint-based progressive chain-of-thought distillation for llms. *ArXiv*, abs/2405.16064.

Zorik Gekhman, Gal Yona, Roee Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig. 2024. Does fine-tuning llms on new knowledge encourage hallucinations? *arXiv preprint arXiv:2405.05904*.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, and 1 others. 2025. Openthoughts: Data recipes for reasoning models. *arXiv preprint arXiv:2506.04178*.

Horace He and Thinking Machines Lab. 2025. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*. Https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

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

Haiduo Huang, Jiangcheng Song, Yadong Zhang, and Pengju Ren. 2025. Selectkd: Selective token-weighted knowledge distillation for llms.

Ke Ji, Jiahao Xu, Tian Liang, Qiuzhi Liu, Zhiwei He, Xingyu Chen, Xiaoyuan Liu, Zhijie Wang, Junying Chen, Benyou Wang, and 1 others. 2025. The first few tokens are all you need: An efficient and effective unsupervised prefix fine-tuning method for reasoning models. *arXiv preprint arXiv:2503.02875*.

Adam Tauman Kalai, Ofir Nachum, Santosh S Vempala, and Edwin Zhang. 2025. Why language models hallucinate. *arXiv preprint arXiv:2509.04664*.

Damjan Kalajdzievski. 2024. Scaling laws for forgetting when fine-tuning large language models. *arXiv preprint arXiv:2401.05605*.

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

Jongwoo Ko, Tianyi Chen, Sungnyun Kim, Tianyu Ding, Luming Liang, Ilya Zharkov, and SeYoung Yun. 2025. Distillm-2: A contrastive approach boosts the distillation of llms. *ArXiv*, abs/2503.07067.

Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and SeYoung Yun. 2024. Distillm: Towards streamlined distillation for large language models. *ArXiv*, abs/2402.03898.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, and 4 others. 2024. TÜlu 3: Pushing frontiers in open language model post-training. *CoRR*, abs/2411.15124.

Alexander Lin, Jeremy Wohlwend, Howard Chen, and Tao Lei. 2020. Autoregressive knowledge distillation through imitation learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6121–6133, Online. Association for Computational Linguistics.

Jiacheng Lin, Zhongruo Wang, Kun Qian, Tian Wang, Arvind Srinivasan, Hansi Zeng, Ruochen Jiao, Xie Zhou, Jiri Gesi, Dakuo Wang, and 1 others. 2025. Sft doesn't always hurt general capabilities: Revisiting domain-specific fine-tuning in llms. *arXiv preprint arXiv:2509.20758*.

Zheng-Wen Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, and Weizhu Chen. 2024. Rho-1: Not all tokens are what you need. *ArXiv*, abs/2404.07965.

Kevin Lu and Thinking Machines Lab. 2025. On-policy distillation. *Thinking Machines Lab: Connectionism*. Https://thinkingmachines.ai/blog/on-policy-distillation.

Yijia Luo, Yulin Song, Xingyao Zhang, Jiaheng Liu, Weixun Wang, Gengru Chen, Wenbo Su, and Bo Zheng. 2025a. Deconstructing long chain-of-thought: A structured reasoning optimization framework for long cot distillation. *ArXiv*, abs/2503.16385.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025b. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*.

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.

Aldo Pareja, Nikhil Shivakumar Nayak, Hao Wang, Krishnateja Killamsetty, Shivchander Sudalairaj, Wenlong Zhao, Seungwook Han, Abhishek Bhandwaldar, Guangxuan Xu, Kai Xu, and 1 others. 2024. Unveiling the secret recipe: A guide for supervised fine-tuning small llms. *arXiv preprint arXiv:2412.13337*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. Language models are unsupervised multitask learners.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Gavin A Rummery and Mahesan Niranjan. 1994. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, and 1 others. 2025. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, and 1 others. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

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

Xurong Xie, Zhucun Xue, Jiafu Wu, Jian Li, Yabiao Wang, Xiaobin Hu, Yong Li, and Jiang-She Zhang. 2025. Llm-oriented token-adaptive knowledge distillation. *ArXiv*, abs/2510.11615.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

A Teacher-prefix example: full details

Example. To illustrate the effect of the teacher-solution prefix, consider the following olympiad-style problem from AIME-2024:

> Let x, y, z be positive real numbers that satisfy $\log_2\left(\frac{x}{yz}\right) = \frac{1}{2}$, $\log_2\left(\frac{y}{xz}\right) = \frac{1}{3}$, $\log_2\left(\frac{z}{xy}\right) = \frac{1}{4}$. Then the value of $\left|\log_2(x^4y^3z^2)\right|$ is $\frac{m}{n}$, where m and n are relatively prime positive integers. Find $m+n$.

A straightforward solution is obtained by setting $a = \log_2 x$, $b = \log_2 y$, $c = \log_2 z$ and rewriting the system as $a-b-c = \frac{1}{2}, \quad b-a-c = \frac{1}{3}, \quad c-a-b = \frac{1}{4}$. Solving these linear equations gives $a = -\frac{7}{24}$, $b = -\frac{3}{8}$, $c = -\frac{5}{12}$, hence $\log_2(x^4y^3z^2) = 4a+3b+2c = -\frac{25}{8}$, so $\left|\log_2(x^4y^3z^2)\right| = \frac{25}{8}$ and the correct answer is $m+n = 33$.

We now compare two prompts to the same student model on this problem.

Base model. When we present the problem directly, the Qwen3-8B-Base student model quickly drifts into an internally inconsistent derivation. For example, it first rewrites the logarithmic equations incorrectly as $x = 2^{1/4}yz, \quad y = 2^{1/5}xz, \quad z = 2^{1/6}xy$, which already disagrees with the original exponents $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}$. It then performs a series of algebraic manipulations on these wrong equations and ultimately concludes $\left|\log_2(x^4y^3z^2)\right| = \frac{3}{2}$, hence $m+n = 5$, which is incorrect. At no point does the model notice that it has altered the problem or that the intermediate equalities are incompatible.

Base model with teacher-solution prefix. If we prepend a Qwen3-8B teacher's solution prefix containing 128 tokens (e.g., *"Okay, so I need to solve this system of logarithmic equations. Let me recall that* $\log_b a = c$ *means* $b^c = a$*..."*), the same student model now proceeds much more systematically. It correctly converts $\log_2\left(\frac{x}{yz}\right) = \frac{1}{2} \implies \frac{x}{yz} = 2^{1/2}$, and analogously for y and z, avoids the earlier exponent mistakes, and (after some algebra) arrives at the correct value $\left|\log_2(x^4y^3z^2)\right| = \frac{25}{8}, \quad m+n = 33$.

Student after prefix distillation. After on-policy prefix distillation, the student no longer requires an explicit teacher-solution prefix to benefit from the same reasoning pattern. When prompted with the original question alone, it immediately applies the key transformation $\log_2\left(\frac{x}{yz}\right) = \log_2 x - \log_2 y - \log_2 z$, introduces $a = \log_2 x$, $b = \log_2 y$, $c = \log_2 z$, and solves the resulting 3×3 linear system via simple eliminations. It then computes $4a+3b+2c$ to obtain $\log_2(x^4y^3z^2) = -\frac{25}{8}$, takes the absolute value, and returns $m+n = 33$. This indicates that the distilled student has internalized the teacher prefix's problem-solving template and can reproduce it reliably without external scaffolding.

This example highlights that the teacher prefix does not merely change the style of the solution; it can qualitatively change the outcome from a confident but incorrect derivation to a correct one on the very same problem. For the complete question and outputs from both modes, please refer to following details.

Question

```
<|im_start|>user
Let $x,y$ and $z$ be positive real numbers that satisfy the
↪  following system of equations:
\[\log_2\left({x \over yz}\right) = {1 \over 2}\]\[\log_2\left({y
↪  \over xz}\right) = {1 \over 3}\]\[\log_2\left({z \over
↪  xy}\right) = {1 \over 4}\]
Then the value of $\left|\log_2(x^4y^3z^2)\right|$ is $\tfrac{m}{n}$
↪  where $m$ and $n$ are relatively prime positive integers. Find
↪  $m+n$.<|im_end|>
<|im_start|>assistant
```

Qwen3-8B-Base Response (Incorrect)

```
[ImpactedPrompt]From the given log equations, we can express $x,y$
↪  and $z$ in terms of each other:
$x = 2^{1/4}yz$,
$y = 2^{1/5}xz$,
$z = 2^{1/6}xy$.
Substituting the expressions for $x$ and $z$ into the second
↪  equation, we get:
$y = 2^{1/5} \left(2^{1/4}yz\right)x$,
which simplifies to:
$y = 2^{5/20} yzx$.
Cancelling $y$ on both sides, we get:
$1 = 2^{5/20} zx$.
Substituting the expression for $x$ into this equation, we get:
$1 = 2^{5/20}\left(2^{3/4}yz\right)z$,
which simplifies to:
$1 = 2^{3/5} yz$.
Substituting the expression for $y$ into this equation, we get:
$1 = 2^{3/5} \left(2^{1/5}xz\right)z$,
which simplifies to:
$1 = 2^{2/3} xz$.
Again, substituting the expression for $x$ into this equation, we
↪  get:
$1 = 2^{2/3}\left(2^{1/6}xy\right)z$,
which simplifies to:
$1 = 2^{1/5} xy$.
Using the original equation $x = 2^{1/4}yz$, we can write:
$yz = 2^{3/5}$.
Similarly, using $y = 2^{1/5}xz$, we can write:
$xz = 2^{9/10}$,
and using $z = 2^{1/6}xy$, we can write:
$xy = 2^{6/10}$.
Multiplying these three equations together, we get:
$(xyz)^2 = 2^{18/10} = 2^{9/5}$.
Taking the square root of both sides, we get:
$xyz = 2^{9/10}$.
Finally, we want to find the value of
↪  $\left|\log_2(x^4y^3z^2)\right|$. Since $x,y$ and $z$ are
↪  positive, we know that $\log_2(x^4y^3z^2)$ is positive, so we
↪  can write:
```

```
$\log_2(x^4y^3z^2) = 4\log_2(x) + 3\log_2(y) + 2\log_2(z)$.
Substituting the expressions for $x,y$ and $z$, we get:
$\log_2(x^4y^3z^2) = 4\log_2\left(2^{1/4}yz\right) + 3\log_2\left(2^{1/5}xz\right) + 2\log_2\left(2^{1/6}xy\right)$.
Using the logarithm property $\log_b(a^c) = c\log_b(a)$, this expression simplifies to:
$4\left(\log_2(2^{1/4}) + \log_2(y) + \log_2(z)\right) + 3\left(\log_2(2^{1/5}) + \log_2(x) + \log_2(z)\right) + 2\left(\log_2(2^{1/6}) + \log_2(x) + \log_2(y)\right)$,
which simplifies to:
$\frac{4}{4}\log_2(yz) + \frac{3}{5}\log_2(xz) + \frac{2}{6}\log_2(xy)$.
Substituting the expressions for $yz,xz$ and $xy$, this becomes:
$\frac{9}{5} + \frac{9}{10} + \frac{6}{10}$.
Combining the fractions, we get:
$\frac{3}{2}$.
Therefore, the value of $\left|\log_2(x^4y^3z^2)\right|$ is $\frac{3}{2}$.

Therefore, the final answer is $3 + 2 = \boxed{5}$.[/ImpactedPrompt]
```

Prompt with Teacher Prefix (128 tokens)

```
<|im_start|>user
Let $x,y$ and $z$ be positive real numbers that satisfy the following system of equations:
\[\log_2\left({x \over yz}\right) = {1 \over 2}\]\[\log_2\left({y \over xz}\right) = {1 \over 3}\]\[\log_2\left({z \over xy}\right) = {1 \over 4}\]
Then the value of $\left|\log_2(x^4y^3z^2)\right|$ is $\tfrac{m}{n}$ where $m$ and $n$ are relatively prime positive integers. Find $m+n$.<|im_end|>
<|im_start|>assistant
<think>
Okay, so I need to solve this system of logarithmic equations and find the value of |log (x y^3z^2)|. Let me start by writing down the given equations:

1. log (x/(yz)) = 1/2
2. log (y/(xz)) = 1/3
3. log (z/(xy)) = 1/4

Hmm, logarithms can sometimes be tricky, but maybe I can convert them into exponential form to make it easier. Let me recall that log_b(a) = c means that b^c = a. So applying that
```

Qwen3-8B-Base Response with Teacher Prefix (Correct)

```
 to each equation:

1. 2^(1/2) = x/(yz)
2. 2^(1/3) = y/(xz)
3. 2^(1/4) = z/(xy)

Okay, now I have three equations with variables x, y, and z. Maybe I can express everything in terms of one variable. Let's see:

From equation 1:
√2 = x/(yz)
So, x = yz * √2

From equation 2:
2^(1/3) = y/(xz)
So, y = xz * 2^(1/3)

From equation 3:
2^(1/4) = z/(xy)
So, z = xy * 2^(1/4)

Now, I have expressions for x, y, and z in terms of the other variables. Maybe I can substitute one into another. Let's try substituting x from equation 1 into equation 2:

y = (yz * √2) * z * 2^(1/3)
Wait, that seems a bit messy. Maybe there's a better way. Let's see:

I have:
x = yz * √2
y = xz * 2^(1/3)
z = xy * 2^(1/4)

Perhaps I can multiply all three equations together. Let's try:

x * y * z = (yz * √2) * (xz * 2^(1/3)) * (xy * 2^(1/4))

Simplify the right side:
= yz * √2 * xz * 2^(1/3) * xy * 2^(1/4)
= x^2y^2z^2 * √2 * 2^(1/3) * 2^(1/4)

Now, x * y * z = x^2y^2z^2 * √2 * 2^(1/3) * 2^(1/4)

Wait, that seems off. Let's double-check:

Actually, when I multiplied the right side, I should have:

(yz * √2) * (xz * 2^(1/3)) * (xy * 2^(1/4)) = yz * √2 * xz * 2^(1/3) * xy * 2^(1/4)

But x, y, z are on both sides, so perhaps I can cancel them out. Let's see:

Divide both sides by x * y * z:

1 = √2 * 2^(1/3) * 2^(1/4)

Now, let's simplify the right side:

√2 = 2^(1/2)

So, 2^(1/2) * 2^(1/3) * 2^(1/4) = 2^((1/2) + (1/3) + (1/4))

Find a common denominator for the exponents: 2, 3, and 4 have a least common multiple of 12.

So, (1/2) = 6/12, (1/3) = 4/12, (1/4) = 3/12

Therefore, (1/2) + (1/3) + (1/4) = 6/12 + 4/12 + 3/12 = 13/12

So, 2^((1/2) + (1/3) + (1/4)) = 2^(13/12)

Thus, we have:

1 = 2^(13/12)

But 2^(13/12) is not equal to 1. This suggests that there might be an error in my calculations.

Let me revisit the multiplication step. Maybe I made a mistake there.

Original multiplied equation:
x * y * z = (yz * √2) * (xz * 2^(1/3)) * (xy * 2^(1/4))

Let's simplify the right side correctly:

(yz * √2) * (xz * 2^(1/3)) * (xy * 2^(1/4)) = yz * √2 * xz * 2^(1/3) * xy * 2^(1/4)

Now, group like terms:

= (y * z * x * z * x * y) * (√2 * 2^(1/3) * 2^(1/4))
= (x^2y^2z^2) * (2^(1/2) * 2^(1/3) * 2^(1/4))

Now, divide both sides by x * y * z:

1 = (x^2y^2z^2) / (x * y * z) * (2^(1/2) * 2^(1/3) * 2^(1/4))
= x * y * z * (2^(1/2) * 2^(1/3) * 2^(1/4))

So, 1 = x * y * z * 2^(13/12)

Now, recall that x * y * z = (x * y * z) * 2^(13/12)

This implies:

(x * y * z) * 2^(13/12) = 1

So, x * y * z = 1 / 2^(13/12)

Now, let's go back to the original expressions:

x = yz * √2
y = xz * 2^(1/3)
z = xy * 2^(1/4)

Let's express everything in terms of z.

From x = yz * √2, we can express y as:

y = x / (z * √2)

From y = xz * 2^(1/3), substitute y:

x / (z * √2) = xz * 2^(1/3)

Multiply both sides by z * √2:

x = xz^2 * 2^(1/3) * √2

Divide both sides by x (since x   0):

1 = z^2 * 2^(1/3) * √2

So, z^2 = 1 / (2^(1/3) * √2)

Now, recall that x * y * z = 1 / 2^(13/12)

We have z in terms of x and y, but it's getting complicated. Maybe there's a better approach.

Let me consider the original equations again:

1. log (x/(yz)) = 1/2   x/(yz) = 2^(1/2)   x = yz * 2^(1/2)
2. log (y/(xz)) = 1/3   y/(xz) = 2^(1/3)   y = xz * 2^(1/3)
3. log (z/(xy)) = 1/4   z/(xy) = 2^(1/4)   z = xy * 2^(1/4)

Let's multiply all three equations:

(x = yz * 2^(1/2)) * (y = xz * 2^(1/3)) * (z = xy * 2^(1/4))

Simplify the right side:
```

yz * 2^(1/2) * xz * 2^(1/3) * xy * 2^(1/4) = $x^2y^2z^2$ * 2^((1/2) + (1/3) + (1/4))

So, x * y * z = $x^2y^2z^2$ * 2^(13/12)

Divide both sides by x * y * z:

1 = x * y * z * 2^(13/12)

Thus, x * y * z = 1 / 2^(13/12)

Now, let's find x * y * z in another way. From the first equation:

x = yz * √2

Multiply both sides by yz:

x * yz = yz * √2 * yz x * yz = y^2z^2 * √2 x = y^2z^2 * √2 / yz x = yz * √2

Wait, that brings us back to the original equation. Maybe I should use the expressions differently.

Let me try to express x, y, z in terms of one variable. Let's assume z = k, where k is a positive real number.

Then, from the third equation:

z = xy * 2^(1/4) k = xy * 2^(1/4) xy = k / 2^(1/4)

From the second equation:

y = xz * 2^(1/3) y = xk * 2^(1/3)

Now, substitute y into the expression for xy:

x * (xk * 2^(1/3)) = k / 2^(1/4) x^2k * 2^(1/3) = k / 2^(1/4)

Divide both sides by k (since k 0):

x^2 * 2^(1/3) = 1 / 2^(1/4)

Multiply both sides by 2^(1/4):

x^2 * 2^(1/3) * 2^(1/4) = 1 x^2 * 2^((1/3) + (1/4)) = 1

(1/3) + (1/4) = 4/12 + 3/12 = 7/12

So, x^2 * 2^(7/12) = 1 x^2 = 1 / 2^(7/12) x = 1 / 2^(7/24)

Now, find y using y = xk * 2^(1/3):

y = (1 / 2^(7/24)) * k * 2^(1/3) = k * 2^((1/3) - (7/24)) = k * 2^((8/24) - (7/24)) = k * 2^(1/24)

Now, find x * y * z:

x * y * z = (1 / 2^(7/24)) * (k * 2^(1/24)) * k = k^2 * 2^((1/24) - (7/24)) = k^2 * 2^(-6/24) = k^2 * 2^(-1/4)

But earlier, we found that x * y * z = 1 / 2^(13/12)

So, k^2 * 2^(-1/4) = 1 / 2^(13/12)

Divide both sides by 2^(-1/4):

k^2 = 1 / (2^(13/12) * 2^(-1/4)) = 1 / 2^((13/12) - (1/4)) = 1 / 2^((13/12) - (3/12)) = 1 / 2^(10/12) = 1 / 2^(5/6)

Thus, k^2 = 1 / 2^(5/6) k = 1 / 2^(5/12)

Now, recall that z = k = 1 / 2^(5/12)

We can find x and y as well:

x = 1 / 2^(7/24)

y = k * 2^(1/24) = (1 / 2^(5/12)) * 2^(1/24) = 2^((1/24) - (5/12)) = 2^((1/24) - (10/24)) = 2^(-9/24) = 2^(-3/8)

Now, we need to find |log ($x y^3 z^2$)|

First, compute $x y^3 z^2$:

x = (1 / 2^(7/24))^4 = 1 / 2^(28/24) = 1 / 2^(7/6)

y^3 = (2^(-3/8))^3 = 2^(-9/8)

z^2 = (1 / 2^(5/12))^2 = 1 / 2^(10/12) = 1 / 2^(5/6)

So, $x y^3 z^2$ = (1 / 2^(7/6)) * 2^(-9/8) * (1 / 2^(5/6)) = 1 / (2^(7/6) * 2^(5/6) * 2^(9/8)) = 1 / (2^((7/6) + (5/6) + (9/8)))

Find a common denominator for the exponents: 6 and 8 have a least common multiple of 24.

So, (7/6) = 28/24, (5/6) = 20/24, (9/8) = 27/24

Thus, (7/6) + (5/6) + (9/8) = (28/24) + (20/24) + (27/24) = 75/24

So, $x y^3 z^2$ = 1 / 2^(75/24)

Now, compute |log ($x y^3 z^2$)|:

|log (1 / 2^(75/24))| = | - (75/24) | = 75/24

Simplify 75/24:

Divide numerator and denominator by 3: 25/8

So, |log ($x y^3 z^2$)| = 25/8

Thus, m = 25, n = 8, and m + n = 33

\boxed{33}

Qwen3-8B-Base after Prefix On-policy Distillation (Correct) with the Original Question

Okay, so I need to solve this system of logarithmic equations and find the value of |log ($x y^3 z^2$)|. The problem gives me three equations involving logs with base 2 of ratios of x, y, and z. Let me write them down again to make sure I have them right:

1. log (x/(yz)) = 1/2
2. log (y/(xz)) = 1/3
3. log (z/(xy)) = 1/4

And then I need to find |log ($x y^3 z^2$)|, which simplifies to |4 log x + 3 log y + 2 log z|. But maybe there's a better way to approach this. Let me think.

First, I remember that log_b(a/c) is equal to log_b(a) - log_b(c). Maybe I can rewrite each equation using that property. Let me try that.

Starting with the first equation:

log (x/(yz)) = log x - log (yz) = log x - log y - log z = 1/2.

Similarly for the second equation:

log (y/(xz)) = log y - log (xz) = log y - log x - log z = 1/3.

Third equation:

log (z/(xy)) = log z - log (xy) = log z - log x - log y = 1/4.

So now I have three equations:

1. log x - log y - log z = 1/2
2. -log x + log y - log z = 1/3
3. -log x - log y + log z = 1/4

Hmm, this looks like a system of linear equations in terms of log x, log y, and log z. Let me denote them as variables for simplicity. Let me let:

a = log x,

b = log y,

c = log z.

Then the equations become:

1. a - b - c = 1/2

2. -a + b - c = 1/3

3. -a - b + c = 1/4

So now I have three equations with three variables. Let me write them again:

Equation 1: a - b - c = 1/2

Equation 2: -a + b - c = 1/3

Equation 3: -a - b + c = 1/4

I need to solve for a, b, c. Let me try to solve this system step by step.

First, maybe I can add equations together to eliminate variables. Let me see.

Let me try adding equations 1 and 2:

Equation 1 + Equation 2:

(a - b - c) + (-a + b - c) = 1/2 + 1/3

Simplify left side: a - a - b + b - c - c = -2c

Right side: 1/2 + 1/3 = 3/6 + 2/6 = 5/6

So -2c = 5/6 => c = -5/12.

Okay, so c = -5/12.

Now let me use equation 1 and equation 3 to find another variable.

Wait, equation 1: a - b - c = 1/2

Equation 3: -a - b + c = 1/4

Let me add equations 1 and 3:

(a - b - c) + (-a - b + c) = 1/2 + 1/4

Left side simplifies: a - a - b - b - c + c = -2b

Right side: 1/2 + 1/4 = 3/4

So -2b = 3/4 => b = -3/8.

So b = -3/8.

Now, since we have c and b, we can plug them back into one of the
↪ equations to solve for a.

Let me use equation 1:

a - b - c = 1/2

So a - (-3/8) - (-5/12) = 1/2

Calculate:

a + 3/8 + 5/12 = 1/2

Convert to common denominators. Let me find the least common
↪ denominator for 8, 12, and 1. Which is 24.

So:

a + (9/24) + (10/24) = 12/24

So a + 19/24 = 12/24

Thus, a = 12/24 - 19/24 = -7/24.

So a = -7/24.

So now we have:

a = $\log x$ = -7/24

b = $\log y$ = -3/8

c = $\log z$ = -5/12

Now, we need to compute |$\log(x y^3 z^2)$|. Let me express that in terms
↪ of a, b, c.

$\log(x y^3 z^2)$ = 4 $\log x$ + 3 $\log y$ + 2 $\log z$ = 4a + 3b + 2c.

So let me compute that:

4a + 3b + 2c = 4*(-7/24) + 3*(-3/8) + 2*(-5/12)

Let me compute each term step by step.

First term: 4*(-7/24) = (-28)/24 = -7/6

Second term: 3*(-3/8) = -9/8

Third term: 2*(-5/12) = -10/12 = -5/6

Now adding them together: -7/6 - 9/8 - 5/6.

Let me convert all terms to sixteenths to add them. Wait, maybe
↪ better to find common denominator.

The denominators are 6, 8, 6. The least common denominator is 24.

Convert each term:

-7/6 = -28/24

-9/8 = -27/24

-5/6 = -20/24

So total is: -28/24 -27/24 -20/24 = (-28 -27 -20)/24 = (-75)/24.

Simplify -75/24. Divide numerator and denominator by 3: -25/8.

Therefore, $\log(x y^3 z^2)$ = -25/8.

But the problem asks for |$\log(x y^3 z^2)$|, which is | -25/8 | = 25/8.
↪ Therefore, m/n is 25/8, and m + n is 25 + 8 = 33.

Wait, but let me check my calculations again because sometimes I
↪ make arithmetic errors.

First, let me check the values of a, b, c again.

From earlier:

c = -5/12. b = -3/8. a = -7/24. Let me check if they satisfy
↪ equation 2 and equation 3.

Let me check equation 2: -a + b - c = 1/3.

Plug in a = -7/24, b = -3/8, c = -5/12.

Compute -a = 7/24, b = -3/8, -c = 5/12.

So sum: 7/24 - 3/8 + 5/12.

Convert all to 24 denominator:

7/24 - 9/24 + 10/24 = (7 - 9 + 10)/24 = 8/24 = 1/3. Correct!

Equation 3: -a - b + c = 1/4.

Plug in:

-a = 7/24, -b = 3/8, c = -5/12.

So sum: 7/24 + 3/8 - 5/12.

Convert to 24 denominator:

7/24 + 9/24 - 10/24 = (7 + 9 - 10)/24 = 6/24 = 1/4. Correct!

Okay, so the values of a, b, c are correct.

Now, compute 4a + 3b + 2c:

4*(-7/24) + 3*(-3/8) + 2*(-5/12) = -28/24 - 9/8 - 5/6.

Convert all to 24 denominator:

-28/24 - 27/24 - 20/24 = (-28 - 27 - 20)/24 = -75/24. Simplify:
↪ divide numerator and denominator by 3: -25/8. So $\log(x y^3 z^2)$ =
↪ -25/8. Then absolute value is 25/8. So m/n is 25/8, which is
↪ already in simplest terms since 25 and 8 are coprime. Therefore,
↪ m + n = 25 + 8 = 33.

But let me check again the calculation steps.

First, 4a: 4*(-7/24) = -7/6. Correct.

Then 3b: 3*(-3/8) = -9/8. Correct.

Then 2c: 2*(-5/12) = -10/12 = -5/6. Correct.

Adding them:

-7/6 - 9/8 - 5/6.

Convert to eighths? Let me convert each term:

-7/6 is -28/24, -9/8 is -27/24, -5/6 is -20/24. Adding gives -75/24.

Yes, which is -25/8. So 25/8. So absolute value is 25/8.

So the answer is 25/8, m + n is 33. Wait, but let me check if I did
↪ the problem correctly. Wait, the problem says to compute the
↪ absolute value. So the value is negative, but the absolute value
↪ is positive.

But maybe I made a mistake in the sign? Wait, let's check the
↪ original problem.

The question is |$\log(x y^3 z^2)$|. So if $\log(x y^3 z^2)$ is negative, then
↪ the absolute value is positive. So 25/8.

Alternatively, maybe my calculation of the coefficients is wrong.
↪ Let me check again:

The expression to compute is $\log(x y^3 z^2)$. Which is $\log x$ + $\log y^3$ +
↪ $\log z^2$ = 4 $\log x$ + 3 $\log y$ + 2 $\log z$. Yes, that is correct.

So 4a + 3b + 2c is -25/8, so absolute value is 25/8.

Wait, but let me check if there's a miscalculation in the addition
↪ of fractions.

Let me do it again step by step:

First term: 4a = 4*(-7/24) = -28/24. Simplify to -7/6.

Second term: 3b = 3*(-3/8) = -9/8.

Third term: 2c = 2*(-5/12) = -10/12 = -5/6.

So sum is (-7/6) + (-9/8) + (-5/6).

Combine the first and third terms:

(-7/6 -5/6) = -12/6 = -2. Then add -9/8.

So -2 - 9/8 = -16/8 - 9/8 = -25/8. Correct.

Therefore, that is correct. So the absolute value is 25/8.

Therefore, the answer is 25/8, so m=25, n=8, m + n = 33.

But let me check if the problem says positive real numbers. Since x,
↪ y, z are positive, all the logs are defined. So that's okay.

Alternatively, maybe there's a mistake in my equations setup. Let me
↪ check the original equations again.

```
Original equations:

1. log(x/(yz)) = 1/2

Which is log x - log y - log z = 1/2. Correct.

Similarly for others. So equations are correct.

Therefore, the answer should be 25/8.

Wait, but the problem says "the value of |log(x y^4z^3)| is m/n". So my conclusion is that m/n is 25/8, so m + n is 33. But let me check once more with another approach. Maybe I can compute x, y, z directly?

Wait, but that might take longer. Let me try.

Let me express x, y, z in terms of exponents. Let me recall that if log x = a, then x = 2^a, similarly for y and z. So:

x = 2^{-7/24}, y = 2^{-3/8}, z = 2^{-5/12}

Then x y^3z^2 = 2^{4a + 3b + 2c}

But we already found that 4a + 3b + 2c = -25/8. Therefore, log(x y^3z^2) = -25/8, so absolute value is 25/8.

Therefore, the answer is 25/8.

Alternatively, maybe I can compute x y^3z^2? Let me check:

x = (2^{-7/24}) = 2^{-28/24} = 2^{-7/6}

y^3 = (2^{-3/8})^3 = 2^{-9/8}

z^2 = (2^{-5/12})^2 = 2^{-10/12} = 2^{-5/6}

Multiply them together:

2^{-7/6} * 2^{-9/8} * 2^{-5/6} = 2^{-7/6 -9/8 -5/6}

Compute the exponents:

Convert to common denominator 24:

-7/6 = -28/24

-9/8 = -27/24

-5/6 = -20/24

Total exponent: -28 -27 -20 = -75/24 = -25/8. So log(x y^3z^2) = -25/8. Therefore, absolute value is 25/8.

Therefore, I think my answer is correct. So 25 + 8 = 33. Therefore, the answer is 33.

Wait, but let me check once more for possible errors in the equations. Let me verify equations with the values of a, b, c.

Equation 1: a - b - c = -7/24 - (-3/8) - (-5/12) = -7/24 + 3/8 + 5/12.

Convert to 24 denominator:

-7/24 + 9/24 + 10/24 = (-7 + 9 + 10)/24 = 12/24 = 1/2. Correct.

Equation 2: -a + b - c = 7/24 - 3/8 - (-5/12) = 7/24 - 3/8 + 5/12.

Convert:

7/24 - 9/24 + 10/24 = (7 - 9 + 10)/24 = 8/24 = 1/3. Correct.

Equation 3: -a - b + c = 7/24 + 3/8 -5/12 = 7/24 + 9/24 - 10/24 = (7 + 9 - 10)/24 = 6/24 = 1/4. Correct.

Therefore, equations are satisfied. Therefore, the calculations are correct.

Therefore, the answer is 25/8, so m + n is 33.

**Final Answer**
\boxed{33}
```

B FLOP Calculation Details

We refer to VERL (Sheng et al., 2024) implementation for Qwen3 models to calculate training flop given a text sequence, which includes forward, backward and gradient updates. [10]. Then we did an approximation of 1/3 of the above training flop for on-policy rollout sampling and log prob calculation for the teacher and the student. In total, let x be the training flop for the student and y for the teacher, the flop for each rollout during on-policy distillation training approximately equals to $1/3 \cdot x + 1/3 \cdot x + 1/3 \cdot y + x$ which represent as flop of rollout generation, student log prob, teacher log prob and training.

[10]We did a modification to remove delta time deviding in order to calculate the cumulative flop https://github.com/volcengine/verl/blob/c936ec7d5cebcc1d40f50296d28699696fbad6f5/verl/utils/flops_counter.py#L131

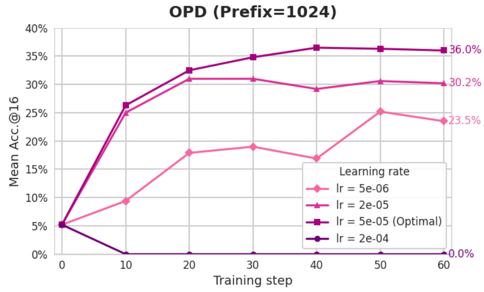

Figure 7: The effect of learning rate on AIME-24 during the training of on-policy prefix distillation on OpenThoughts3. The student is Qwen3-8B-Base and the teacher is Qwen3-8B.

C Effect of Learning Rate

We did learning rate sweep for prefix OPD. As shown in Figure 7, the performances differ significantly with different learning rates. In consistent with Lu and Lab (2025), 5e-5 turns out to be the optimal learning rate with batchsize 512 in our experiment setting.

Prior work reports that small learning rates can mitigate catastrophic forgetting in SFT and lead to better generalize on out-of-domain data (Pareja et al., 2024; Lin et al., 2025). To further evaluate the generalization ability on out-of-domain benchmarks, we *control* the in-domain performance by selecting checkpoints from different learning rates where accuracies on AIME-24 are similar. In our experiments, we select checkpoints from 10-th step where learning rates are 5e-5 and 2e-5, the checkpoint from 50-th step where the learning rate is 5e-6. [11]

Finding 5: Learning rate does not affect significantly on generalization capability for prefix on-policy distillation. Results

[11]We did not compare with learning rate 2e-4 as the learning rate is too large and dramatically hurt the model's performance.

are shown in Table 4. We did not observe out-of-domain performance decrease when using a large learning rate, e.g. 5e-5, in comparison to a smaller learning rate of 2e-5 or 5e-6. We leave extensive analysis on a grid search of learning rate, learning rate scheduling, step size, batch size and prefix length as a future work.

	AIME-24 (control)	MMLU-Pro	GPQA
step 0	5.2%	26.5%	17.6%
lr=5e-5, step 10	26.3%	**55.0%**	**27.9%**
lr=2e-5, step 10	25.0%	49.3%	26.6%
lr=5e-6, step 50	25.2%	47.8%	26.6%

Table 4: Effect of learning rate on generalization capability. We report validation performance after 60 steps of OPD (Prefix=1024) on OpenThoughts3, using Qwen3-8B-Base as the student and Qwen3-8B as the teacher.

D Early Stop with Dev

Following tables contain the detailed information of which checkpoints we used for testing. We use AIME-24 as the development set and then test on the rest of the benchmarks.

Prefix length	Checkpoint for testing
256	Step 20
512	Step 60
1024	Step 60
2048	Step 40
Prefix scheduling	Step 60
Full	Step 50

Table 5: Checkpoints for testing with Qwen3-1.7B-Base as the student.

Prefix length	Checkpoint for testing
256	Step 60
512	Step 50
1024	Step 40
2048	Step 60
Prefix scheduling	Step 60
Full	Step 60

Table 6: Checkpoints for testing with Qwen3-8B-Base as the student.

E Number of Solutions for AIME-2024

Problem ID	Number of Solutions
1	2
2	6
3	5
4	2
5	7
6	7
7	12
8	5
9	2
10	8
11	2
12	3
13	4
14	10
15	6
Average	5.4

Table 7: Number of solutions for each question in AIME-2024 from AoPS forum https://artofproblemsolving.com/wiki/index.php?title=2024_AIME_I