DO WE REALLY NEED KL DIVERGENCE FOR ON-POLICY DISTILLATION OF LARGE LANGUAGE MODELS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Since the advent of knowledge distillation, KL divergence has been the standard loss in distillation. Recently, on-policy distillation (OPD) has emerged as an efficient post-training paradigm for LLMs. As a distillation method, OPD naturally inherits KL divergence as its standard loss. However, in this work, we find that KL divergence may not be necessary for OPD. We show that simply preserving the update direction is sufficient for effective OPD. As long as the update direction is toward the teacher, OPD works. More precisely, it is not the direction of every token, but the direction of a small subset of tokens where the teacher and student disagree strongly. We first show that simply assigning a reward of $+1$ to tokens where the teacher probability is higher than the student probability and -1 where it is lower, which merely encourages updates toward the teacher, reproduces almost the same training mode as OPD with reverse KL. We further show that only the direction of a small subset of tokens with large teacher-student disagreement is critical, and training works as long as their update direction is toward the teacher, even if other tokens are pulled away from the teacher. And as an application of these findings, we introduce Consensus Multi-Teacher On-Policy Distillation (C-MOPD) to improve Multi-Teacher On-Policy Distillation (MOPD). Unlike MOPD, which routes each sample to a single teacher and may cause capability conflicts across domains, C-MOPD lets every sample be supervised by all teachers. Experiments show that C-MOPD consistently outperforms MOPD on both math and code benchmarks.

1 INTRODUCTION

Since the advent of knowledge distillation (Hinton et al., 2015), KL divergence has been the default loss in distillation, as it allows the student to match the teacher's soft targets and thereby capture the dark knowledge encoded in the teacher's output distribution. Recently, on-policy distillation (OPD) (Xu et al., 2026; Xiao et al., 2026; Yang et al., 2025; Zeng et al., 2026) has emerged as an efficient post-training paradigm for LLMs. While classical distillation trains the student on a fixed dataset, OPD aligns the student with the teacher on the student's own rollouts. As a distillation method, OPD naturally inherits KL divergence as its standard loss. Minimizing the reverse KL divergence between the student and the teacher on the student's own rollouts preserves the on-policy property and achieves strong performance (Song & Zheng, 2026; Wu et al., 2025; Zhang et al., 2026; Zhao et al., 2026). In this work, however, we find that KL divergence may not be necessary for OPD. We show that simply preserving the update direction, without its magnitude, is sufficient for effective OPD. As long as the update direction is toward the teacher, OPD works. More precisely, it is not the direction of every token, but the direction of a small subset of tokens where the teacher and student disagree strongly. We first show that simply assigning a reward of $+1$ to tokens where the teacher probability is higher than the student probability, and -1 where the teacher probability is lower than the student probability, which merely encourages updates toward the teacher, reproduces almost the same training mode as OPD with reverse KL and achieves comparable or better results on code and math benchmarks. We further show that it is not the direction of every token that matters, but only that of a small critical subset of tokens with large teacher-student disagreement: OPD works as long as their update direction is toward the teacher, and OPD fails as long as their update direction is away

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.

from the teacher. In our experiments, we find that keeping only high-disagreement tokens, which in some cases account for less than 2% of all tokens, and ensuring their update direction is toward the teacher reproduces OPD; once their direction is reversed, OPD fails to train. We also find that the low-disagreement tokens, which constitute the vast majority of tokens, can be updated away from the teacher without severely affecting OPD. These findings challenge the prevailing assumption that KL divergence is essential for on-policy distillation, and suggest that the key ingredient is not distribution matching but merely directional updates on a small subset of high-disagreement tokens.

As an application of these findings, we propose Consensus Multi-Teacher On-Policy Distillation (C-MOPD) to improve Multi-Teacher On-Policy Distillation (MOPD) (Ma et al., 2026; Blakeman et al., 2025; Team et al., 2026). In MOPD, for each sample from the student's own rollouts, the student computes the reverse KL divergence against a domain-specific teacher and updates accordingly. As the number of samples increases, the student integrates the capabilities of different teachers. But a drawback is that updating on samples from one domain may degrade the model's capability in another domain. In C-MOPD, every update is guided by all teachers: when all teachers agree on the update direction, we pull the student toward all of them; when the teachers conflict, we fall back to the domain-specific teacher and adopt its direction only if it does not strongly conflict with the other teachers, and otherwise we assign a zero reward. This design is motivated by our earlier finding that in OPD, tokens with small teacher-student disagreement can be updated away from the teacher without hurting performance. In this way, C-MOPD ensures that each update either pulls toward all teachers, or toward the domain-specific teacher without hurting the capabilities associated with the other teachers' directions. Experiments show that C-MOPD consistently outperforms MOPD.

Our contributions can be summarized as below:

- We find that simply ensuring the update direction is toward the teacher is sufficient for OPD, questioning whether KL divergence is necessary for OPD.
- We further find that only the direction of tokens with large teacher-student disagreement is critical. In some cases, keeping only less than 2% of the tokens while ensuring their direction is toward the teacher makes OPD work.
- We propose Consensus Multi-Teacher On-Policy Distillation (C-MOPD) to improve MOPD, which guides every token with all teachers, consistently outperforming MOPD.

2 Related Work

On-Policy Distillation with Reverse KL. On-policy Distillation (OPD) has recently emerged as an efficient post-training paradigm that provides dense, token-level supervisory signals by distilling a teacher model's output distribution into the student (Xu et al., 2026; Xiao et al., 2026; Yang et al., 2025; Zeng et al., 2026; Agarwal et al., 2024; Li et al., 2026). The de facto loss for OPD is reverse KL divergence, which is mode-seeking and preserves the on-policy property (Zhao et al., 2026; Agarwal et al., 2024; Shao et al., 2026; Song & Zheng, 2026; Wu et al., 2025; Zhang et al., 2026). In this work, however, we question whether reverse KL is truly necessary for effective teacher guidance. We find that simply assigning $+1$ to tokens where the teacher probability is higher than the student probability and -1 where it is lower already reproduces almost the same training mode as OPD, with comparable or better results.

Multi-Teacher On-Policy Distillation. Multi-teacher on-policy distillation (MOPD) has become a common approach for integrating the capabilities of multiple teacher models into a single student (Ma et al., 2026; Blakeman et al., 2025; Team et al., 2026; Gao et al., 2026; Chen et al., 2026; Sun et al., 2026; He et al., 2026a). In MOPD, each sample is routed to a single domain-specific teacher, so that the student gradually acquires different teachers' expertise as it sees more samples. However, based on our observation that OPD only needs to maintain updates toward the teacher, we propose Consensus Multi-Teacher On-Policy Distillation (C-MOPD), which allows every sample to receive guidance from all teachers. This avoids the loss of one domain's capability when updating on samples from another domain.

3 PRELIMINARY

On-Policy Distillation with Reverse KL. On-Policy Distillation (OPD) aims to transfer the capability of a teacher policy π_T to a student policy π_θ by aligning the student with the teacher on the student's own rollouts. Formally, given a query q, the student generates an output sequence $o \sim \pi_\theta(\cdot \mid q)$, and the default objective is to minimize the reverse KL divergence between the two policies at every token position:

$$\min_\theta \ \mathbb{E}_{q,\, o\sim\pi_\theta(\cdot|q)} \left[\sum_{t=1}^{|o|} D_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid q, o_{<t}) \,\big\|\, \pi_T(\cdot \mid q, o_{<t})\big) \right].$$

Because the expectation is taken over trajectories sampled from π_θ itself, the training remains on-policy and benefits from dense token-level supervision. To derive the update rule, one can differentiate the objective and obtain a policy-gradient-like expression:

$$\nabla_\theta \mathcal{L}_{\mathrm{OPD}}(\theta) = \mathbb{E}_{q,\, o\sim\pi_\theta(\cdot|q)} \left[\sum_{t=1}^{|o|} \Big(\log \pi_T(o_t \mid q, o_{<t}) - \log \pi_\theta(o_t \mid q, o_{<t}) \Big) \nabla_\theta \log \pi_\theta(o_t \mid q, o_{<t}) \right].$$

This expression shows that each token receives a dense reward, which is exactly the log-ratio

$$r_t = \log \frac{\pi_T(o_t \mid q, o_{<t})}{\pi_\theta(o_t \mid q, o_{<t})}.$$

Multi-Teacher On-Policy Distillation. Multi-Teacher On-Policy Distillation (MOPD) extends OPD to multiple domain-specific teachers. Let $\{\pi_{T_1}, \ldots, \pi_{T_K}\}$ denote K teachers, each specialized in a different domain. For each query q, a routing function $d(q) \in \{1, \ldots, K\}$ selects the teacher corresponding to the domain of q. The student is then optimized to minimize the reverse KL divergence against the routed teacher on its own rollouts:

$$\min_\theta \ \mathbb{E}_{q,\, o\sim\pi_\theta(\cdot|q)} \left[\sum_{t=1}^{|o|} D_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid q, o_{<t}) \,\big\|\, \pi_{T_{d(q)}}(\cdot \mid q, o_{<t})\big) \right].$$

4 TEACHER-DIRECTIONAL UPDATES SUFFICE FOR ON-POLICY DISTILLATION

4.1 METHOD

We replace the reverse KL objective in OPD with a simple directional reward.

Given a query q, the student generates a rollout $o \sim \pi_\theta(\cdot \mid q)$. For each token o_t, we compare the teacher probability $\pi_T(o_t \mid q, o_{<t})$ with the student probability $\pi_\theta(o_t \mid q, o_{<t})$. We assign a reward

$$r_t = \begin{cases} +1, & \text{if } \pi_T(o_t \mid q, o_{<t}) > \pi_\theta(o_t \mid q, o_{<t}), \\ -1, & \text{if } \pi_T(o_t \mid q, o_{<t}) < \pi_\theta(o_t \mid q, o_{<t}). \end{cases}$$

Specifically, when the teacher probability is higher than the student probability, the $+1$ reward increases the probability of that token in the student, thereby pulling the student toward the teacher. When the teacher probability is lower than the student probability, the -1 reward decreases the probability of that token in the student, again moving the student toward the teacher. In summary, this reward merely ensures that the student updates in the direction of the teacher, without any magnitude information. Note that the case where the two probabilities are exactly equal is almost impossible in practice, so we do not include it in the formula above; in our implementation, we assign a reward of 0 to such tokens.

The student is then optimized with a policy gradient objective:

$$\max_\theta \ \mathbb{E}_{q,\, o\sim\pi_\theta(\cdot|q)} \left[\sum_{t=1}^{|o|} r_t \log \pi_\theta(o_t \mid q, o_{<t}) \right].$$

We refer to this method as **BinaryOPD**.

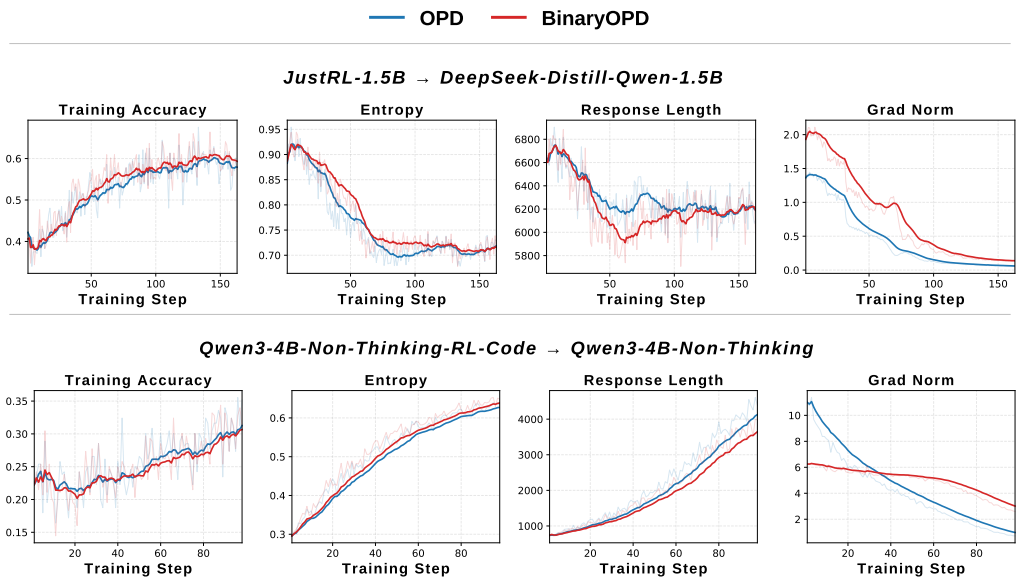


Figure 1: Training dynamics of OPD and BinaryOPD on one math student-teacher pair and one code pair. Full training dynamics are provided in Appendix D.

4.2 Experiments

4.2.1 Experimental Setup

We compare standard OPD against our BinaryOPD on both mathematical reasoning and code generation tasks. For each student-teacher pair, we train the student with either the reverse KL divergence (OPD) or the binary-directional reward (BinaryOPD) on the same dataset, using identical training hyperparameters, rollout budgets, and optimization settings.

For math, we use five student-teacher pairs covering different model families and scales. For code, we use two student-teacher pairs. The training datasets are DAPO-Math-17k (Yu et al., 2026), DeepMath (He et al., 2026b), and Eurus (Cui et al., 2025). Table 1 summarizes all pairs.

Table 1: Student-teacher pairs and training datasets used in our experiments.

Student	Teacher	Dataset
Math		
DeepSeek-Distill-Qwen-1.5B	JustRL-1.5B	DAPO-Math-17k
Qwen3-1.7B-Base	Qwen3-4B-Base-RL	DAPO-Math-17k
Llama-3.2-3B-Instruct	GT-Llama3.2-3B-MATH	DeepMath
Qwen3-4B-Non-Thinking	Qwen3-4B-Non-Thinking-RL-Math	DeepMath
Qwen3-30B-A3B-Non-Thinking	Qwen3-30B-A3B-Instruct-2507	DeepMath
Code		
Qwen3-4B-Non-Thinking	Qwen3-4B-Non-Thinking-RL-Code	Eurus
DeepSeek-Distill-Qwen-1.5B	Nemotron-Research-Reasoning-1.5B	Eurus

For math, we evaluate on AIME24, AIME25, AMC, MATH500 (Lightman et al., 2023), Minerva (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024). For code, we evaluate on LiveCodeBench v6 (Jain et al., 2025), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021). Experimental details are provided in Appendix C.

Table 2: Math benchmark results. All results are averaged over 8 samples (Avg@8).

Method	AIME24	AIME25	AMC	MATH500	Minerva	Olympiad	Avg
DeepSeek-Distill-Qwen-1.5B (Student) vs. JustRL-1.5B (Teacher)							
Student	23.7	17.0	59.7	80.9	23.7	41.4	41.0
Teacher	46.2	37.5	82.0	89.2	30.7	53.5	56.5
OPD	43.8	**32.5**	**78.8**	88.5	**31.7**	52.6	54.7
BinaryOPD	**45.4**	32.1	78.5	**88.9**	31.6	52.6	**54.9**
Qwen3-1.7B-Base (Student) vs. Qwen3-4B-Base-RL (Teacher)							
Student	4.1	1.7	23.2	48.9	8.9	17.1	17.3
Teacher	10.6	13.1	40.3	74.2	17.2	30.0	30.9
OPD	7.5	3.3	26.4	57.8	**11.4**	20.6	21.2
BinaryOPD	**8.1**	**4.0**	**30.3**	**59.1**	10.0	**21.6**	**22.2**
Llama-3.2-3B-Instruct (Student) vs. GT-Llama3.2-3B-MATH (Teacher)							
Student	2.9	0.0	14.8	34.5	7.7	7.9	11.3
Teacher	11.7	0.0	20.9	49.1	15.7	15.4	18.8
OPD	**12.5**	**0.4**	**22.7**	46.0	13.3	14.3	**18.2**
BinaryOPD	9.2	0.0	21.7	**47.2**	**14.8**	**14.6**	17.9
Qwen3-4B-Non-Thinking (Student) vs. Qwen3-4B-Non-Thinking-RL-Math (Teacher)							
Student	25.0	16.7	58.4	81.3	27.7	42.9	42.0
Teacher	61.2	56.3	88.0	93.2	36.7	60.5	66.0
OPD	60.4	51.7	85.7	**93.0**	34.5	59.2	64.1
BinaryOPD	**61.3**	**55.0**	**85.7**	92.8	**35.9**	**59.8**	**65.1**
Qwen3-30B-A3B-Non-Thinking (Student) vs. Qwen3-30B-A3B-Instruct-2507 (Teacher)							
Student	33.3	20.0	65.2	85.9	32.8	51.2	48.1
Teacher	74.1	62.1	90.2	95.7	34.3	66.8	70.5
OPD	46.7	**31.3**	75.3	**89.6**	33.4	52.7	54.8
BinaryOPD	**47.5**	30.8	**76.1**	89.4	**34.0**	**53.1**	**55.2**

Table 3: Code benchmark results. All results are averaged over 8 samples (Avg@8).

Method	LiveCodeBench v6	HumanEval	MBPP	Avg
Qwen3-4B-Non-Thinking (Student) vs. Qwen3-4B-Non-Thinking-RL-Code (Teacher)				
Student	25.1	80.8	48.7	51.5
Teacher	36.0	90.6	56.2	60.9
OPD	**27.9**	85.4	**53.0**	**55.4**
BinaryOPD	27.8	**85.7**	52.1	55.2
DeepSeek-Distill-Qwen-1.5B (Student) vs. Nemotron-Research-Reasoning-1.5B (Teacher)				
Student	18.0	66.1	40.7	41.6
Teacher	28.6	80.0	47.2	51.9
OPD	**27.7**	79.6	45.6	**51.0**
BinaryOPD	27.6	**78.1**	**46.1**	50.6

4.2.2 Experimental Results

We present the main benchmark results in Table 2 (math) and Table 3 (code), and the training dynamics in Figure 1.

Preserving the update direction is sufficient. Across all student-teacher pairs, BinaryOPD, which replaces the reverse KL objective with a simple binary directional reward, achieves performance comparable to or slightly better than OPD. For example, on the Qwen3-4B-Non-Thinking math pair, BinaryOPD reaches an average score of 65.1 versus 64.1 for OPD; on the Qwen3-1.7B-Base math pair, BinaryOPD achieves 22.2 versus 21.2. On the code benchmark, BinaryOPD and OPD are nearly identical (55.2 vs. 55.4). The training dynamics in Figure 1 further confirm this: BinaryOPD and OPD exhibit the same trend in training accuracy, entropy, and response length. This indicates that

the directional signal alone—whether the teacher prefers a token more or less than the student—is sufficient to reproduce the OPD training mode, and that the magnitude information in reverse KL may be unnecessary.

4.3 WHY CAN BINARYOPD WORK? A REPEATED-FEEDBACK VIEW.

Unlike conventional distillation, OPD repeatedly obtains fresh teacher supervision on states visited by the evolving student. Recent work has shown that even a small number of queries can cover much of the state space visited by full-data OPD, suggesting substantial redundancy in OPD states (Fu et al., 2026). We further show in Appendix A that this redundancy is intrinsic to the training process itself: state exploration saturates almost immediately, and states encountered later in training repeatedly return to regions already visited much earlier.

This changes how an OPD update should be interpreted. If a state were observed only once, the magnitude of the teacher–student discrepancy would be needed to determine how far the student should move. Under repeated visitation, however, supervision becomes a closed-loop feedback process. OPD only needs to indicate whether the student probability is above or below the teacher probability. The same state region is then revisited after the student has changed, producing a fresh directional signal; once the student crosses the teacher, the direction automatically reverses. Repeated directional feedback can therefore determine the required amount of correction iteratively, without explicitly encoding its magnitude in any individual update. From this view, the redundancy of OPD states makes the precise KL magnitude not merely replaceable, but unnecessary.

5 ONLY THE DIRECTION OF HIGH-DISAGREEMENT TOKENS IS CRITICAL FOR OPD

Our previous results show that the update direction toward the teacher matters more than its magnitude. In this section, we show that only the direction of a small subset of tokens is critical: OPD works as long as the update direction of a very small subset of tokens with large teacher-student disagreement is toward the teacher.

5.1 METHOD

We use the log-ratio to represent the token-level disagreement between the teacher and the student: $\ell_t = \log \frac{\pi_T(o_t|q,o_{<t})}{\pi_\theta(o_t|q,o_{<t})}$. A large positive ℓ_t means the teacher strongly prefers this token over the student, while a large negative ℓ_t means the student strongly overestimates it relative to the teacher. Based on a threshold $\epsilon > 0$, we partition tokens into three groups:

- **Group A** (high disagreement, teacher prefers more): $\ell_t > \epsilon$.
- **Group B** (low disagreement): $-\epsilon \leq \ell_t \leq \epsilon$.
- **Group C** (high disagreement, student prefers more): $\ell_t < -\epsilon$.

Groups A and C together form the high-disagreement tokens, while Group B contains the vast majority of tokens with low disagreement.

To test which tokens are critical, we run two symmetric experiments with $\epsilon = 0.8$. In the **positive reward** experiment, we assign $+1$ uniformly to the selected tokens and consider three subsets: (i) only Group A, (ii) Groups A and B, (iii) Groups A, B, and C. In the **negative reward** experiment, we assign -1 uniformly to the selected tokens and consider three subsets: (i) only Group C, (ii) Groups C and B, (iii) Groups C, B, and A. All unselected tokens receive 0.

In the positive reward experiment, Group A tokens are those where the teacher probability is higher than the student probability, so assigning $+1$ moves the student toward the teacher; Group C tokens are those where the student probability is higher than the teacher probability, so assigning $+1$ moves the student away from the teacher. In the negative reward experiment, the roles are reversed.

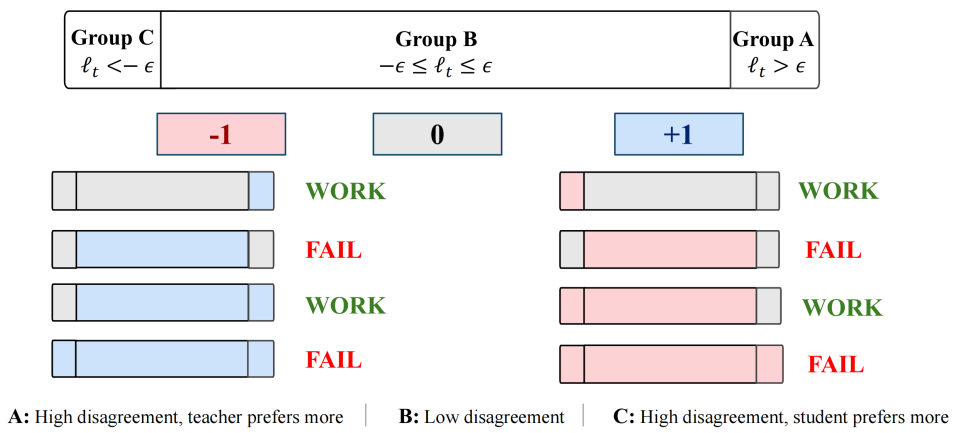
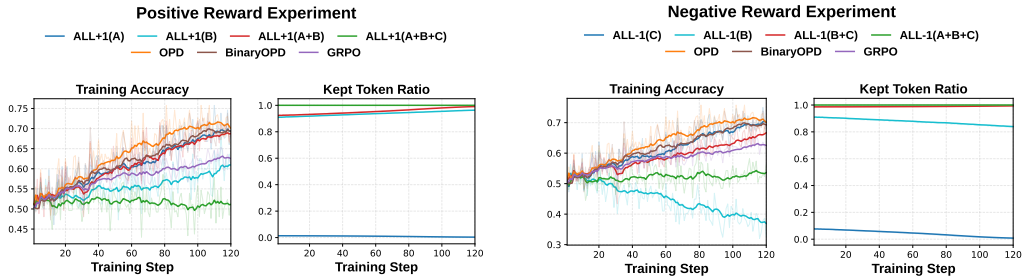


Figure 2: Summary of the positive and negative reward experiments. The positive experiment shows that keeping only Group A or Groups A+B leads to successful training, while adding Group C causes failure. The negative experiment exhibits a symmetric pattern.



(a) Positive reward experiment

(b) Negative reward experiment

Figure 3: Training dynamics for the positive and negative reward experiments with $\epsilon = 0.8$, using Qwen3-4B-Non-Thinking-RL-Math as the teacher and Qwen3-4B-Non-Thinking as the student.

5.2 Results

Figure 3 shows the training dynamics for the Qwen3-4B-Non-Thinking-RL-Math teacher and Qwen3-4B-Non-Thinking student with $\epsilon = 0.8$. As shown, Group B (low disagreement) accounts for over 90% of all tokens, while Groups A and C (high disagreement) each account for only a small fraction.

The results are surprising. In the positive reward experiment:

- **Only Group A.** Training works. This setting keeps less than 1.5% of tokens (Group A) and assigns $+1$ to them.
- **Only Group B.** Training improves but underperforms GRPO, which is also considered a failure for OPD's expected effectiveness.
- **Groups A and B.** Training works. This setting assigns $+1$ to over 90% of the tokens (Groups A and B).
- **Groups A, B, and C.** Training fails. Compared with the working setting of Groups A and B, this setting only adds Group C, which accounts for less than 5% of all tokens.

In the negative reward experiment, the symmetric pattern holds:

- **Only Group C.** Training works. This setting keeps only Group C and assigns -1 to them.
- **Only Group B.** Training fails.
- **Groups C and B.** Training works and surpasses GRPO, although slightly worse than keeping only Group C. This setting assigns -1 to over 90% of the tokens (Groups C and B).

Table 4: Math benchmark results for the positive reward experiment (avg@8, %). Note that **ALL+1(A) keeps less than 1.5% of all tokens.**

Method	AIME24	AIME25	AMC	MATH500	Minerva	Olympiad	Avg
OPD	60.4	51.7	85.7	93.0	34.5	59.2	64.1
BinaryOPD	61.3	55.0	85.7	92.8	35.9	59.8	65.1
ALL+1(A)	60.8	52.0	85.9	92.9	35.0	59.5	64.5
ALL+1(A+B)	60.5	51.5	86.0	92.7	34.8	59.0	64.3
ALL+1(A+B+C)	24.0	15.5	56.0	79.5	26.0	41.0	40.3

- **Groups C, B, and A.** Training fails. Compared with the working setting of Groups C and B, this setting only adds Group A, which accounts for less than 5% of all tokens.

These results show that the critical tokens are the small fraction of high-disagreement tokens in Groups A and C. As long as their update direction is toward the teacher, OPD works. Once their direction is pulled away from the teacher, OPD fails. In contrast, the vast majority of low-disagreement tokens in Group B are not critical: removing them does not break OPD, and including them with an incorrect reward direction may cause some degradation but still allows training to proceed, as long as the high-disagreement tokens are handled correctly. We also conduct experiments with $\epsilon = 0.2$ and 0.5, as well as another student-teacher pair (JustRL-1.5B and DeepSeek-Distill-Qwen-1.5B). Complete experiments are provided in Appendix D.

6 Consensus Multi-Teacher On-Policy Distillation

6.1 Method

Our observation that OPD only needs updates toward the teacher, and that the majority of tokens, which do not have large teacher-student disagreement, can be pulled away from the teacher without hurting performance, suggests that in the multi-teacher setting we need not route each sample to a single teacher. Instead, we can guide every token with all teachers simultaneously, and when teachers conflict, we can still update toward the domain-specific teacher while ensuring that this update does not hurt the capabilities associated with other teachers' directions. Based on this idea, we propose **Consensus Multi-Teacher On-Policy Distillation (C-MOPD)**.

Let $\{\pi_{T_1}, \ldots, \pi_{T_K}\}$ be K teachers. For a query q, the student generates a rollout $o \sim \pi_\theta(\cdot \mid q)$. At each token o_t, let

$$p_s = \pi_\theta(o_t \mid q, o_{<t}), \qquad p_k = \pi_{T_k}(o_t \mid q, o_{<t}), \quad k = 1, \ldots, K.$$

Let k^* be the index of the domain-specific teacher for this sample, and define

$$\ell_k = \log(p_k / p_s), \qquad d = \operatorname{sign}(\ell_{k^*}) \in \{+1, -1\}.$$

We assign the reward as follows:

$$r_t = \begin{cases} +1, & \text{if } p_k > p_s \ \forall k, \\ -1, & \text{if } p_k < p_s \ \forall k, \\ +1, & \text{if teachers conflict, } d = +1, \text{ and } \ell_k > -\epsilon \ \forall k \neq k^*, \\ -1, & \text{if teachers conflict, } d = -1, \text{ and } \ell_k < \epsilon \ \forall k \neq k^*, \\ 0, & \text{otherwise.} \end{cases}$$

This design ensures that each update either pulls toward all teachers, or toward the domain-specific teacher without hurting the capabilities associated with the other teachers' directions.

6.2 Experiments

6.2.1 Experimental Setup

We evaluate C-MOPD in a multi-teacher setting with two domain-specific teachers: a math teacher and a code teacher. The student model is Qwen3-4B-Non-Thinking. The math teacher is Qwen3-4B-Non-

Table 5: Math and code benchmark results for C-MOPD (avg@8, %).

	Math						Code		
Model	**AIME24**	**AIME25**	**AMC**	**MATH500**	**Minerva**	**Olympiad**	**LCB v6**	**HumanEval**	**MBPP**
Student	25.0	16.7	58.4	81.3	27.7	42.9	25.1	80.8	48.7
Math Teacher	61.2	56.3	88.0	93.2	36.7	60.5	–	–	–
Code Teacher	–	–	–	–	–	–	36.0	90.6	56.2
MOPD	60.0	52.5	86.4	92.9	34.3	58.8	27.9	85.9	53.5
C-MOPD ($\epsilon = 0.0$)	59.8	52.8	86.2	92.8	34.6	58.9	28.2	85.7	53.8
C-MOPD ($\epsilon = 0.8$)	**61.3**	**53.9**	**88.0**	**93.0**	**35.5**	**60.4**	**32.9**	**87.8**	**54.3**

Thinking-RL-Math, and the code teacher is Qwen3-4B-Non-Thinking-RL-Code. The training set is a mixture of math and code problems, containing 25,276 DeepMath problems, 9,639 CodeContests problems, 9,579 TACO problems, 3,462 APPS problems, and 2,596 Codeforces problems. We compare C-MOPD against MOPD. Based on our empirical findings in Section 5, we set $\epsilon = 0.8$ in C-MOPD by default. We also compare with a conservative setting of $\epsilon = 0.0$.

6.2.2 Experimental Results

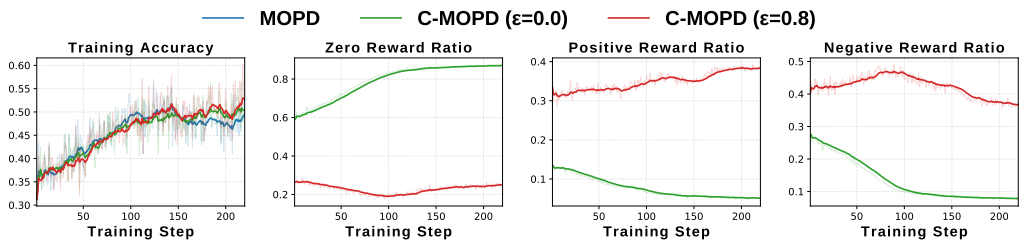


Figure 4: Training dynamics of C-MOPD.

Table 5 shows that C-MOPD with $\epsilon = 0.8$ consistently outperforms MOPD across almost all math and code benchmarks. In contrast, $\epsilon = 0.0$ is overly conservative: its Zero Reward Ratio increases monotonically and eventually exceeds 90%, meaning that as training proceeds, an increasing number of tokens become conflicting across teachers, so the model refuses to update on most of them, and its performance only matches MOPD. This result reinforces our earlier finding: tokens with small teacher-student disagreement can be updated away from the teacher without severely hurting performance. By allowing updates toward the domain teacher when the conflict is mild, $\epsilon = 0.8$ exploits this observation and achieves the best overall results.

7 Conclusion

In this work, we revisit the role of reverse KL divergence in on-policy distillation (OPD). We show that KL divergence may not be necessary: a simple binary directional reward, which only ensures updates toward the teacher, reproduces OPD-like training and achieves comparable or better results. We further find that only a small subset of tokens with large teacher-student disagreement is critical, and that the verifier signal has little effect on OPD. As an application, we propose C-MOPD, which consistently outperforms MOPD on both math and code benchmarks. We hope this work encourages a rethinking of KL's role in LLM post-training and opens the door to simpler distillation objectives.

AI USE STATEMENT

In this work, we used generative AI tools for the following required-disclosure tasks: implement methods, clean and reformat dataset, assist with translation and support qualitative and thematic data analysis.

We have not used generative AI tools for the following other required-disclosure tasks that were relevant to our research but performed entirely by the authors: help develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on research methodology or experiments.

The remaining required-disclosure tasks are not applicable to this work because our study did not involve them: generate synthetic data sets, assist in the writing of proofs, provide critical ingredients for proving mathematical claims and formulate mathematical claims.

Additionally, we used generative AI tools for the following recommended-disclosure tasks: draft parts of a research paper, edit a research paper to improve readability, format references, and create or edit software code (for utility scripts and visualisation functions).

We have reviewed all AI-assisted work. Specifically:

- All AI-generated code was tested on multiple runs, verified for correctness against expected outputs, and peer-reviewed by at least two authors.
- The cleaned dataset was manually inspected for consistency and missing-value handling after AI-assisted preprocessing.
- AI-suggested steps in mathematical proofs were independently re-derived and cross-checked by the authors.
- AI-drafted text was substantially revised to match our scientific voice and checked for factual accuracy against our experimental results.

We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI. The core research ideas, experimental design, and final interpretation of results were performed solely by the authors without AI involvement.

ETHICS STATEMENT

This paper studies post-training algorithms for LLMs. It does not involve human subjects, personal data, or privacy-sensitive information. All datasets used are publicly available and contain mathematical problems. We foresee no ethical concerns and declare no conflicts of interest.

REPRODUCIBILITY STATEMENT

We have made extensive efforts to ensure the reproducibility of our work. All models, datasets, and evaluation benchmarks used in this paper are publicly available and are described in detail in Appendix C. The hyperparameters for all experiments are listed in Table 9, and the hardware setup is provided in Appendix F. For the proposed BinaryOPD, the algorithmic details are given in Section 4.1. For C-MOPD, the method is described in Section 6.1, and the full training dynamics for all methods across all student-teacher pairs are provided in Appendix D.

REFERENCES

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

Mohammad Sadegh Akhondzadeh, Vijay Lingam, Atula Tejaswi, Chanakya Ekbote, Sujay Sanghavi, and Aleksandar Bojchevski. Reward-gated on-policy distillation. *arXiv preprint arXiv:2607.04037*, 2026.

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

Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, et al. Nvidia nemotron 3: Efficient and open intelligence. *arXiv preprint arXiv:2512.20856*, 2025.

Qiye Cai, Yichuan Ma, Linyang Li, Peiji Li, Yongkang Chen, Qipeng Guo, Yicheng Zou, Tao Gui, Xiaocheng Feng, and Bing Qin. H2 sd: Hybrid hindsight self-distillation. *arXiv preprint arXiv:2607.18955*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Tianlei Chen, Jiao Ou, Ziyuan Liu, Ruiming Tang, Jian Liang, and Han Li. Counteraction-aware multi-teacher on-policy distillation for general capability recovery with domain preservation. *arXiv preprint arXiv:2605.27115*, 2026.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Yifan Ding, Xincheng Wei, Yoshua Y Li, Ziheng Li, Yuquan Lu, Siyu Zhang, Dongsheng Ma, Rongxiang Weng, Xunliang Cai, and Yun Chen. Saf-opd: Stable advantage fusion for on-policy distillation. *arXiv preprint arXiv:2607.29209*, 2026.

Zixuan Fu, Bingxiang He, Yuxin Zuo, Haohuan Huang, Jinqian Zhang, Ruhang Xiao, Cheng Qian, Qinyu Luo, Huan-ang Gao, Yudong Wang, et al. Rethinking on-policy distillation of large language models ii: One training example. *arXiv preprint arXiv:2609.04172*, 2026.

Huan-ang Gao, Haohan Chi, Yong Yan, Shiyuan Feng, Hanlin Wu, Zheng Jiang, Bingxiang He, Wei-Ying Ma, Ya-Qin Zhang, and Hao Zhou. Open-mopd: Diagnosing and fixing capability imbalance in multi-teacher on-policy distillation. *arXiv preprint arXiv:2608.19098*, 2026.

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

Bingxiang He, Zekai Qu, Zeyuan Liu, Yinghao Chen, Yuxin Zuo, Cheng Qian, Kaiyan Zhang, Weize Chen, Chaojun Xiao, Ganqu Cui, et al. Justrl: Scaling a 1.5 b llm with a simple rl recipe. *arXiv preprint arXiv:2512.16649*, 2025.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Xixiang He, Xingming Li, Baiqi Wu, Qiyao Sun, Xuanyu Ji, Ao Cheng, and Qingyong Hu. Learn from whoever is right: Answer-verified multi-teacher distillation for multi-domain llms. *arXiv preprint arXiv:2609.02548*, 2026a.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 138306–138322, 2026b.

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

Wenjin Hou, Shangpin Peng, Weinong Wang, Zheng Ruan, Yue Zhang, Zhenglin Zhou, Mingqi Gao, Yifei Chen, Kaiqi Wang, Hongming Yang, et al. Uni-opd: Unifying on-policy distillation with a dual-perspective recipe. *arXiv preprint arXiv:2605.03677*, 2026.

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

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Wenze Lin, Jiale Zhao, Xitai Jiang, Songde Rao, Yining Li, Shenzhi Wang, Bingxiang He, and Gao Huang. On-policy distillation with verifiable reward. *arXiv preprint arXiv:2608.24696*, 2026.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *Advances in Neural Information Processing Systems*, 38:17998–18031, 2026.

Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, Qibin Yang, Bofei Gao, Yudong Wang, Rang Li, et al. Mopd: Multi-teacher on-policy distillation for capability integration in llm post-training. *arXiv preprint arXiv:2606.30406*, 2026.

Bing Shao, Jiazheng Zhang, Long Ma, Yujiong Shen, Senjie Jin, Xin Guo, Yuming Yang, Mingxu Chai, Zhiheng Xi, Boyang Liu, et al. A token-level analysis of sampled-token reverse-kl on-policy distillation. *arXiv preprint arXiv:2608.25643*, 2026.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Mingyang Song and Mao Zheng. A survey of on-policy distillation for large language models. *arXiv preprint arXiv:2604.00626*, 2026.

Zechen Sun, Zhiwei Zhang, Fei Zhao, Juntao Li, Mu Chuan, Huayu Deng, Guojian Zhan, Wenliang Chen, Yao Hu, and Min Zhang. D^3-mopd: Adaptive dynamic domain scheduling for efficient multi-teacher distillation. *arXiv preprint arXiv:2608.24987*, 2026.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Chen Wang, Zhaochun Li, Jionghao Bai, Yining Zhang, Hexuan Deng, Ge Lan, and Yue Wang. Distilled reinforcement learning for llm post-training. *arXiv preprint arXiv:2607.17247*, 2026.

Taiqiang Wu, Chaofan Tao, Jiahao Wang, Runming Yang, Zhe Zhao, and Ngai Wong. Rethinking kullback-leibler divergence in knowledge distillation for large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 5737–5755, 2025.

Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

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

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled rlvr. *arXiv preprint arXiv:2604.03128*, 2026a.

Wenkai Yang, Weijie Liu, Ruobing Xie, Kai Yang, Saiyong Yang, and Yankai Lin. Learning beyond teacher: Generalized on-policy distillation with reward extrapolation. *arXiv preprint arXiv:2602.12125*, 2026b.

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

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

Dongxu Zhang, Zhichao Yang, Sepehr Janghorbani, Jun Han, Andrew Ressler II, Qian Qian, Gregory D Lyng, Sanjit Singh Batra, and Robert E Tillman. Fast and effective on-policy distillation from reasoning prefixes. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 25553–25569, 2026.

Zizhuo Zhang, Jianing Zhu, Xinmu Ge, Zihua Zhao, Zhanke Zhou, Xuan Li, Xiao Feng, Jiangchao Yao, and Bo Han. Co-rewarding: Stable self-supervised rl for eliciting reasoning in large language models. *arXiv preprint arXiv:2508.00410*, 2025.

Anhao Zhao, Haoran Xin, Yingqi Fan, Junlong Tong, Wenjie Li, and Xiaoyu Shen. Decoupling kl and trajectories: A unified perspective for sft, dagger, offline rl, and opd in llm distillation. *arXiv preprint arXiv:2605.16826*, 2026.

A State Exploration Is Highly Redundant in OPD Training

We show that OPD training visits a highly redundant state space that changes little over training. Each state is unique at the token level, but states from different training steps and prompts often occupy nearby regions in a fixed teacher representation space. Increasing the number of observed states gives only small gains in covering future states. This suggests limited expansion of the represented state space during training.

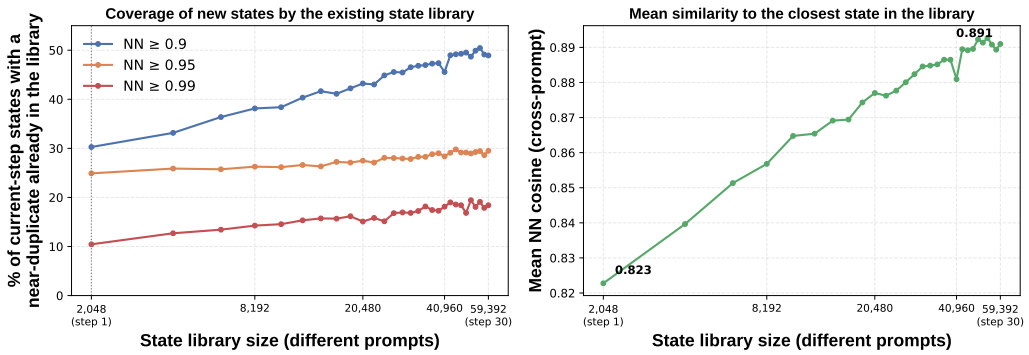


Figure 5: **State-space coverage grows slowly despite a rapidly expanding state library.** For each current state, we search only among previously visited states from different prompts. *Left:* fraction of current-step states whose closest earlier neighbor exceeds cosine thresholds 0.9, 0.95, and 0.99, as a function of library size. Expanding the library by $29\times$ raises $\cos \geq 0.99$ coverage from 10.4% to 18.4%. *Right:* mean similarity to the closest earlier state increases from 0.823 to 0.891.

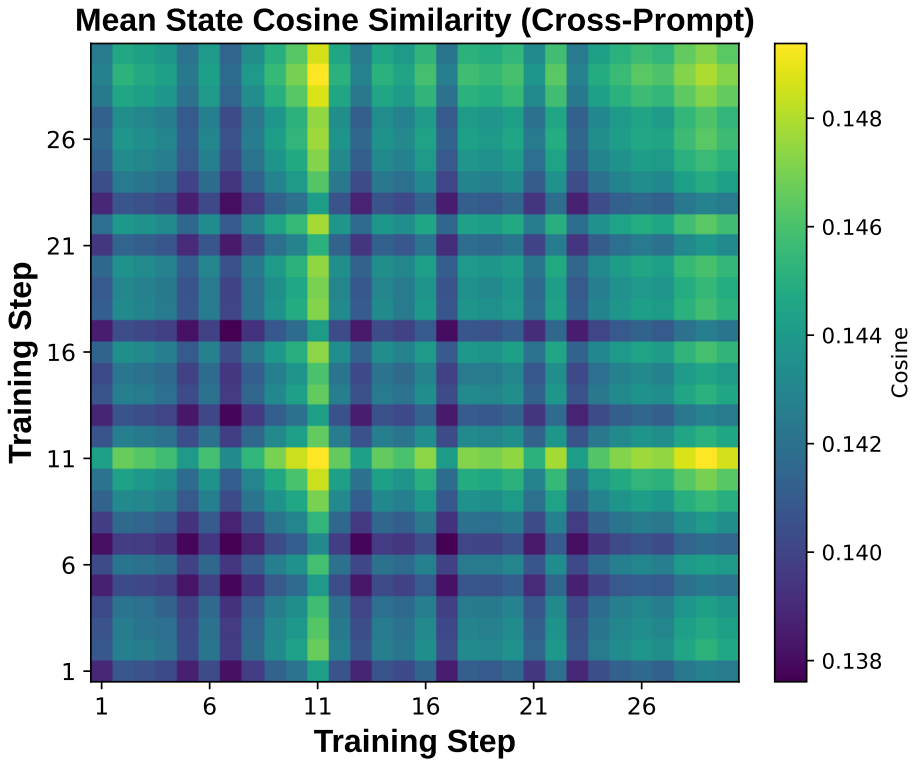


Figure 6: **The geometry of visited states remains stable across training.** Mean cross-prompt cosine similarity between states collected at training steps i and j. All off-diagonal entries lie in the narrow range $[0.138, 0.149]$, with no clear separation between early and late training steps.

Setup. We instrument OPD training to record the states visited by on-policy rollouts. We use a Qwen3-4B student distilled from Qwen3-4B-Non-Thinking-RL-Math on DeepMath. A state is the full autoregressive context: the prompt x and the student-generated prefix $y_{<t}$, denoted $s_t = (x, y_{<t})$. Over 30 training steps we collect 61,440 states from 7,680 distinct prompts, with 256 rollouts per step and 8 uniformly spaced positions per rollout. Each state is unique at the token level. To compare states across training, we map each state into a fixed representation space using the frozen teacher: $h_T(s)$ is

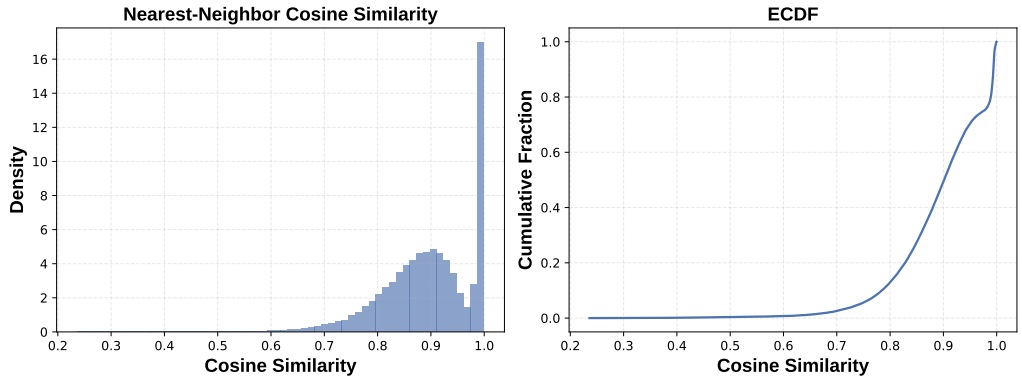


Figure 7: **Strong cross-prompt nearest-neighbor redundancy.** Distribution of nearest-neighbor cosine similarity over all $61{,}440$ states, where the neighbor is required to originate from a different prompt. The mean nearest-neighbor similarity is 0.893; 29.5% of states have a neighbor with cosine similarity at least 0.95, and 18.5% have one at least 0.99.

Table 6: Cross-prompt nearest-neighbor similarity over all $61{,}440$ states. The last two columns report the fraction of states whose nearest *different-prompt* neighbor exceeds the corresponding threshold.

	mean	median	p90	p95	$\% \geq 0.99$	$\% \geq 0.95$
Cross-prompt NN	0.893	0.901	0.994	0.995	18.5%	29.5%

the teacher's final-layer hidden state at the last token of s. All similarities are cosine similarities in this space. Since the teacher is fixed, representations from different steps and prompts are directly comparable.

Strong cross-prompt redundancy. For each state, we find its nearest neighbor among states from a different prompt (Figure 7). Even with this restriction, the nearest-neighbor similarity is high. The mean is 0.893 and the median is 0.901. The upper tail is concentrated: the 90th and 95th percentiles reach 0.994 and 0.995. Overall, 29.5% of states have a different-prompt neighbor with cosine similarity at least 0.95, and 18.5% have one at least 0.99 (Table 6). Restricting to different prompts leaves the mean nearest-neighbor similarity almost unchanged: 0.893 versus 0.894 without this restriction. Thus, this redundancy is not just from repeated prefixes of the same prompt. States from different training problems often lie in nearby regions of the teacher representation space.

The visited-state geometry remains stable across training. We next ask whether later training steps occupy systematically different regions of the representation space. Figure 6 shows the mean cross-prompt cosine similarity between states from every pair of training steps. The matrix is nearly flat: all off-diagonal entries fall within a narrow range $[0.138, 0.149]$. In particular, the similarity between step 1 and step 30 is comparable to that between nearby steps. A nearest-neighbor analysis agrees: the fraction of states whose nearest neighbor comes from the same training step is about the chance level of $1/30$. Later states do not tend to cluster with other late-training states rather than with much earlier states. These results do not mean the exact state distribution is unchanged. Rather, under this fixed representation, training does not produce a clear progressive separation of visited states into new regions.

A small early state library already covers a substantial fraction of later states. Figure 5 quantifies this redundancy. For each state at a given step, we search only among states from earlier steps and different prompts. The 2,048 states from step 1 already give a neighbor with cosine similarity at least 0.99 for 10.4% of states in the next 29 steps, and at least 0.95 for 24.9%. Expanding the library by $29\times$, from 2,048 to 59,392 states, adds only modest coverage: the fraction with a $\cos \geq 0.99$ neighbor rises from 10.4% to 18.4%, and the mean similarity to the closest earlier state

goes from 0.823 to 0.891. So even though the library grows a lot, the representational novelty from more training steps grows much more slowly.

Taken together, these results show substantial redundancy in the states visited during OPD. Training keeps generating token-level distinct contexts, but many of them are close to states already seen under other prompts. This repeated coverage helps explain why OPD can stay effective even when each update keeps only coarse directional information instead of the precise magnitude of the teacher-student discrepancy. Because the same or nearby states are visited repeatedly, the student is corrected many times in the same direction, and BinaryOPD can work despite discarding all magnitude information.

B Whether the OPD Update Direction Is Aligned with the Verifier Is Not Important

Recently, combining OPD with RLVR has become a popular direction. OPD provides dense teacher guidance but does not consider trajectory correctness, while RLVR (Guo et al., 2025; Shao et al., 2024; Jaech et al., 2024; Trinh et al., 2024; Yang et al., 2024) optimizes directly toward trajectory-level correctness but suffers from sparse rewards. Combining them therefore appears to be an effective and promising direction (Wang et al., 2026; Cai et al., 2026; Hou et al., 2026; Yang et al., 2026a; Akhondzadeh et al., 2026; Lin et al., 2026; Ding et al., 2026). As a side investigation, we explore whether the verifier signal is important for OPD. Our setup builds directly on BinaryOPD. In the standard RLVR setting, trajectories with correct final answers are positively reinforced, while trajectories with incorrect final answers are negatively reinforced. Here, we completely reverse this relationship. Specifically:

- On trajectories with **correct** final answers, we keep only tokens whose teacher probability is *lower* than the student probability, and assign them a reward of -1.
- On trajectories with **incorrect** final answers, we keep only tokens whose teacher probability is *higher* than the student probability, and assign them a reward of $+1$.

As a result, tokens in correct trajectories receive non-positive rewards, while tokens in incorrect trajectories receive non-negative rewards. This is the exact opposite of what a verifier would normally encourage.

Formally, for a rollout o with final-answer correctness $c \in \{0, 1\}$, the reward at token o_t is

$$r_t = \begin{cases} -1, & \text{if } c = 1 \text{ and } \pi_T(o_t \mid q, o_{<t}) < \pi_\theta(o_t \mid q, o_{<t}), \\ +1, & \text{if } c = 0 \text{ and } \pi_T(o_t \mid q, o_{<t}) > \pi_\theta(o_t \mid q, o_{<t}), \\ 0, & \text{otherwise.} \end{cases}$$

We denote the reversed-mask variant as **Verifier-Conflicted**. For comparison, we also evaluate **Verifier-Aligned**, which keeps only tokens whose update direction aligns with the verifier signal.

If the verifier signal were beneficial for OPD, this reversed masking should severely degrade performance or completely change the training dynamics, and Verifier-Aligned should yield improvements over Verifier-Conflicted and OPD.

Tables 7 and 8 show that reversing the verifier signal does not hurt BinaryOPD. Across all student-teacher pairs, Verifier Conflicted performs on par with or even slightly better than OPD and Binary-OPD. For example, on the DeepSeek-Distill-Qwen-1.5B pair, Verifier Conflicted achieves the best average score of 55.1, while Verifier Aligned gets 54.4. On the Qwen3-4B-Non-Thinking math pair, all methods are close, with Verifier Conflicted and Verifier Aligned both reaching around 64. On code, Verifier Conflicted even achieves the highest LiveCodeBench v6 score (29.1). In contrast, Verifier Aligned shows no consistent improvement over OPD or BinaryOPD. As further shown in Figure 8 (Appendix D), all methods exhibit nearly identical training dynamics, confirming that the verifier signal has little influence on the training pattern. Even if the BinaryOPD signal completely contradicts the verifier, the performance remains unaffected. These results clearly indicate that whether the update direction aligns with the verifier signal has little effect on BinaryOPD performance, questioning the necessity of incorporating the verifier signal into OPD.

Table 7: Math benchmark results. All results are averaged over 8 samples (Avg@8).

Method	AIME24	AIME25	AMC	MATH500	Minerva	Olympiad	Avg
DeepSeek-Distill-Qwen-1.5B (Student) vs. JustRL-1.5B (Teacher)							
Student	23.7	17.0	59.7	80.9	23.7	41.4	41.0
Teacher	46.2	37.5	82.0	89.2	30.7	53.5	56.5
OPD	43.8	32.5	78.8	88.5	31.7	52.6	54.7
BinaryOPD	**45.4**	32.1	78.5	**88.9**	31.6	52.6	54.9
Verifier-Aligned	44.6	31.3	78.0	88.7	31.3	52.6	54.4
Verifier-Conflicted	45.0	**32.9**	**79.2**	88.8	**31.8**	**53.0**	**55.1**
Qwen3-1.7B-Base (Student) vs. Qwen3-4B-Base-RL (Teacher)							
Student	4.1	1.7	23.2	48.9	8.9	17.1	17.3
Teacher	10.6	13.1	40.3	74.2	17.2	30.0	30.9
OPD	7.5	3.3	26.4	57.8	11.4	20.6	21.2
BinaryOPD	**8.1**	**4.0**	30.3	**59.1**	10.0	21.6	**22.2**
Verifier-Aligned	5.0	2.9	29.5	58.5	11.0	21.4	21.4
Verifier-Conflicted	6.5	2.3	**31.2**	57.3	11.3	**23.4**	22.0
Llama-3.2-3B-Instruct (Student) vs. GT-Llama3.2-3B-MATH (Teacher)							
Student	2.9	0.0	14.8	34.5	7.7	7.9	11.3
Teacher	11.7	0.0	20.9	49.1	15.7	15.4	18.8
OPD	**12.5**	0.4	**22.7**	46.0	13.3	14.3	**18.2**
BinaryOPD	9.2	0.0	21.7	47.2	14.8	14.6	17.9
Verifier-Aligned	8.3	**0.8**	20.8	46.3	**15.3**	14.2	17.6
Verifier-Conflicted	9.2	0.0	21.7	**47.4**	15.2	**14.8**	18.1
Qwen3-4B-Non-Thinking (Student) vs. Qwen3-4B-Non-Thinking-RL-Math (Teacher)							
Student	25.0	16.7	58.4	81.3	27.7	42.9	42.0
Teacher	61.2	56.3	88.0	93.2	36.7	60.5	66.0
OPD	60.4	51.7	85.7	93.0	34.5	59.2	64.1
BinaryOPD	**61.3**	**55.0**	**85.7**	92.8	**35.9**	**59.8**	**65.1**
Verifier-Aligned	**61.3**	**55.0**	84.6	92.7	34.6	59.3	64.6
Verifier-Conflicted	**61.3**	51.7	84.9	**93.2**	34.4	**59.8**	64.2

Table 8: Code benchmark results. All results are averaged over 8 samples (Avg@8).

Method	LiveCodeBench v6	HumanEval	MBPP	Avg
Qwen3-4B-Non-Thinking (Student) vs. Qwen3-4B-Non-Thinking-RL-Code (Teacher)				
Student	25.1	80.8	48.7	51.5
Teacher	36.0	90.6	56.2	60.9
OPD	27.9	85.4	53.0	**55.4**
BinaryOPD	27.8	**85.7**	52.1	55.2
Verifier-Aligned	28.1	84.6	**52.2**	55.0
Verifier-Conflicted	**29.1**	84.1	51.9	55.0

C Experimental Details

Models and Datasets. We use JustRL-1.5B following (He et al., 2025), Qwen3-4B-Non-Thinking-RL-Math and Qwen3-4B-Non-Thinking-RL-Code following (Yang et al., 2026b), GT-Llama3.2-3B-MATH following (Zhang et al., 2025), Nemotron-Research-Reasoning-1.5B following (Liu et al., 2026) and Qwen3-4B-Base-RL. We obtain Qwen3-4B-Base-RL by training Qwen3-4B-Base with GRPO on DAPO-MATH-17K for 3 epochs. All these teachers are domain-specialized models obtained by running reinforcement learning on their respective domains. The DeepMath dataset follows (Yang et al., 2026b), which selects 57K samples with a difficulty level greater than or equal to 6.

Hyperparameters. We use the **Verl** framework for training (Sheng et al., 2025). We use the hyperparameters listed in Table 9 for all experiments.

Table 9: Hyperparameter settings.

Hyperparameter	**Value**
Learning Rate	1e-6
Train Batch Size	256
Max Response Length (Training)	8192
Max Response Length (Evaluation)	16384
Max Prompt Length	1024
Rollout Temperature	1.0
Evaluation Temperature	0.7
Evaluation Top-p	0.95

D Training Dynamics

Figure 8 shows the full training dynamics of all methods (OPD, BinaryOPD, Verifier Aligned, Verifier Conflicted) in the experiments of Section 4 and Appendix B. Across all methods, the curves are nearly identical in training accuracy, entropy, response length, and gradient norm, confirming that removing the KL magnitude or reversing the verifier signal does not change the training pattern.

Figures 9 and 10 provide the training dynamics for the positive reward experiment and the negative reward experiment in Section 5, respectively. In both experiments, successful settings exhibit normal training curves, while settings that incorrectly reward the high-disagreement group fail to converge or degrade clearly.

E Visualization of High-Disagreement Tokens

Figure 11 visualizes an example of high-disagreement tokens. The tokens highlighted in blue have $\log(p_T/p_s) > 0.8$, and those highlighted in red have $\log(p_T/p_s) < -0.8$. This example is from the JustRL-1.5B teacher and the DeepSeek-Distill-Qwen-1.5B student.

F Hardware Setup

All experiments are conducted on NVIDIA RTX 5090, A100, A800, H20, and H100 GPUs.

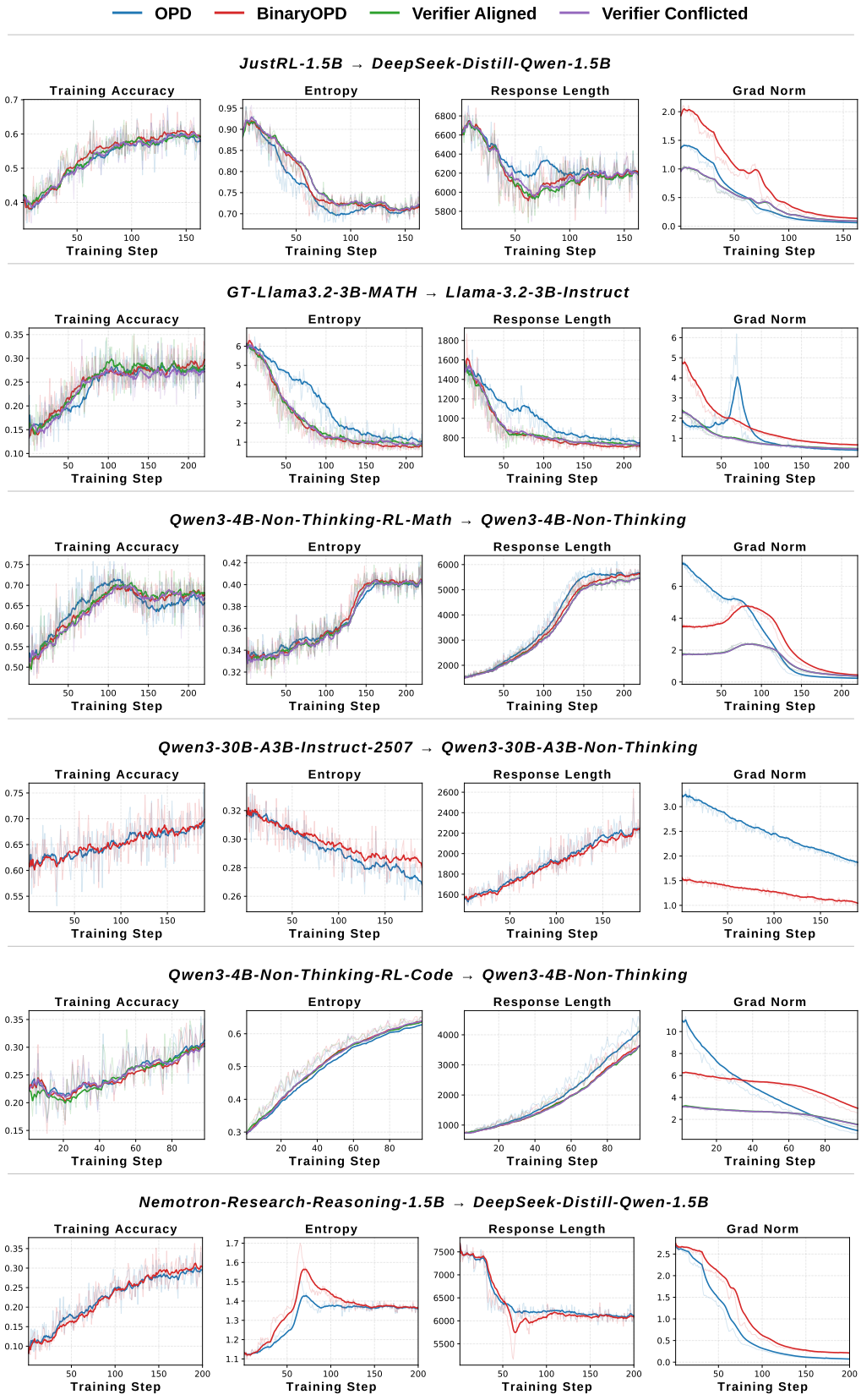


Figure 8: Full training dynamics of all methods (OPD, BinaryOPD, Verifier-Aligned, Verifier-Conflicted) across all student-teacher pairs and datasets. From left to right: training accuracy, entropy, response length, and gradient norm.

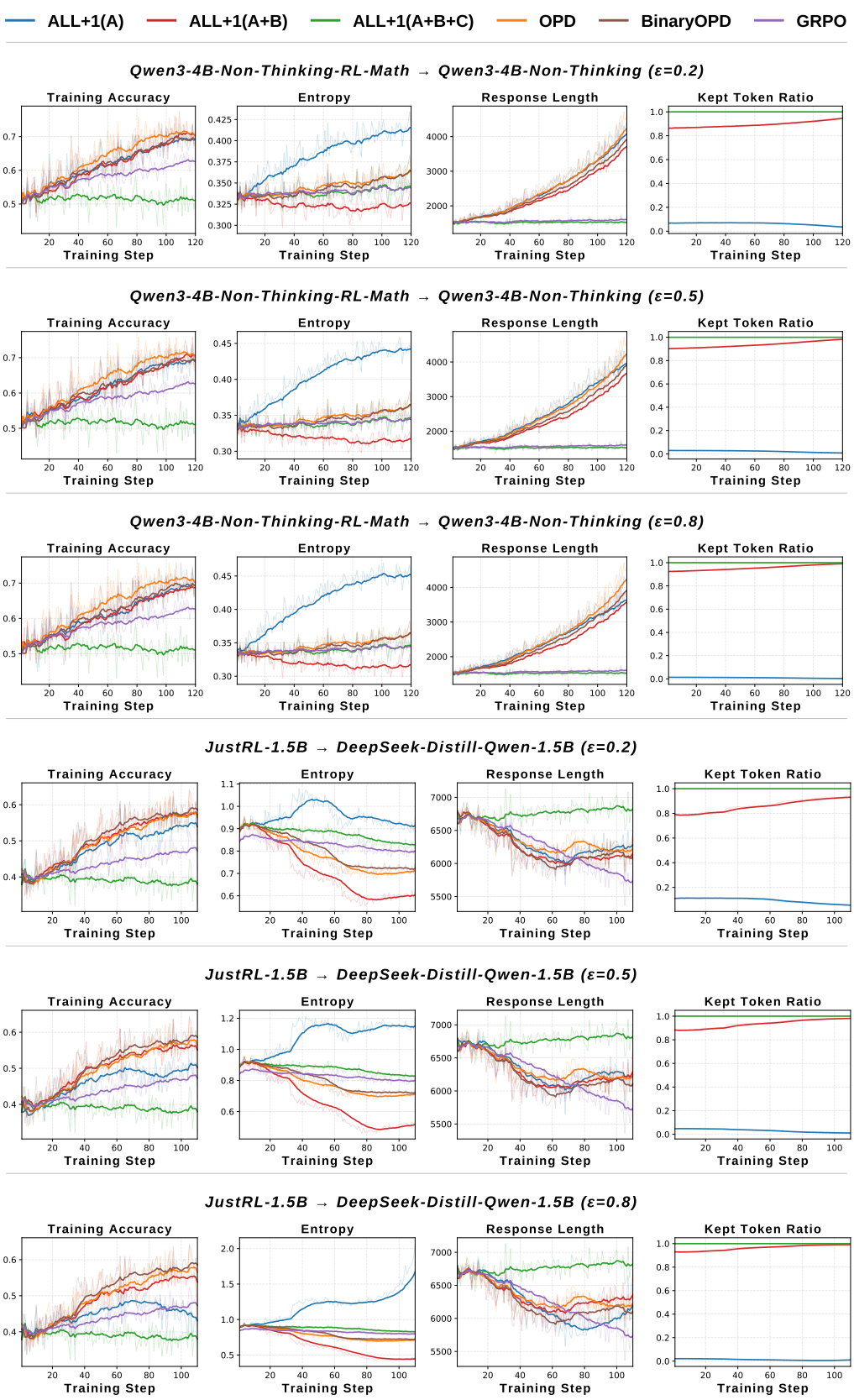


Figure 9: Full training dynamics for the positive reward experiment (ALL+1) across different ϵ values and student-teacher pairs.

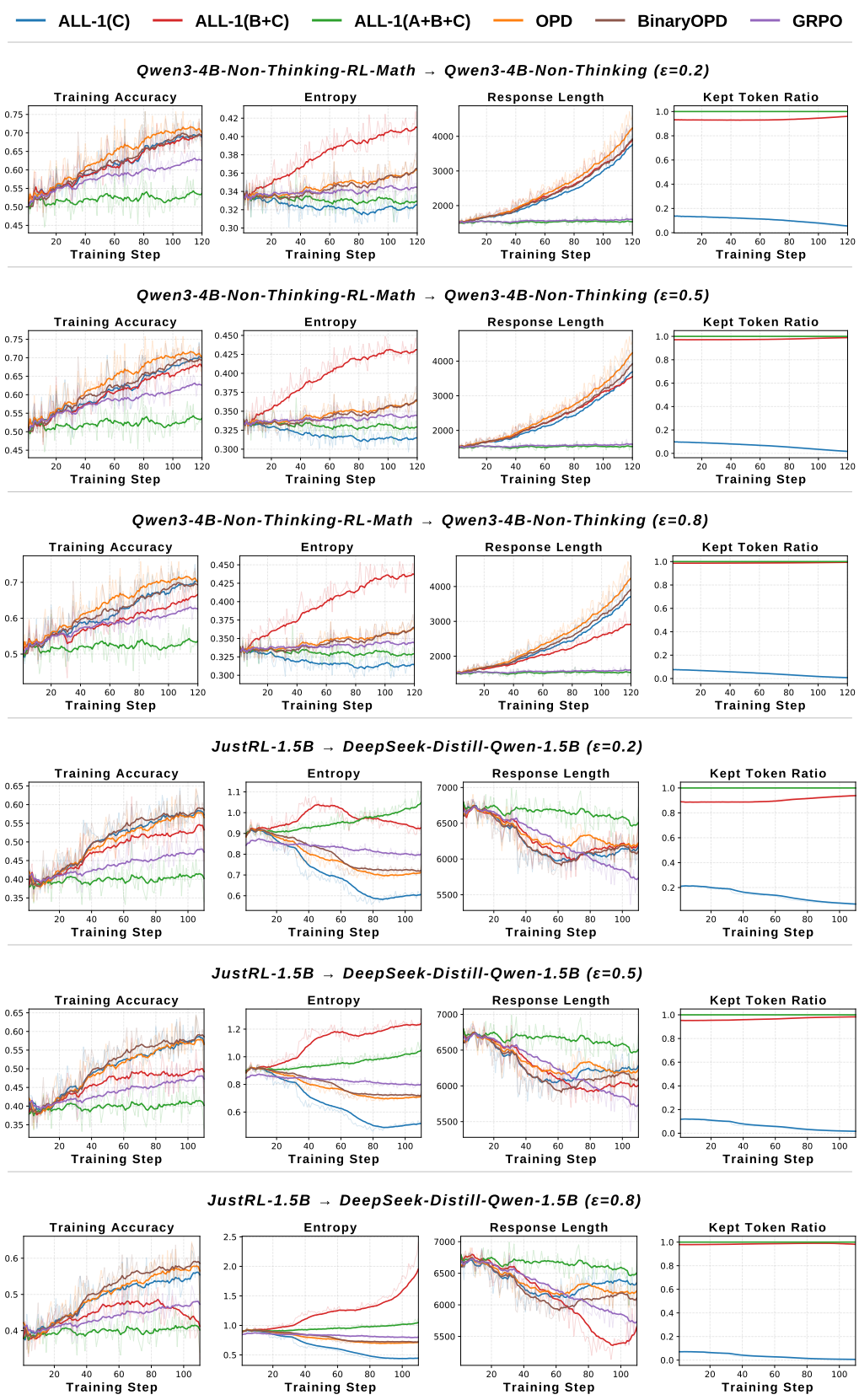
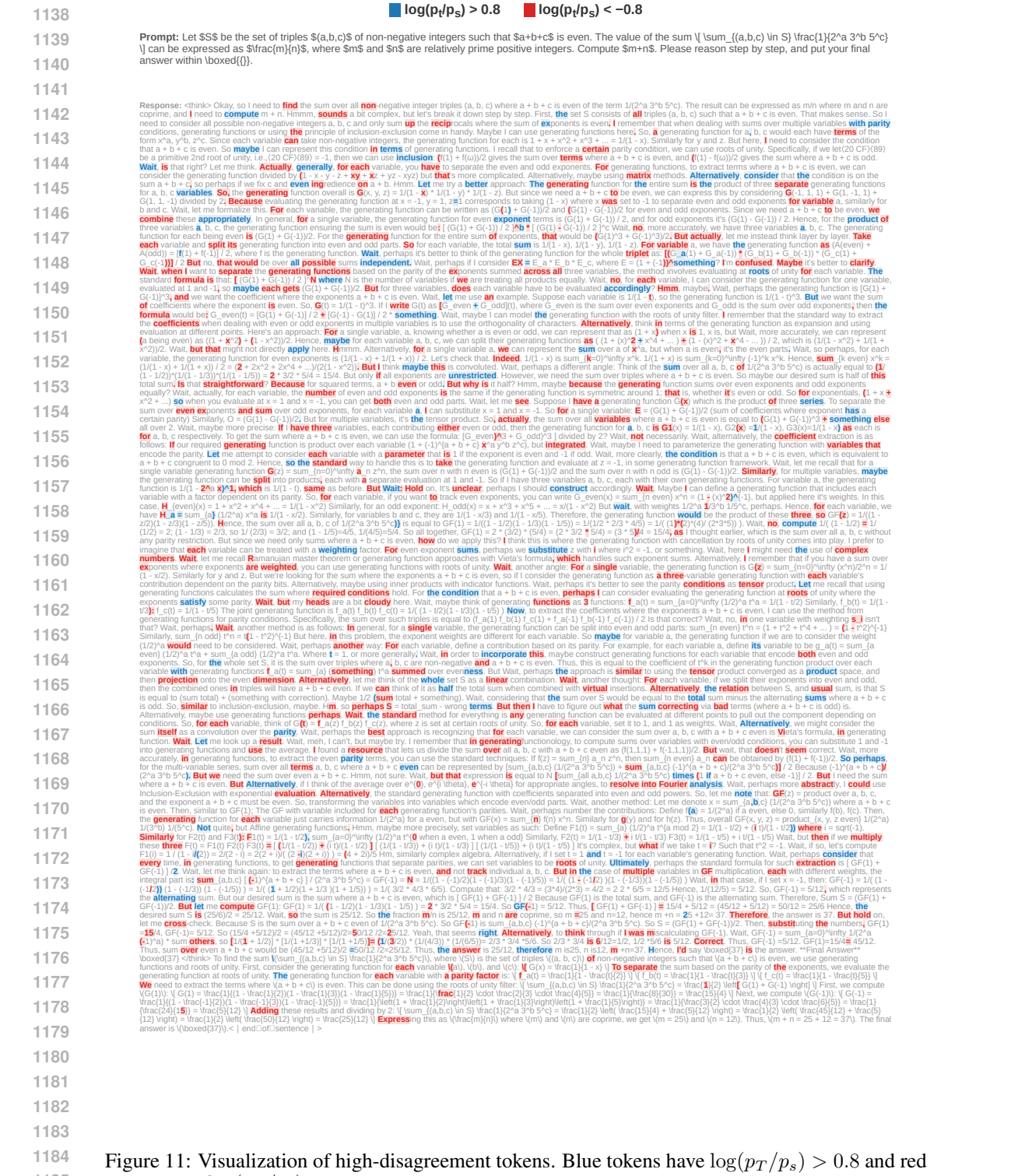


Figure 10: Full training dynamics for the negative reward experiment (ALL-1) across different ϵ values and student-teacher pairs.


log(p_t/p_s) > 0.8
log(p_t/p_s) < −0.8
Prompt: Let S be the set of triples (a,b,c) of non-negative integers such that $a+b+c$ is even. The value of the sum \[\sum_{(a,b,c) \in S} \frac{1}{2^a 3^b 5^c} \] can be expressed as $\frac{m}{n}$, where m and n are relatively prime positive integers. Compute $m+n$. Please reason step by step, and put your final answer within \boxed{{}}.
Response: <think> Okay, so I need to find the sum over all non-negative integer triples (a, b, c) where a + b + c is even of the term 1/(2^a 3^b 5^c). The result can be expressed as m/n where m and n are coprime, and I need to compute m + n. Hmmm, sounds a bit complex, but let's break it down step by step. First, the set S consists of all triples (a, b, c) such that a + b + c is even. That makes sense. So I need to consider all possible non-negative integers a, b, c and only sum up the reciprocals where the sum of exponents is even. I remember that when dealing with sums over multiple variables with parity conditions, generating functions or using the principle of inclusion-exclusion come in handy. Maybe I can use generating functions here. So, a generating function for a, b, c would each have terms of the form x^a, y^b, z^c. Since each variable can take non-negative integers, the generating function for each is 1 + x + x^2 + x^3 + ... = 1/(1 - x). Similarly for y and z. But here, I need to consider the condition that a + b + c is even. So maybe I can represent this condition in terms of generating functions. I recall that to enforce a certain parity condition, we can use roots of unity. Specifically, if we let(20 CF)(89) be a primitive 2nd root of unity, i.e.,(20 CF)(89) = -1, then we can use inclusion. (f(1) + f(ω))/2 gives the sum over terms where a + b + c is even, and (f(1) - f(ω))/2 gives the sum where a + b + c is odd. Wait, is that right? Let me think. Actually, generally, for each variable, you have to separate the even and odd exponents. For generating functions, to extract terms where a + b + c is even, we can consider the generating function divided by (1 - x - y - z + xy + xz + yz - xyz) but that's more complicated. Alternatively, maybe using matrix methods. Alternatively, consider that the condition is on the sum a + b + c, so perhaps if we fix c and even ing redience on a + b. Hmm. Let me try a better approach: The generating function for the entire sum is the product of three separate generating functions for a, b, c variables. So, the generating function overall is G(x, y, z) = 1/(1 - x) * 1/(1 - y) * 1/(1 - z). But since we need a + b + c to be even, we can express this by considering G(-1, 1, 1) + G(1, -1, 1) + G(1, 1, -1) divided by 2. Because evaluating the generating function at x = -1, y = 1, z=1 corresponds to taking (1 - x) where x was set to -1 to separate even and odd exponents for variable a, similarly for b and c. Wait, let me formalize this. For each variable, the generating function can be written as (G(1) + G(-1))/2 and (G(1) - G(-1))/2 for even and odd exponents. Since we need a + b + c to be even, we combine these appropriately. In general, for a single variable, the generating function for even exponent terms is (G(1) + G(-1)) / 2, and for odd exponents it's (G(1) - G(-1)) / 2. Hence, for the product of three variables a, b, c, the generating function ensuring the sum is even would be: [(G(1) + G(-1)) / 2]^b * [(G(1) + G(-1)) / 2]^c Wait, no, more accurately, we have three variables a, b, c. The generating function for each being even is (G(1) + G(-1))/2. For the generating function for the entire sum of exponents, that would be (G(1)^3 + G(-1)^3)/2. But actually, let me instead think layer by layer. Take each variable and split its generating function into even and odd parts. So for each variable, the total sum is 1/(1 - x), 1/(1 - y), 1/(1 - z). For variable a, we have the generating function as (A(even) + A(odd)) = [f(1) + f(-1)] / 2, where f is the generating function. Wait, perhaps it's better to think of the generating function for the whole triplet as: [(G_a(1) + G_a(-1)) * (G_b(1) + G_b(-1)) * (G_c(1) + G_c(-1))] / 2 But no, that would be over all possible sums independent. Wait, perhaps if I consider EX = E_a * E_b * E_c, where E = (1 + (-1)^something? I'm confused. Maybe it's better to clarify. Wait, when I want to separate the generating functions based on the parity of the exponents summed across all three variables, the method involves evaluating at roots of unity for each variable. The standard formula is that: [(G(1) + G(-1)) / 2]^N where N is the number of variables if we are treating all products equally. Wait, no, for each variable, I can consider the generating function for one variable, evaluated at 1 and -1, so maybe each gets (G(1) + G(-1))/2. But for three variables, does each variable have to be evaluated accordingly? Hmm, maybe. Wait, perhaps the generating function is [G(1) + G(-1)]^3, and we want the coefficient where the exponents a + b + c is even. Wait, let me use an example. Suppose each variable is 1/(1 - t), so the generating function is 1/(1 - t)^3. But we want the sum of coefficients where the exponent is even. So, G(t) = 1/(1 - t)^3. If I write G(t) as [G_even + G_odd](t), where G_even is the sum over even exponents and G_odd is the sum over odd exponents, then the formula would be: G_even(t) = [G(1) + G(-1)] / 2 + [G(1) - G(1)] / 2 * something. Wait, maybe I can model the generating function with the roots of unity filter. I remember that the standard way to extract the coefficients when dealing with even or odd exponents in multiple variables is to use the orthogonality of characters. Alternatively, think in terms of the generating function as expansion and using evaluation at different points. Here's an approach: For a single variable, a, knowing whether a is even or odd, we can represent that as (1 + x) when x is 1, x is, but Wait, more accurately, we can represent (a being even) as ((1 + x^2) + (1 - x^2))/2. Hence, maybe for each variable a, b, c, we can split their generating functions as ((1 + (x)^2 + x^4 + ...) + (1 - (x)^2 + x^4 - ...)) / 2, which is (1/(1 - x^2) + 1/(1 + x^2))/2. Wait, but that might not directly apply here. Hmmm. Alternatively, for a single variable a, we can represent the sum over a of x^a, but when a is even, it's the even parts. Wait, so perhaps, for each variable, the generating function for even exponents is (1/(1 - x) + 1/(1 + x)) / 2. Let's check that. Indeed, 1/(1 - x) is sum_{k=0}^\infty x^k. 1/(1 + x) is sum_{k=0}^\infty (-1)^k x^k. Hence, sum_{k even} x^k = (1/(1 - x) + 1/(1 + x)) / 2 = (2 + 2x^2 + 2x^4 + ...)/(2(1 - x^2)). But I think maybe this is convoluted. Wait, perhaps a different angle: Think of the sum over all a, b, c of 1/(2^a 3^b 5^c) is actually equal to (1/(1 - 1/2))*(1/(1 - 1/3))*(1/(1 - 1/5)) = 2 * 3/2 * 5/4 = 15/4. But only if all exponents are unrestricted. However, we need the sum over triples where a + b + c is even. So maybe our desired sum is half of this total sum. Is that straightforward? Because for squared terms, a + b even or odd. But why is it half? Hmm, maybe because the generating function sums over even exponents and odd exponents equally? Wait, actually, for each variable, the number of even and odd exponents is the same if the generating function is symmetric around 1; that is, whether it's even or odd. So for exponentials, (1 + x + x^2 + ...) so when you evaluate at x = 1 and x = -1, you can get both even and odd parts. Wait, let me see. Suppose I have a generating function G(x) which is the product of three series. To separate the sum over even exponents and sum over odd exponents, for each variable a, I can substitute x = 1 and x = -1. So for a single variable: E = (G(1) + G(-1))/2 (sum of coefficients where exponent has a certain parity) Similarly, O = (G(1) - G(-1))/2. But for multiple variables, it's the tensor product. So, actually, the sum over all variables where a + b + c is even is equal to (G(1) + G(-1))^3 + something else all over 2. Wait, maybe more precise. If I have three variables, each contributing either even or odd, then the generating function for a, b, c is G1(x) = 1/(1 - x), G2(x) = 1/(1 - x), G3(x)=1/(1 - x) as each is for a, b, c respectively. To get the sum where a + b + c is even, we can use the formula: [G_even)^3 + G_odd)^3] divided by 2? Wait, not necessarily. Wait, alternatively, the coefficient extraction is as follows: If our required generating function is product over each variable (1 + (-1)^{a + b + c} x^a y^b z^c), but integrated. Wait, maybe I need to parameterize the generating function with variables that encode the parity. Let me attempt to consider each variable with a parameter that is 1 if the exponent is even and -1 if odd. Wait, more clearly, the condition is that a + b + c is even, which is equivalent to a + b + c congruent to 0 mod 2. Hence, so the standard way to handle this is to take the generating function and evaluate at z = -1, in some generating function framework. Wait, let me recall that for a single variable generating function G(z) = sum_{n=0}^\infty a_n z^n, the sum over n with n even is (G(1) + G(-1))/2 and the sum over n with n odd is (G(1) - G(-1))/2. Similarly, for multiple variables, maybe the generating function can be split into products, each with a separate evaluation at 1 and -1. So if I have three variables a, b, c, each with their own generating functions. For variable a, the generating function is 1/(1 - 2^a x)^1, which is 1/(1 - t), same as before. But Wait; Hold on, it's unclear; perhaps I should construct accordingly. Wait, Maybe I can define a generating function that includes each variable with a factor dependent on its parity. So, for each variable, if you want to track even exponents, you can write G_even(x) = sum_{n even} x^n = (1 + (x)^2)^{-1}, but applied here it's weights. In this case, H_{even}(x) = 1 + x^2 + x^4 + ... = 1/(1 - x^2) Similarly, for an odd exponent: H_odd(x) = x + x^3 + x^5 + ... = x/(1 - x^2) But wait, with weights 1/2^a 1/3^b 1/5^c, perhaps. Hence, for each variable, we have H_a = sum_{a} (1/2^a) x^a is 1/(1 - x/2). Similarly, for variables b and c, they are 1/(1 - x/3) and 1/(1 - x/5). Therefore, the generating function would be the product of these three. so GF(z) = 1/((1 - z/2)(1 - z/3)(1 - z/5)). Hence, the sum over a, b, c,