MOPD: MULTI-ROLLOUT ON-POLICY DISTILLATION VIA PEER SUCCESSES AND FAILURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are often post-trained with sparse verifier rewards, which indicate whether a sampled trajectory succeeds but provide limited guidance about where reasoning succeeds or fails. On-policy distillation (OPD) offers denser token-level supervision by training on student-generated trajectories, yet existing methods typically distill each rollout independently and ignore the other attempts sampled for the same prompt. We introduce Multi-Rollout On-Policy Distillation (MOPD), a peer-conditioned distillation framework that uses the student's local rollout group to construct more informative teacher signals. MOPD conditions the teacher on both successful and failed peer rollouts: successes provide positive evidence for valid reasoning patterns, while failures provide structured negative evidence about plausible mistakes to avoid. Experiments on coding, mathematical reasoning, scientific question answering, and tool-use benchmarks show that MOPD improves over standard on-policy baselines on most benchmarks, and also improves over baselines that reuse the same rollout group through verified-success SFT or preference optimization. Under matched demonstration budgets, random peers, peers from unrelated prompts, and success-only contexts all fall short of the mixed context, indicating that the gain comes from the same-prompt success–failure contrast rather than from a longer teacher context. Teacher-signal analysis shows the same ordering: mixed contexts align teacher scores with verifier rewards better than any single-type context. These results indicate that effective on-policy distillation should exploit the student's multi-rollout trial-and-error behavior rather than treating rollouts as isolated samples. Code is available at `https://anonymous.4open.science/r/mopd_code-C7BC/`.

1 INTRODUCTION

Large language models (LLMs) are often post-trained with reinforcement learning from verifiable rewards for math, coding, etc, where sampled solutions are scored by answer checkers, unit tests, or task-specific verifiers (Guo et al., 2025; Schulman et al., 2017; Shao et al., 2024). Although effective, these rewards are usually sparse: they indicate whether an entire trajectory succeeds or fails, but provide little guidance about which tokens or reasoning steps caused the outcome (Yue et al., 2025; Chan et al., 2024). Distillation and self-distillation provide a complementary form of supervision by converting teacher or privileged-model judgments into dense token-level signals (Zelikman et al., 2022; Singh et al., 2024; Chen et al., 2024; Yang et al., 2024). On-policy distillation further aligns this supervision with the student's own behavior by training on trajectories sampled from the student policy (Agarwal et al., 2024; Gu et al., 2024; Ko et al., 2024).

Despite this advantage, existing OPD and on-policy self-distillation methods underuse a key source of information already available during training: multiple rollouts generated for the same problem instance. In many pipelines, each rollout is distilled independently, so the teacher signal for one trajectory is computed without access to the other attempts sampled from the same student. This design discards the local structure of the rollout group. For reasoning-intensive tasks, such local structure is highly informative: successful rollouts reveal valid solution strategies, while failed rollouts expose plausible but incorrect reasoning paths, missing constraints, formatting errors, or execution mistakes. Treating rollouts independently prevents the teacher from comparing these alternatives and identifying where an unsuccessful trajectory diverges from a successful one.

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.

We propose *Multi-Rollout On-Policy Distillation (MOPD)*, a peer-conditioned distillation framework that explicitly uses the student's rollout group to construct teacher signals. For each prompt, the student samples multiple on-policy trajectories, which are scored by a verifier and partitioned into successful and failed sets. MOPD then conditions the teacher on peer rollouts from the same problem instance when supervising a student's trajectory. The key insight is that **multi-rollout sampling creates a local trial-and-error space around each problem**. Instead of viewing each trajectory in isolation, the teacher can compare the current rollout against peer successes and failures, using this contrast to produce more targeted supervision. This turns OPD from independent imitation of individual trajectories into a comparative learning process: the teacher is no longer only a global expert, but also a local diagnostic model that can recognize instance-specific errors and tell apart failures that merely look plausible from solutions that are actually correct.

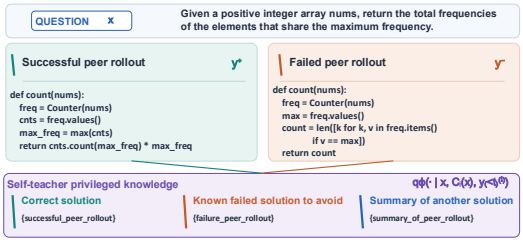


Figure 1: MOPD Illustration.

Prior work also assumes that a self-teacher becomes reliable once conditioned on privileged information, but tests that assumption only indirectly, through downstream accuracy. Checking whether the privileged teacher can recover the correct answer is no better a diagnostic: with answer-related information already in its context, the teacher can look accurate by exploiting shortcuts rather than by producing a faithful signal over the student's own trajectories. **What matters is not whether the self-teacher knows the answer, but whether its preferences are aligned with correctness among the rollouts the student actually generates.** We therefore introduce an analysis of self-teacher signal quality: for each prompt we hold the student-generated rollouts and their verifier rewards fixed, vary only the context shown to the self-teacher, and measure how well the teacher's scores agree with those rewards.

Empirically, MOPD improves over standard on-policy baselines across coding, mathematical reasoning, scientific question answering and tool use, over baselines that reuse the same rollout group through verified-success SFT or preference optimization, and across model families (including Llama-3.2-3B-Instruct, where the base policy is far weaker). The strongest gains come from mixed peer contexts combining successes and failures, and the self-teacher signal analysis shows the same two-success-plus-one-failure context aligns best with verifier rewards. Effective distillation should exploit the student's multi-rollout trial-and-error behavior, not only the strength of the teacher.

2 Related Work

On-Policy Distillation Knowledge distillation transfers a teacher's predictive distribution to a smaller student, but off-policy training creates a distribution mismatch: the student is supervised on states it never visits at inference (Hinton et al., 2015; Kim & Rush, 2016; Ross et al., 2011). On-policy distillation addresses this by sampling trajectories from the student's own policy and computing teacher supervision on those same trajectories (Agarwal et al., 2024; Gu et al., 2024; Ko et al., 2024). Within this framework, work has focused on divergence objectives, token reweighting, and interpolation between on- and off-policy regimes (Wen et al., 2023; Huang et al., 2025; Zhang et al., 2026a; Jin et al., 2026). A parallel thread replaces an external teacher with self-distillation: one copy of the model is conditioned on privileged context—solutions, hints, or environment feedback—and supervises a second copy that lacks it (Hübotter et al., 2026; Ye et al., 2026; Zhao et al., 2026b; Penaloza et al., 2026; Yang et al., 2026). Information can also be aggregated across multiple sampled solutions through consistency, voting, or committee teachers (Wang et al., 2023; Muennighoff et al., 2025; Li et al., 2025b). However, the teacher signal for each rollout is constructed independently, leaving cross-rollout structure within a sampling group unexploited.

Multi-Rollout Reinforcement Learning and Verifier-Guided Improvement Post-training with verifiable rewards replaces learned reward models with programmatic checkers and updates the policy from group-relative advantages computed over multiple sampled rollouts per prompt (Shao et al., 2024; Wen et al., 2026; Ouyang et al., 2022; Rafailov et al., 2023). Process reward models provide

denser credit assignment by grading intermediate reasoning steps (Cobbe et al., 2021; Setlur et al., 2025; Zhang et al., 2025c), and hybrid pipelines couple these signals with on-policy distillation (Xu et al., 2025b;a; Zhang et al., 2026b). While these methods exploit multiple rollouts per prompt through advantage normalization or rejection sampling, the supervisory signal for any single trajectory is computed without conditioning on the contents of the others. The work most adjacent to ours treats successes and failures as complementary evidence at the level of advantages or filters, but stops short of routing both into a single peer-conditioned distillation target.

Extended related work and the positioning of the current method is in Section A.

3 PRELIMINARIES

We consider the problem of distilling reasoning capabilities from a teacher model into a student model under an on-policy sampling regime. Let $x \in \mathcal{X}$ denote an input problem or prompt, and let $y = (y_1, \ldots, y_T)$ denote an output trajectory generated autoregressively by a student policy π_θ. At decoding step t, we write the prefix as $y_{<t}$ and the token-level context as $c_t = (x, y_{<t})$. The student distribution is $\pi_\theta(\cdot \mid c_t)$, while the teacher distribution is denoted by $q_\phi(\cdot \mid c_t)$, possibly conditioned on additional privileged information. In prior self-distillation methods, the privileged information available typically includes ground-truth outcomes, verified successful answers (Ye et al., 2025; 2026; Zhao et al., 2026b; Yang et al., 2026), or environment feedback (Hübotter et al., 2026).

On-policy distillation. Offline distillation trains the student on trajectories sampled from a teacher or from a fixed dataset. This creates a distribution mismatch: during inference, the student may visit states that were not observed during training. On-policy distillation addresses this issue by first sampling trajectories from the student itself and then obtaining teacher supervision on those same trajectories. Formally, for each prompt x, the student generates $y \sim \pi_\theta(\cdot \mid x)$. The teacher then provides token-level targets along the student-induced trajectory. A standard on-policy distillation loss can be written as

$$\mathcal{L}_{\mathrm{OPD}} = \mathbb{E}_{x \sim \mathcal{D},\, y \sim \pi_\theta(\cdot|x)} \left[\sum_{t=1}^{T} D(\pi_\theta(\cdot \mid x, y_{<t}) \,\|\, q_\phi(\cdot \mid x, y_{<t})) \right], \tag{1}$$

where $D(\cdot)$ denotes a divergence loss, either KL, reverse KL, or Jensen-Shannon (JS), as detailed in Section H. This formulation provides dense token-level supervision on the student's visited states.

Nevertheless, standard OPD treats each sampled trajectory independently. When multiple rollouts are generated for the same prompt, the teacher distribution for one rollout is usually computed without access to the successes and failures observed in the other rollouts. This prevents the teacher from exploiting local, instance-specific evidence contained in the rollout group.

4 MULTI-ROLLOUT ON-POLICY DISTILLATION

We propose **Multi-Rollout On-Policy Distillation** (MOPD), a peer-conditioned distillation framework that exploits the local structure of multiple on-policy rollouts generated for the same question as in Fig. 2. Standard on-policy distillation supervises each sampled trajectory independently: the teacher evaluates the student's current trajectory without direct access to the alternative attempts produced for the same instance. This ignores a useful source of instance-specific information. In reasoning tasks, successful rollouts reveal valid solution paths, while failed rollouts expose plausible but incorrect branches. MOPD uses this local contrast to construct a more informative teacher signal.

At a high level, MOPD first samples a group of student rollouts for each prompt, scores them with a verifier, partitions them into successful and failed trajectories, and then conditions the teacher on peer information from the same rollout group. The resulting teacher distribution is no longer only a global distribution over next tokens; it becomes a *peer-conditioned* distribution that can compare the current trajectory against other local attempts. This allows the teacher to provide sharper token-level supervision, especially near branching points where failed trajectories diverge from successful ones.

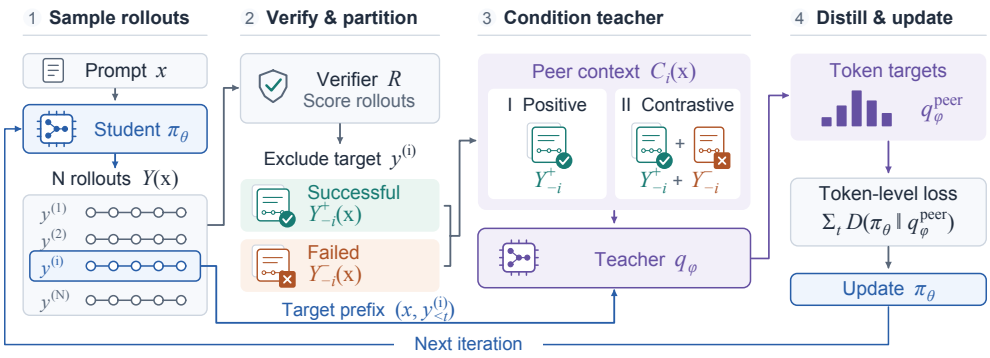


Figure 2: MOPD pipeline. For each prompt the student samples a rollout group, a verifier partitions it into successful peers $Y^+(x)$ and failed peers $Y^-(x)$, and the teacher is conditioned on peers drawn from that partition before supplying token-level targets for the target rollout $y^{(i)}$. Variant I conditions on $Y^+(x)$ only; Variant II conditions on both sets.

4.1 Multi-rollout peer conditioning

For each prompt x, the student samples a group of N on-policy trajectories,

$$Y(x) = \{y^{(i)}\}_{i=1}^{N}, \qquad y^{(i)} \sim \pi_\theta(\cdot \mid x). \tag{2}$$

Each trajectory is evaluated by a verifier or reward function R, and the rollout group is partitioned according to a success threshold τ:

$$Y^+(x) = \{y^{(i)} : R(x, y^{(i)}) \geq \tau\}, \qquad Y^-(x) = \{y^{(i)} : R(x, y^{(i)}) < \tau\}. \tag{3}$$

Here $Y^+(x)$ contains peer successes and $Y^-(x)$ contains peer failures. For a target rollout $y^{(i)}$, we exclude the target itself and define

$$Y^+_{-i}(x) = Y^+(x) \setminus \{y^{(i)}\}, \qquad Y^-_{-i}(x) = Y^-(x) \setminus \{y^{(i)}\}. \tag{4}$$

MOPD constructs a peer context from these two sets: $C_i(x) = \mathcal{C}_v\left(x, y^{(i)}, Y^+_{-i}(x), Y^-_{-i}(x)\right)$, where $\mathcal{C}_v$ denotes a context construction rule. The index v specifies which peer-conditioning strategy is used. Given this context, the teacher distribution for the target trajectory becomes $q_\phi^{\text{peer}}(\cdot \mid x, y^{(i)}_{<t}) = q_\phi(\cdot \mid x, C_i(x), y^{(i)}_{<t})$. Compared with a standard teacher distribution $q_\phi(\cdot \mid x, y^{(i)}_{<t})$, the peer-conditioned teacher can use other rollouts from the same instance as local evidence. Successful peers provide positive support for correct reasoning patterns, while failed peers provide negative evidence about misleading or invalid trajectories.

The rollout-level MOPD loss is

$$\mathcal{L}^{(i)}_{\text{MOPD}} = \sum_{t=1}^{T_i} D\Big(\pi_\theta(\cdot \mid x, y^{(i)}_{<t}) \,\|\, q_\phi^{\text{peer}}(\cdot \mid x, y^{(i)}_{<t})\Big), \tag{5}$$

where D is a token-level divergence. We use the generalized Jensen–Shannon divergence with skew coefficient $\alpha \in [0, 1]$, which interpolates between the two KL directions through the mixture $m_\alpha = (1-\alpha)\pi_\theta + \alpha\, q_\phi^{\text{peer}}$:

$$D_\alpha\Big(\pi_\theta \,\|\, q_\phi^{\text{peer}}\Big) = (1-\alpha)\, D_{\text{KL}}(\pi_\theta \,\|\, m_\alpha) + \alpha\, D_{\text{KL}}\Big(q_\phi^{\text{peer}} \,\|\, m_\alpha\Big). \tag{6}$$

The endpoints recover the two familiar objectives: $\alpha{=}1$ gives mode-seeking reverse KL and $\alpha{=}0$ gives mode-covering forward KL, while $\alpha{=}0.5$ gives the symmetric Jensen–Shannon divergence. This is a single knob rather than two separate settings: we use $\alpha{=}1$ (reverse KL) for math, where the long traces made the symmetric objective the less stable of the two in preliminary runs, and $\alpha{=}0.5$ (symmetric JS) everywhere else. Section H states the properties of the endpoints.

MOPD optimizes Equation (5) alone. It is a pure distillation objective: there is no policy-gradient term and no value network, and the verifier enters only through the partition of Equation (3) that decides which peers the teacher sees.

4.2 PEER-CONTEXT CONSTRUCTION

The main design choice in MOPD is the construction of $C_i(x)$. We compare two natural choices: a positive-only context that exposes the teacher to additional successful peers, and a contrastive context that adds failed peers as structured negative evidence. This comparison isolates whether failure-as-negative-evidence sharpens the teacher signal beyond success-only conditioning.

Positive peer imitation. The first strategy conditions the teacher on successful peer trajectories only. Let $y^\star$ denote a primary successful rollout and let y^+ denote an additional successful peer, both selected from $Y_{-i}^+(x)$ when available. The context is

$$C_i^{(1)}(x) = \mathrm{Template}(y^\star) \oplus \mathrm{SuccessTemplate}(y^+), \tag{7}$$

where $\oplus$ denotes text concatenation. If only one successful peer is available, the context reduces to the primary successful trajectory, $C_i^{(1)}(x) = \mathrm{Template}(y^\star)$.

This variant treats successful rollouts as peer experts. It encourages the teacher to evaluate the target trajectory in light of alternative correct reasoning paths. Compared with using a single reference solution, positive peer imitation broadens the set of correct trajectories the teacher conditions on, so the distilled student is anchored to multiple valid derivation styles rather than one.

Contrastive success–failure conditioning. The second strategy conditions the teacher on both successful and failed peer trajectories. Given successful peers $y^\star, y^+ \in Y_{-i}^+(x)$ and a failed peer $y^- \in Y_{-i}^-(x)$, we define

$$C_i^{(2)}(x) = \mathrm{Template}(y^\star) \oplus \mathrm{SuccessTemplate}(y^+) \oplus \mathrm{FailureTemplate}(y^-). \tag{8}$$

This variant provides the teacher with contrastive local evidence. Successful rollouts indicate what the target trajectory should resemble, while failed rollouts identify plausible but incorrect reasoning patterns that should be avoided. The contrastive context can therefore sharpen token-level supervision by helping the teacher distinguish between superficially similar correct and incorrect trajectories. In tasks with sparse terminal rewards, this is especially useful because the failed peer exposes where an otherwise plausible solution path becomes invalid.

Peer availability and fallbacks. Equation (8) assumes a primary successful rollout $y^\star$, which does not always exist: early in training a whole group can fail, and late in training a whole group can succeed, leaving no failed peer. We therefore define the context for every availability pattern. Writing $T = \mathrm{Template}(y^\star)$, $S = \mathrm{SuccessTemplate}(y^+)$ and $F = \mathrm{FailureTemplate}(y^-)$ for brevity,

$$C_i^{(2)}(x) = \begin{cases} T \oplus S \oplus F, & |Y_{-i}^+(x)| \geq 2, \;\; |Y_{-i}^-(x)| \geq 1, \\ T \oplus S, & |Y_{-i}^+(x)| \geq 2, \;\; Y_{-i}^-(x) = \varnothing, \\ T \oplus F, & |Y_{-i}^+(x)| = 1, \;\; |Y_{-i}^-(x)| \geq 1, \\ T, & |Y_{-i}^+(x)| = 1, \;\; Y_{-i}^-(x) = \varnothing, \\ \varnothing, & Y_{-i}^+(x) = \varnothing, \end{cases} \tag{9}$$

where $y^\star, y^+ \in Y_{-i}^+(x)$, $y^- \in Y_{-i}^-(x)$, and the number of successful peers is capped at two. The second and fourth cases are the *all-success* groups: with no failed peer to instantiate F the context degrades to positive-only conditioning (Equation (7)). These are not hypothetical, since solve rates reach 71–87% late in training (Section 5.5). In the last case the peer context is empty and MOPD reduces exactly to standard on-policy distillation on the student's own trajectory, i.e. $q_\phi^{\mathrm{peer}} = q_\phi$ in Equation (5). MOPD therefore never degrades below the OPD baseline on prompts with no positive evidence, and recovers peer conditioning as soon as the student solves the prompt once. This matters in practice: at the start of training 57% of Qwen3-4B's LiveCodeBench-v6 prompts yield no successful rollout (Section 5.5).

We deliberately do *not* fall back to a failure-only context: Section 5.2 shows such a context leaves the teacher at or below chance on every discrimination metric.

Which peers are shown. Equation (9) fixes how many peers enter the context, not which ones. With $N{=}8$ the leave-one-out pools usually hold more candidates than slots. We take $y^\star$ and y^+ in sampling order — the two lowest-indexed successes in $Y^+_{-i}(x)$ — and draw y^- uniformly at random from $Y^-_{-i}(x)$. Since the group is resampled at every step, all three are redrawn each time. Section G.1 compares this against a deterministic alternative.

Unified view The two variants share a gated form $C_i(x) = \mathrm{Template}(y^\star) \oplus \lambda_+ A_+(x) \oplus \lambda_- A_-(x)$, with $A_\pm(x)$ the additional successful and failed demonstrations: Variant I is $(\lambda_+, \lambda_-) = (1, 0)$, Variant II is $(1, 1)$, and Equation (9) is this form with $\lambda_+ = \mathbb{1}[|Y^+_{-i}(x)| \geq 2]$, $\lambda_- = \mathbb{1}[|Y^-_{-i}(x)| \geq 1]$ and the context suppressed when $Y^+_{-i}(x) = \varnothing$. Algorithm 1 summarizes the training procedure.

5 Experiments

We compare MOPD against the base model, GRPO, and SDPO. SDPO conditions its self-teacher on environment feedback when available; MOPD instead conditions the teacher on same-prompt peer rollouts, both successful and failed. This tests whether peer-augmented teacher information adds anything beyond scalar-reward RL and feedback-conditioned self-distillation.

Benchmarks. We consider four families of tasks that require multi-step reasoning or precise execution. **Science QA.** We use the reasoning subsets (L3) from SciKnowEval (Feng et al., 2024), covering undergraduate-level questions in biology, chemistry, physics, and materials science. **Tool Use.** We evaluate tool-use reasoning on ToolAlpaca (Tang et al., 2023), where the model must map a user request and an API specification to the correct tool call. **Code.** We evaluate on LiveCodeBench v6 (LCBv6) (Jain et al., 2025), which contains 131 contest-style programming problems released between February and May 2025. Training and evaluation use a *temporal* split: we post-train on LiveCodeBench problems released *before* February 2025 and evaluate only on these 131 later-released problems, which are never seen during training, so the evaluation is inductive rather than in-sample. "Training prompts" elsewhere in the paper always means the pre-February split. **Math.** For math training, we use a filtered subset of DeepMath (He et al., 2025), selecting 57K examples with difficulty level at least 6 following (Yang et al., 2026). We evaluate on AIME2024, AIME2025, and HMMT25. **Metrics.** For code and math, we report mean@8 and pass@8, following standard multi-sample evaluation. All results are reported as percentages. More **implementation details** are in Section B.

5.1 Main Results for Self-Distillation

Math reasoning. Table 2 shows results on AIME2024, AIME2025, and HMMT25 using Qwen3-4B as the base model. MOPD improves over the base model and over SDPO on every split, with the largest margins on AIME2024 (28.54 vs. 16.89 mean@8) and HMMT25 Feb. (18.50 vs. 5.06). Against GRPO the picture is mixed, and we state it explicitly rather than aggregating it away: GRPO is ahead on AIME2025 and on HMMT25 Feb. pass@8, while MOPD leads on AIME2024, on HMMT25 Feb. mean@8, and on both HMMT25 Nov. metrics. The explanation is not the amount of feedback — GRPO, SDPO and MOPD see the same binary verifier outcome, and MOPD needs no textual environment feedback at all — but the shape of the update. GRPO's group-relative advantage spreads mass over whichever derivation paths get sampled, which suits AIME-style problems admitting many unrelated routes to one integer answer, whereas MOPD distills a mode-seeking reverse-KL target anchored to one peer derivation. Consistent with this, MOPD leads on mean@8 on three of four splits and trails mainly on pass@8, the coverage metric. **Science QA.** Table 3 shows MOPD achieves the best performance across all four domains, on science multi-choice questions benchmarks. **Tool Use.** Tool Use requires precise mapping from natural-language requests to structured API calls, our method outperforms every baseline in Table 5. Peer conditioning therefore extends beyond reasoning-rich settings to tasks where the primary challenge is accurate structured-output prediction.

Table 1: Cross-family results with Llama-3.2-3B-Instruct. The base policy is near zero on both benchmarks, i.e. the most extreme sparse-success regime.

	SciQA-Chem		**LCB-v6**	
Method	mean@8	pass@8	mean@8	pass@8
Base	0.18	0.92	0.00	0.00
GRPO	0.00	0.00	0.00	0.00
SDPO	9.61	28.04	8.06	20.52
MOPD	**24.23**	**64.92**	**25.27**	**43.26**

Table 2: Math reasoning results on AIME2024/2025 and HMMT25. All methods start from the same Qwen3-4B model and use the same generation budget. Bold marks the best result in each column.

	AIME2025		AIME2024		HMMT25 Feb.		HMMT25 Nov.	
Method	mean@8	pass@8	mean@8	pass@8	mean@8	pass@8	mean@8	pass@8
Qwen3-4B	17.92	32.57	16.89	31.60	5.06	10.00	9.58	13.91
GRPO	**25.80**	**36.66**	17.09	32.05	16.92	**20.94**	13.16	19.53
SDPO	7.81	16.66	7.29	16.29	8.59	13.61	12.01	18.49
MOPD	25.41	36.28	**28.54**	**33.12**	**18.50**	19.58	**15.83**	**22.47**

Table 3: ScienceQA accuracy by subject, with Qwen3-8B as the base policy.

Method	**Bio.**	**Chem.**	**Phys.**	**Mat.**
Qwen3-8B	30.89	42.98	58.44	65.59
GRPO	47.32	64.24	64.12	72.09
SDPO	50.60	62.91	67.36	72.34
MOPD	**55.69**	**74.29**	**76.25**	**78.59**

Table 4: LiveCodeBench-v6 across two Qwen3 scales.

	Qwen3-4B		Qwen3-8B	
Method	mean@8	pass@8	mean@8	pass@8
Base	28.80	49.36	30.97	53.02
GRPO	40.75	55.43	43.65	58.72
SDPO	48.84	63.23	49.49	64.34
MOPD	**57.01**	**65.48**	**61.82**	**67.23**

LiveCodeBench. Table 4 reports LCBv6 results for Qwen3-4B and Qwen3-8B, a setting where the environment supplies structured feedback (compiler errors, test verdicts) at the end of each rollout. MOPD improves over the base model, GRPO and SDPO at both scales, reaching 61.82 mean@8 and 67.23 pass@8 on Qwen3-8B.

Generalization beyond the Qwen3 family. On Llama-3.2-3B-Instruct (Table 1), a far weaker starting policy, MOPD improves over SDPO by 14.6 and 17.2 mean@8 points. The margin is wider than on Qwen3, which the mechanism predicts: a weak policy produces groups of many failures and few successes, exactly where one successful peer plus an explicit contrast is most informative.

Case Study. Comparing training-time rollouts and step-matched checkpoints of SDPO and MOPD on the same prompts (Section I) shows that **both start from the same failure pattern early in training, but MOPD suppresses it by checkpoint time** by repeatedly showing the teacher failed peers of that bug family with an explicit "avoid this mistake" instruction — a channel SDPO lacks.

5.2 Analysis of Self-Teacher Signal Quality

Setup: self-teacher signal evaluation. To understand why peer information helps, we analyze the self-teacher signal itself. For each question x we sample rollouts $\{y^{(i)}\}_{i=1}^{n}$ from the base policy and score each one with the verifier ($r_i \in [0,1]$) and with the self-teacher, using its average token log-probability $s_i = \frac{1}{T_i}\sum_{t=1}^{T_i} \log q_\phi(y_t^{(i)} \mid x, C_i(x), y_{<t}^{(i)})$. For a given prompt the candidate rollouts and their rewards are fixed and only the teacher-visible context $C_i(x)$ varies, which isolates how peer information affects the reliability of the signal. We therefore score each condition on the prompts where its context activates and report that count n per row in Tables 10 and 11. Since n varies by condition, these tables diagnose signal quality *where a context is available* and are not difficulty-matched; the difficulty-matched evidence is the downstream control of Section 5.4. The nine conditions are listed in Figure 3; they span no peer, label-agnostic random peers, success-only, failure-only, verifier-score-ranked, and mixed contexts, all on Qwen3-4B rollouts for LiveCodeBench-v6. **Metrics** cover ranking (mean Spearman ρ, mean Kendall's τ), discrimination (success AUC, point-biserial correlation) and calibration (sigmoid Brier score); all six are defined in Section G, which also states the aggregation convention needed to read their absolute values.

Results and interpretation. Figure 3, Figure 7 and Tables 10 and 11 together reveal how peer context composition affects teacher signal quality and downstream performance. 1) **Successful peers are the primary driver**: any condition with at least one successful peer lifts Success AUC from 0.13 to above 0.32, and Spearman ρ and Kendall τ from 0.00 to above 0.10 and 0.13; a failed peer alone

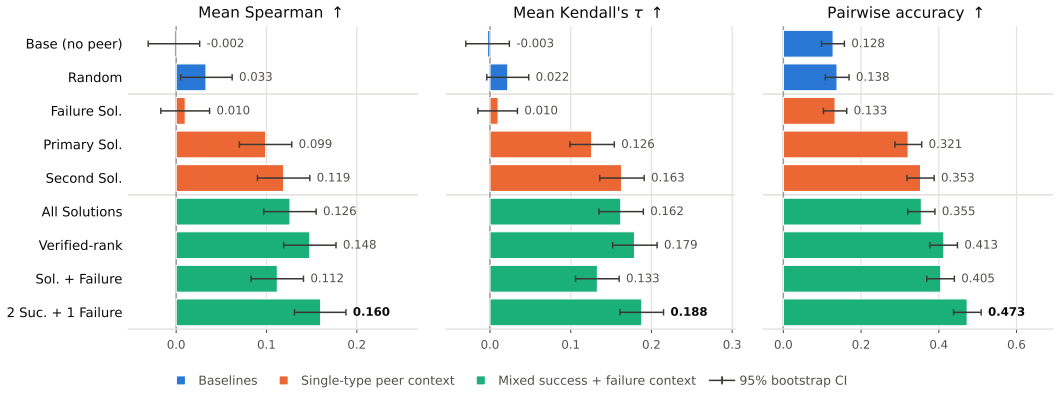


Figure 3: Self-teacher-signal *ranking* quality across the nine context conditions. Each panel reports a prompt-level metric averaged over the prompts on which that condition's context can be built; higher is better.

Table 5: Tool-use success rate, with Qwen3-8B as the base policy.

Method	Qwen3-8B	GRPO	SDPO	**MOPD**
Tool Use	59.11	60.32	63.45	**66.73**

Table 6: Ablation on peer-context construction for MOPD: combinations of successful and failed peer rollouts included in the self-teacher context. "k solution(s)" counts the peers placed in the context, drawn from the leave-one-out pools of Equation (4). The last two rows are budgeted by wall-clock time, not gradient steps, so they are not step-matched to the rows above (Section D).

	QA-Chemistry		**LiveCodeBench**	
Peer context $\mathcal{C}(u)$	**mean@8**	**pass@8**	**mean@8**	**pass@8**
1 successful solution	51.90	58.92	49.49	64.34
2 successful solutions	52.62	59.02	55.62	64.02
1 failure solution	35.67	37.64	46.72	54.86
1 success + 1 failure	60.50	68.12	58.90	62.01
2 success + 1 failure	74.29	83.53	61.82	67.23
All peers (2 h budget)	73.57	83.11	58.43	63.68
All peers (4 h budget)	81.25	84.49	58.95	64.71

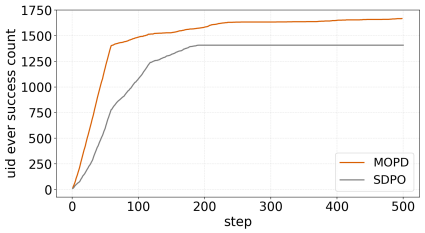


Figure 4: Cumulative ever-success count on the ScienceQA-Chemistry training split (1920 prompts): those with at least one correct rollout among their N samples by the current step. The first ~ 60 steps are one epoch at batch size 32, so the early rise also reflects prompts being visited for the first time.

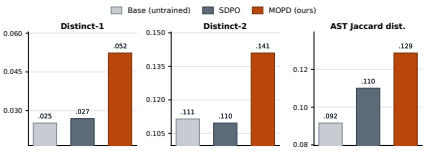


Figure 5: Diversity analysis.

barely moves any discrimination metric. 2) **Failed peers complement them**: adding a failure to a success-only context lifts Success AUC from 0.32 to 0.41, so negative evidence sharpens a boundary that positive evidence alone leaves blurred. 3) **The combination is best**: "2 success + 1 failure" tops 4 of the 5 ranking and discrimination metrics and gives the highest LCB downstream mean@8 among the compact contexts.

5.3 Solution Diversity Analysis

Does exposure to multiple peer rollouts make the student's own solutions more diverse? We measure two axes that are well defined for code, both comparing the responses a model produces *for the same prompt*: Distinct-1/2 for cross-rollout lexical variation, and AST-node Jaccard distance for variation in syntactic composition (definitions and caveats in Section J). Both are computed on LiveCodeBench-v6 checkpoints at step 50, 8 responses per problem. Figure 5 shows that MOPD

produces substantially more diverse outputs than both the base model and SDPO across all three metrics.

5.4 Isolating the mechanism: contrast or context?

A peer-conditioned teacher sees a longer, more structured prompt, so the gains might come from extra context or extra demonstrations rather than from the same-prompt success–failure contrast. We settle this with controls that fix the demonstration count — and hence the context length — and vary only *which* peers are shown: every row of Table 7 places exactly three peer rollouts in the teacher context on LiveCodeBench-v6 with Qwen3-4B under the same rollout budget, optimizer and number of updates. Three confounds separate cleanly. *Context length is not the driver*: random same-prompt peers occupy as many tokens as MOPD's and still lose 11.2 mean@8 points (45.83 vs. 57.01). *Locality matters*: successful rollouts from *other* prompts are worse still (37.10). *Failures carry signal of their own*: a success-only control with three verified successes matches MOPD in length and demonstration quality yet trails by 7.4 points (49.61), and ranking peers by verifier score and keeping the top three still falls 5.6 short (51.41). A different *objective* on the same group also falls short: verified-success SFT and DPO on same-prompt pairs, trained on exactly these rollouts, reach only 37.36 and 40.26 (Table 8). Full tables in Section D.

5.5 Behavior when successes are scarce

Starts with scarce success, but saturates at a higher success rate. We track the *ever-success ratio*: the fraction of training prompts on which the student has produced at least one successful rollout up to the current step. 1) In the first batch only 43% of LiveCodeBench-v6 prompts and 40% of ScienceQA-Chemistry prompts yield any success, so peer conditioning is unavailable on the majority of them; by the end of training the cumulative ratio reaches 71% and 87%. Figure 4 shows this as a count on ScienceQA-Chemistry, where MOPD ends at 1668 of 1920 training prompts (87%) against SDPO's 1407 (73%) and reaches any given level earlier under the same budget. 2) **The gain is largest exactly where successes are scarce.** Binning training prompts into terciles by their step-0 success rate (Figure 6a), the hard bin rises from 16% to 32% on LiveCodeBench-v6 and from 15% to 64% on ScienceQA-Chemistry, against 28% and 58% for SDPO at an identical budget; and at a reduced budget of 2 rollouts per prompt, where successes are rarer still, MOPD beats GRPO by a wide margin (Figure 6b). MOPD therefore makes *more* progress precisely where the empty-success-set concern applies. 3) **The mechanism is self-reinforcing through the policy, not a buffer**: no past successes are stored and the partition is redrawn from fresh rollouts every step, so an early success helps later only by raising the prompt's solve probability. The cumulative ratio is thus an upper envelope on peer availability, not the per-step rate at which Equation (9) fires.

6 Conclusion

We introduced MOPD, a peer-conditioned on-policy distillation framework that builds teacher signals from the student's own rollout group. Conditioning the teacher on both successful and failed peers turns independent trajectory imitation into a comparative process: successes give positive evidence for valid reasoning, failures expose the error patterns to avoid. Across code, math, science QA and tool use, MOPD improves over standard on-policy baselines and over baselines that reuse the same rollout group through verified-success SFT or preference optimization; matched-budget controls attribute the gain to the same-prompt success–failure contrast than to a longer teacher context, and the result transfers to a second model family. Effective distillation should exploit the student's rollout group as a structured collection of successes and failures rather than treating each trajectory in isolation.

AI use statement

In this work, we used generative AI tools for editing (grammar, spelling, and word choice), for drafting sections of the paper, for implementing standard methods in our codebase, and for retrieval and discovery of related work: candidate references were surfaced with AI assistance and every one of them was then read and verified against its primary source by an author before being cited or described. We did not use generative AI tools to generate research ideas, or to generate or fabricate experimental results. All AI-assisted text was reviewed and edited by the authors; all AI-assisted

code was read, executed, and tested by at least two authors, and every number reported in this paper comes from a run we performed ourselves. Generative AI was not used as a subject of study, as a research participant, or as an evaluator of our results. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

Ethics statement

This work studies post-training methods for language models on verifiable reasoning tasks. It involves no human subjects, no personally identifying data, and no newly collected datasets; all benchmarks used are publicly released research benchmarks (SciKnowEval, ToolAlpaca, LiveCodeBench-v6, DeepMath, AIME, HMMT). MOPD inherits the dual-use profile of general post-training methods: it makes models better at code and mathematical reasoning without introducing a new capability class, and it does not weaken or target model safety behavior. One consideration specific to our method is that MOPD amplifies whatever the verifier rewards. Section G.1 quantifies this by injecting verifier label noise; in a deployment where the verifier encodes a biased or gameable notion of success, peer conditioning would propagate that notion into token-level supervision, and we would recommend auditing the verifier before the training recipe.

Reproducibility statement

Our code, including the peer-context construction, the training configurations, and the logging of the ever-success statistics reported in Section 5.5, is available at `https://anonymous.4open.science/r/mopd_code-C7BC/`. Section 5 states the benchmarks, base models, rollout budget, learning rate, context budgets, and divergence choice for every experiment; the peer-context templates and the exact prompts are in the appendix, as are the definitions of all six teacher-signal metrics (Section G) and the diversity metrics (Section J). The matched-budget controls of Section 5.4 reuse the rollout groups of the main runs, so they can be reproduced from the same logs. Where results come from a single run rather than from multiple seeds, we say so in the corresponding table.

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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.13649.

Rong Bao, Donglei Yu, Kai Fan, and Minpeng Liao. Fixing distribution shifts of LLM self-critique via on-policy self-play training. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17680–17700, 2025. URL https://aclanthology.org/2025.acl-long.865/.

Nicolas Boizard, Kevin El Haddad, Céline Hudelot, and Pierre Colombo. Towards cross-tokenizer distillation: the universal logit distillation loss for LLMs. *Transactions on Machine Learning Research*, 2025. URL https://arxiv.org/abs/2402.12030.

Dan Busbridge, Amitis Shidani, Floris Weers, Jason Ramapuram, Etai Littwin, and Russ Webb. Distillation scaling laws. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2502.08606.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=bx24KpJ4Eb. Survey Certification, Featured Certification.

Alex J. Chan, Hao Sun, Samuel Holt, and Mihaela van der Schaar. Dense reward for free in reinforcement learning from human feedback. *arXiv preprint arXiv:2402.00782*, 2024. URL https://arxiv.org/abs/2402.00782.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 6621–6642. PMLR, 2024. URL https://proceedings.mlr.press/v235/chen24j.html.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Ken Ding. HDPO: Hybrid distillation policy optimization via privileged self-distillation. *arXiv preprint arXiv:2603.23871*, 2026. URL https://arxiv.org/abs/2603.23871.

Aradhya Dixit, Tianxi Liang, and Jai Telang. Project aletheia: Verifier-guided distillation of backtracking for small language models. In *Logical and Symbolic Reasoning in Language Models @ AAAI 2026*, 2026. URL https://openreview.net/forum?id=4zupWb2xmE.

Kehua Feng, Xinyi Shen, Weijie Wang, Xiang Zhuang, Yuqi Tang, Qiang Zhang, and Keyan Ding. SciKnowEval: Evaluating multi-level scientific knowledge of large language models. *arXiv preprint arXiv:2406.09098*, 2024. URL https://arxiv.org/abs/2406.09098.

Yuqian Fu, Tinghong Chen, Jiajun Chai, Xihuai Wang, Songjun Tu, Guojun Yin, Wei Lin, Qichao Zhang, Yuanheng Zhu, and Dongbin Zhao. SRFT: A single-stage method with supervised and reinforcement fine-tuning for reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=n6E0r6kQWQ.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/8ac015d409635f196f9e3e9dcfb9a94e-Abstract-Conference.html.

Xinyan Guan, Yanjiang Liu, Xinyu Lu, Boxi Cao, Ben He, Xianpei Han, Le Sun, Jie Lou, Bowen Yu, Yaojie Lu, et al. Search, verify and feedback: Towards next generation post-training paradigm of foundation models via verifier engineering. *arXiv preprint arXiv:2411.11504*, 2024. URL https://arxiv.org/abs/2411.11504.

Arnav Gudibande, Eric Wallace, Charlie Snell, Xinyang Geng, Hao Liu, Pieter Abbeel, Sergey Levine, and Dawn Song. The false promise of imitating proprietary LLMs. *arXiv preprint arXiv:2305.15717*, 2023. URL https://arxiv.org/abs/2305.15717.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://www.nature.com/articles/s41586-025-09422-z.

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. *arXiv preprint arXiv:2504.11456*, 2025. URL https://arxiv.org/abs/2504.11456.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. URL https://arxiv.org/abs/1503.02531.

Jiwoo Hong, Noah Lee, and James Thorne. ORPO: Monolithic preference optimization without reference model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11170–11189, 2024. URL https://aclanthology.org/2024.emnlp-main.626/.

Haiduo Huang, Jiangcheng Song, Yadong Zhang, and Pengju Ren. SelecTKD: Selective token-weighted knowledge distillation for LLMs. *arXiv preprint arXiv:2510.24021*, 2025. URL https://arxiv.org/abs/2510.24021.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026. URL https://arxiv.org/abs/2601.20802.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Yuxin Jiang, Chunkit Chan, Mingyang Chen, and Wei Wang. Lion: Adversarial distillation of proprietary large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://arxiv.org/abs/2305.12870.

Woogyeol Jin, Taywon Min, Yongjin Yang, Dennis Wei, Yi Zhou, Swanand Ravindra Kadhe, Nathalie Baracaldo, and Kimin Lee. Entropy-aware on-policy distillation of language models. *arXiv preprint arXiv:2603.07079*, 2026. URL https://arxiv.org/abs/2603.07079v3.

Seongryong Jung, Suwan Yoon, DongGeon Kim, and Hwanhee Lee. ToDi: Token-wise distillation via fine-grained divergence control. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://aclanthology.org/2025.emnlp-main.409/.

Jeonghye Kim, Xufang Luo, Minbeom Kim, Sangmook Lee, Dohyung Kim, Jiwon Jeon, Dongsheng Li, and Yuqing Yang. Why does self-distillation (sometimes) degrade the reasoning capability of LLMs? *arXiv preprint arXiv:2603.24472*, 2026. URL https://arxiv.org/abs/2603.24472.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1317–1327, 2016. doi: 10.18653/v1/D16-1139. URL https://aclanthology.org/D16-1139/.

Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. DistiLLM: Towards streamlined distillation for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.03898.

Jongwoo Ko, Tianyi Chen, Sungnyun Kim, Tianyu Ding, Luming Liang, Ilya Zharkov, and Se-Young Yun. DistiLLM-2: A contrastive approach boosts the distillation of LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2503.07067.

Jongwoo Ko, Sara Abdali, Young Jin Kim, Tianyi Chen, and Pashmina Cameron. Scaling reasoning efficiently via relaxed on-policy distillation. *arXiv preprint arXiv:2603.11137*, 2026. URL https://arxiv.org/abs/2603.11137.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 110–119, 2016. URL https://aclanthology.org/N16-1014/.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, and Ning Ding. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe, 2026a. URL https://arxiv.org/abs/2604.13016.

Yingru Li, Ziniu Li, and Jiacai Liu. A note on hybrid online reinforcement and imitation learning for LLMs: Formulations and algorithms. *arXiv preprint arXiv:2512.23097*, 2025a. URL https://arxiv.org/abs/2512.23097.

Yu Li, Shu Hong, and Tian Lan. Localizing credit at the divergence: Path-conditioned self-distillation for LLM reasoning. *arXiv preprint arXiv:2606.15576*, 2026b. URL https://arxiv.org/abs/2606.15576.

Zhuochun Li, Yuelyu Ji, Rui Meng, and Daqing He. Learning from committee: Reasoning distillation from a mixture of teachers with peer-review. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4190–4205, 2025b. doi: 10.18653/v1/2025.findings-acl.217. URL https://aclanthology.org/2025.findings-acl.217/.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

Yuanjie Lyu, Chengyu Wang, Jun Huang, and Tong Xu. Student-centered distillation narrows the agentic gap between small and large LLMs. *arXiv preprint arXiv:2509.14257*, 2025. URL https://arxiv.org/abs/2509.14257v3.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems*, volume 37, pp. 124198–124235, 2024. URL https://papers.neurips.cc/paper_files/paper/2024/hash/e099c1c9699814af0be873a175361713-Abstract-Conference.html.

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

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022. URL https://arxiv.org/abs/2203.02155.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *arXiv preprint arXiv:2602.04942*, 2026. URL https://arxiv.org/abs/2602.04942.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741, 2023. URL https://papers.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635. PMLR, 2011. URL https://proceedings.mlr.press/v15/ross11a.html.

Hejian Sang, Yuanda Xu, Zhengze Zhou, Ran He, Zhipeng Wang, and Jiachen Sun. CRISP: Compressed reasoning via iterative self-policy distillation. *arXiv preprint arXiv:2603.05433*, 2026. URL https://arxiv.org/abs/2603.05433v7.

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

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=A6Y7AqlzLW.

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

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026. URL https://arxiv.org/abs/2601.19897.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lNAyUngGFK. Expert Certification.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024. URL https://arxiv.org/abs/2408.03314.

Mingyang Song and Mao Zheng. A survey of on-policy distillation for large language models, 2026. URL https://arxiv.org/abs/2604.00626.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023. URL https://arxiv.org/abs/2306.05301.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jGbRWwIidy.

Yuqiao Wen, Zichao Li, Wenyu Du, and Lili Mou. f-divergence minimization for sequence-level knowledge distillation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://arxiv.org/abs/2307.15190.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why RLVR may or may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025a. URL https://arxiv.org/abs/2507.14843.

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*, 2025b. URL https://aclanthology.org/2025.coling-main.383/.

Hongling Xu, Qi Zhu, Heyuan Deng, Jinpeng Li, Lu Hou, Yasheng Wang, Lifeng Shang, Ruifeng Xu, and Fei Mi. KDRL: Post-training reasoning LLMs via unified knowledge distillation and reinforcement learning. *arXiv preprint arXiv:2506.02208*, 2025a. URL https://arxiv.org/abs/2506.02208.

Shicheng Xu, Liang Pang, Yunchang Zhu, Jia Gu, Zihao Wei, Jingcheng Deng, Feiyang Pan, Huawei Shen, and Xueqi Cheng. RLKD: Distilling LLMs' reasoning via reinforcement learning. *arXiv preprint arXiv:2505.16142*, 2025b. URL https://arxiv.org/abs/2505.16142.

Wenda Xu, Rujun Han, Zifeng Wang, Long T. Le, Dhruv Madeka, Lei Li, William Yang Wang, Rishabh Agarwal, Chen-Yu Lee, and Tomas Pfister. Speculative knowledge distillation: Bridging the teacher-student gap through interleaved sampling. In *The Thirteenth International Conference on Learning Representations*, 2025c. URL https://arxiv.org/abs/2410.11325.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024. URL https://arxiv.org/abs/2402.13116.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*, 2025. URL https://arxiv.org/abs/2504.14945.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

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*, 2026. URL https://arxiv.org/abs/2602.12125.

Zhaorui Yang, Tianyu Pang, Haozhe Feng, Han Wang, Wei Chen, Minfeng Zhu, and Qian Liu. Self-distillation bridges distribution gap in language model fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1028–1043, 2024. URL https://aclanthology.org/2024.acl-long.58/.

Tianzhu Ye, Li Dong, Zewen Chi, Xun Wu, Shaohan Huang, and Furu Wei. Black-box on-policy distillation of large language models. *arXiv preprint arXiv:2511.10643*, 2025. URL https://arxiv.org/abs/2511.10643.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026. URL https://arxiv.org/abs/2602.12275.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://papers.neurips.cc/paper_files/paper/2025/hash/537d5aa768c2d534016a4d06f87bc8fb-Abstract-Conference.html.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022. URL https://papers.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search. In *Advances in Neural Information Processing Systems*, volume 37, pp. 64735–64772, 2024. URL https://arxiv.org/abs/2406.03816.

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. *arXiv preprint arXiv:2602.15260*, 2026a. URL https://arxiv.org/abs/2602.15260.

Songming Zhang, Xue Zhang, Tong Zhang, Bojie Hu, Yufeng Chen, and Jinan Xu. AlignDistil: Token-level language model alignment as adaptive policy distillation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025a. URL https://arxiv.org/abs/2503.02832.

Xiaoying Zhang, Yipeng Zhang, Hao Sun, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-GRPO: Advancing LLM reasoning with natural language and numerical feedback. *arXiv preprint arXiv:2506.03106*, 2025b. URL https://arxiv.org/abs/2506.03106.

Zhaoyang Zhang, Shuli Jiang, Yantao Shen, Yuting Zhang, Dhananjay Ram, Shuo Yang, Zhuowen Tu, Wei Xia, and Stefano Soatto. Reinforcement-aware knowledge distillation for LLM reasoning. *arXiv preprint arXiv:2602.22495*, 2026b. URL https://arxiv.org/abs/2602.22495.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025c. URL https://arxiv.org/abs/2501.07301.

Shiwan Zhao, Zhihu Wang, Xuyang Zhao, Jiaming Zhou, Caiyue Xu, Chenfei Liu, Liting Zhang, Yuhang Jia, Yanzhe Zhang, Hualong Yu, Zichen Xu, Qicheng Li, and Yong Qin. Large language model post-training: A unified view of off-policy and on-policy learning, 2026a. URL https://arxiv.org/abs/2604.07941.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026b. URL https://arxiv.org/abs/2601.18734.

A More Related Work

On-Policy Distillation of Large Language Models Knowledge distillation classically transfers a teacher's predictive distribution to a smaller student through supervision on teacher- or human-curated sequences (Hinton et al., 2015; Kim & Rush, 2016; Xu et al., 2024). For autoregressive language models, this off-policy regime exhibits a well-known limitation: the student is trained on states it never visits at inference, and prediction errors compound over long generations (Gudibande et al., 2023; Ross et al., 2011; Song & Zheng, 2026). The central response is to move the supervision signal onto trajectories drawn from the student's own evolving policy, anchoring training on the distribution that will be operated over at deployment (Agarwal et al., 2024; Gu et al., 2024; Ko et al., 2024; 2025). Within this framework, a substantial body of work focuses on how to construct and weight the per-token learning signal. Forward-, reverse-, and skewed-divergence objectives offer different trade-offs between mode coverage and mode seeking (Gu et al., 2024; Ko et al., 2024; Wen et al., 2023; Wu et al., 2025b; Jung et al., 2025). Other works adapt the supervision to vocabulary mismatches between teacher and student (Boizard et al., 2025), reweight tokens by uncertainty or salience (Huang et al., 2025; Zhang et al., 2025a), or interpolate between fully on- and off-policy regimes via teacher-prefixed rollouts (Xu et al., 2025c; Zhang et al., 2026a). Black-box variants drop the teacher-logit requirement and supervise the student through sampled outputs, sequence-level rewards, or adversarial discrimination (Ye et al., 2025; Jiang et al., 2023). The paradigm now underlies widely deployed post-training pipelines for instruction-following, code generation, and mathematical reasoning (Yang et al., 2025). A complementary thread documents stability and failure modes of these objectives, including capacity-induced learnability gaps, divergence pathologies, and entropy collapse, and proposes adaptive losses or curricula to mitigate them (Busbridge et al., 2025; Li et al., 2026a; Jin et al., 2026). Despite this breadth, the dominant assumption is that the teacher signal for each rollout is constructed independently, leaving cross-rollout structure within a sampling group unexploited.

Self-Distillation and Self-Improvement for Reasoning A second line of work treats the same model as both teacher and student, exploiting either privileged conditioning or repeated sampling to manufacture supervision without an external teacher. Iterative self-training procedures generate candidate solutions, filter by verification or reward, and fine-tune on the surviving trajectories, yielding a coarse form of self-distillation for reasoning and alignment (Zelikman et al., 2022; Singh et al., 2024; Chen et al., 2024; Yang et al., 2024). Subsequent work develops soft self-distillation: one copy of the model is conditioned on additional context—ground-truth solutions, hints, demonstrations, or environment feedback—and produces per-token targets for a second copy that lacks that conditioning (Ye et al., 2026; Zhao et al., 2026b; Hübotter et al., 2026; Shenfeld et al., 2026; Penaloza et al., 2026; Ding, 2026; Yang et al., 2026). Self-distillation has been particularly active in long chain-of-thought reasoning, where supervision is sparse and individual traces are expensive. One direction compresses or reshapes long reasoning traces during distillation (Sang et al., 2026; Ko et al., 2026). Another aggregates information across multiple sampled solutions through consistency, voting, or committee teachers, transforming an ensemble of attempts into a richer learning signal (Wang et al., 2023; Muennighoff et al., 2025; Li et al., 2025b; Snell et al., 2024). A third direction interleaves rollouts with structured self-critique or search before constructing supervision, expanding the teacher's effective view of the problem (Zhang et al., 2024; Bao et al., 2025; Zhang et al., 2025b). Recent analyses also document the limitations of these procedures, noting that self-distillation can degrade reasoning when the student is asked to imitate signals beyond its competence or when the privileged context misaligns with what the student can reproduce (Kim et al., 2026). Across these directions, the common thread is that a single model can sharpen its own training signal when given structured access to additional information or to alternative attempts; whether and how those alternative attempts should be jointly conditioned on within a single distillation step remains underexplored.

Multi-Rollout Reinforcement Learning and Verifier-Guided Improvement A third line of related work concerns post-training paradigms that exploit groups of rollouts and external verification rather than per-token teacher matching. Reinforcement learning from human or AI feedback aligns model behavior with preference labels collected over pairs or groups of completions, with subsequent advances exploring offline, reference-free, and online variants (Ouyang et al., 2022; Rafailov et al., 2023; Hong et al., 2024; Meng et al., 2024; Casper et al., 2023). Reinforcement learning with verifiable rewards extends this idea by replacing learned reward models with programmatic checkers—

answer matching, unit tests, type-checkers—and updating the policy from group-relative advantages computed over multiple sampled rollouts per prompt (Shao et al., 2024; Wen et al., 2026; Wu et al., 2025a; Yue et al., 2025; Fu et al., 2026). A unified perspective on off- and on-policy post-training places these methods alongside on-policy distillation and clarifies when each provides the most useful supervision signal (Zhao et al., 2026a; Yan et al., 2025; Li et al., 2025a). Process- and outcome-level verifiers refine the supervision beyond a single sparse trajectory reward. Step-level verifiers and process reward models grade intermediate states, providing denser credit assignment for reasoning (Lightman et al., 2024; Cobbe et al., 2021; Setlur et al., 2025; Zhang et al., 2025c; Guan et al., 2024; Dixit et al., 2026). Hybrid post-training pipelines couple on-policy distillation with reinforcement learning from verifiers, using teacher-derived signals as an auxiliary objective alongside reward optimization (Xu et al., 2025b;a; Lyu et al., 2025; Zhang et al., 2026b). While these methods do leverage multiple rollouts per prompt—typically through advantage normalization, rejection sampling, or success-conditioned filtering—the supervisory signal for any single trajectory is computed without conditioning on the contents of the others. The work most adjacent to ours therefore treats successes and failures from the same group as complementary streams of evidence at the level of advantages or filters, but stops short of routing both into a single, peer-conditioned distillation target.

Positioning of current method. MOPD is deliberately conservative at the level of machinery: the divergence objective, the optimizer, and the training loop are those of standard on-policy distillation, and the only change is the evidence the teacher is conditioned on. It is therefore closest in spirit to *privileged-information* and *context distillation*, where a teacher sees information withheld from the student and its behavior is compressed back into the student (Hübotter et al., 2026; Ye et al., 2026; Zhao et al., 2026b; Yang et al., 2026). Those methods draw the privileged context from outside the rollout group — ground-truth answers, hints, or environment feedback — so the context is the same in kind for every rollout of a prompt. Our claim is that the rollout group itself is a source of privileged context, and that its value is *comparative*: what helps the teacher is seeing how this prompt's successes and failures differ, information no external hint supplies and that a longer generic context does not substitute for. Section 5.4 tests this claim against matched-budget controls rather than asserting it.

Related directions. Subsequent work conditions a self-teacher on a successful peer drawn from the same training group (Li et al., 2026b), which corresponds to the positive-only variant $C_i^{(1)}$ of Equation (7); we report that variant as a control at three sizes in Table 6 and Table 7. On a different axis, ExOPD (Yang et al., 2026) generalizes the OPD objective itself by rescaling the reward term against the KL regularizer; that is orthogonal to our question — it changes the objective while holding the teacher's evidence fixed, whereas MOPD holds the objective fixed and changes the evidence — and the two should compose.

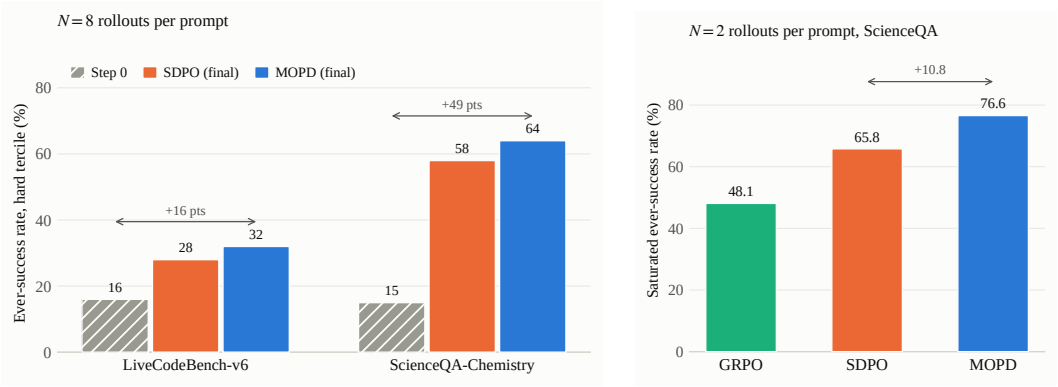
B More Implementation Details

Algorithm 1 MOPD Algorithm

Require: $\mathcal{D}$, π_θ, q_ϕ, R, N, τ, $\mathcal{C}_v$, α
Ensure: Updated student π_θ
for each training iteration **do**
  Draw a prompt batch $\mathcal{B} \subset \mathcal{D}$.
  for each $x \in \mathcal{B}$ **do**
    Sample rollout group $Y(x)$ as in Eq. (2).
    Score and partition rollouts as in Eq. (3).
    for each rollout $i \in \{1, \ldots, N\}$ **do**
      Form $Y^{\pm}_{-i}(x)$ (Eq. (4)); build
      $C_i(x)$ by Eq. (9).
      Query q_ϕ^{peer}; compute
      $\mathcal{L}^{(i)}_{\text{MOPD}}$ (Eq. (5), (6)).
    end for
  end for
  $\mathcal{L}_{\text{MOPD}} \leftarrow \frac{1}{|\mathcal{B}|N} \sum_{x \in \mathcal{B}} \sum_{i=1}^{N} \mathcal{L}^{(i)}_{\text{MOPD}}$
  Update π_θ by minimizing $\mathcal{L}_{\text{MOPD}}$.
end for
return π_θ

Training protocol. For each prompt, we sample a group of on-policy rollouts and evaluate them using a task-specific verifier or reward function. The resulting rollouts are partitioned into successful and failed sets. We then construct the privileged teacher context from peer rollouts, using two successful rollouts and one failed rollout unless otherwise specified. This design exposes the teacher to both correct solution patterns and informative failure modes. Across methods, we keep the rollout generation budget and optimization protocol fixed to ensure a fair comparison. Full RL training runs are expensive, so we report multi-run statistics where they are most informative rather than everywhere: three seeds per method on Science QA (Table 16), and three seeds for the peer-selection ablation on LiveCodeBench-v6 (Table 13), where the comparison is between two variants of MOPD itself and the quantity of interest is precisely the spread. The remaining tables report a single training run per method, as is common at this training scale; we accordingly avoid drawing conclusions from differences smaller than the spread reported in Table 16.

Implementation details. Training is implemented in `verl` with asynchronous vLLM rollouts and FSDP-based optimization. We use a prompt batch size of 32 and sample 8 rollouts per prompt during training. The actor learning rate is set to 1×10^{-5} with 10 warmup steps. Two sequence budgets matter and we set them explicitly. **Per peer.** Every peer in the teacher context is itself a student rollout, so it inherits the generation cap: a maximum prompt length of 2048 tokens and a maximum response length of 4096 tokens (8192 for math). No single peer can therefore exceed 8192 tokens, which bounds the context contribution of each demonstration independently of how many are shown. **Overall.** The assembled peer-conditioned teacher prompt is budgeted at $32{,}768$ tokens, i.e. Qwen3's full native context, so no RoPE extension is used. That budget covers the worst case the construction can produce — the prompt plus three peers at the per-peer cap, $2048 + 3 \times 8192 = 26{,}624$ tokens — with roughly 6k tokens to spare for the templates and chat formatting. The per-peer cap is thus the binding constraint in normal operation; the overall budget is a safety net, and in the rare event that it is reached the assembled prompt is truncated from the right, shortening peers rather than dropping them whole. The student's own trajectory is appended *after* this prompt and is never truncated, so the token-level targets always span the full target rollout. Validation evaluates 8 samples per prompt. Teacher log-probabilities are truncated to the top $k = 100$ vocabulary entries (`distillation_topk`=100) and both distributions are renormalized over that shared support before the divergence is evaluated, so no ε-floor or division by an unnormalized zero arises. We use FlashAttention-2 with bfloat16 model weights throughout. The divergence is the generalized Jensen–Shannon divergence of Equation (6), whose skew coefficient is the `alpha` field of the configuration: $\alpha = 0.5$ (symmetric JS) for code, science QA and tool use, and $\alpha = 1.0$ (reverse KL) for math. This is the *only* role α plays — it selects the divergence, not a weight between two loss terms. MOPD's objective is the token-level



(a) Hard tercile, $N = 8$ rollouts/prompt.

(b) All prompts, $N = 2$ rollouts/prompt.

Figure 6: **Behavior when successes are scarce.** (a) Ever-success rate on the hard tercile of training prompts at the standard budget of $N = 8$ rollouts per prompt. (b) Saturated ever-success rate over *all* ScienceQA prompts at a reduced budget of $N = 2$. The two panels report different populations and are not directly comparable; each compares methods within its own setting.

divergence and nothing else: *no value network is instantiated, no advantages are estimated, and no policy-gradient term is optimized*. The `critic` entries in the `verl` configuration are framework defaults that are never used, and the `timing_s/adv` column of Table 15 is the framework's reward post-processing stage, which is why it costs only 1–2 s per step.

Choice of divergence. The two settings above are the two endpoints-and-midpoint of a single parameter, so "reverse KL for math, JS elsewhere" is one choice of α rather than two objectives. It was a stability choice made on preliminary runs rather than a tuned hyperparameter: on the long math traces (8192-token responses) the symmetric objective was the less stable of the two in our setting, while on the shorter code, science-QA and tool-use responses it trained stably and we preferred it for being symmetric and bounded. We did not sweep α per domain, and we flag the sensitivity of MOPD to it as an open question rather than a settled one.

C Scarce Success Experiment

Group-based policy optimization only gets a learning signal from a prompt once at least one of its rollouts succeeds. If all N rollouts fail, the group-normalized advantages are all zero and the prompt adds no gradient. So when successes are scarce, what limits learning is often whether a method can find *any* success at all, more than how well it uses the successes it has. We measure this with the *ever-success rate*: the fraction of training prompts that have produced at least one correct rollout by a given training step. A prompt counts as solved once it succeeds and stays counted, so the curve can only go up. Its final value, the *saturated* ever-success rate, is the share of prompts the method ever solves during training.

We make successes scarce in two ways. In Figure 6a, we keep the standard budget of $N = 8$ rollouts per prompt and look only at the *hard tercile*: the third of training prompts on which the initial policy has the lowest pass rate. On these prompts the base model rarely succeeds, so a successful rollout mostly has to come from what the method learns during training. In Figure 6b, we cut the rollout budget to $N = 2$ and report the saturated ever-success rate over all ScienceQA prompts. With only two samples per prompt, even prompts of moderate difficulty often get all-fail groups, so this setting tests how well each method explores under a tight sampling budget.

In both settings, MOPD reaches a higher ever-success rate than SDPO and GRPO. It also rises faster in the early phase of training / keeps improving after the baseline plateaus. This means MOPD turns more of the zero-signal prompts into ones that produce useful gradients, instead of only sharpening performance on prompts the policy can already solve. The two panels use different prompt populations and rollout budgets, so their absolute values should not be compared with each other. Each panel compares methods within its own setting.

Table 7: Matched-budget controls on LiveCodeBench-v6 with Qwen3-4B. Every peer-conditioned row places exactly three peer rollouts in the teacher context, so context length and demonstration count are held fixed and only the *selection rule* changes. All rows share the same rollout budget, optimizer, and number of updates.

Peer selection rule	**Same prompt?**	mean@8	pass@8
Base model (no training)	—	28.80	49.36
Unrelated-prompt peers (3 successes)	no	37.10	55.29
Random peers (3, verifier label ignored)	yes	45.83	57.58
Success-only (3 successes, no failure)	yes	49.61	60.67
Verifier-score-ranked (top-3 by reward)	yes	51.41	62.28
MOPD (2 successes + 1 failure)	yes	**57.01**	**65.48**

Table 8: Methods that reuse the same rollout group, LiveCodeBench-v6 with Qwen3-4B, matched rollout and optimization budgets.

Method	mean@8	pass@8
Base model	28.80	49.36
Verified-success SFT (RFT/STaR-style)	37.36	51.47
DPO (same-prompt success–failure pairs)	40.26	53.02
MOPD	**57.01**	**65.48**

D Ablation Experiments on Matched-budget controls

Table 7 separates three confounds at once. **Context length is not the driver**: random same-prompt peers occupy exactly as many tokens as MOPD's peers and still lose 11.2 mean@8 points. **Locality matters**: filling the peer slots with successful rollouts from *other* prompts is worse than random peers from the same prompt, and only 8.3 points above the untrained base model, confirming that the value of a peer comes from its being an attempt at the same problem. **Failures carry signal of their own**: the success-only control uses three successful peers, so it matches MOPD in both length and demonstration quality, yet it trails the mixed context by 7.4 mean@8 points. The strongest control, ranking peers by verifier score and keeping the top three, closes part of the gap but still falls 5.6 points short — consistent with the teacher-signal analysis in Section 5.2, where the same ordering appears across all five alignment metrics.

Baselines that reuse the same rollout group. The controls above vary the teacher context. A complementary question is whether the rollout group is better exploited by a different *objective* altogether. We therefore train two standard alternatives on exactly the same rollouts, under the same rollout and optimization budget: verified-success SFT, which fine-tunes on the successful rollouts only in the style of RFT/STaR (Zelikman et al., 2022; Singh et al., 2024), and DPO on same-prompt success–failure pairs (Rafailov et al., 2023).

Table 8 shows that both alternatives improve substantially over the base model — the rollout group is clearly informative on its own — but both remain well below MOPD. Filtering to successes and imitating them discards the failed rollouts entirely; pairwise preference optimization uses both, but compresses each rollout into a single scalar comparison. MOPD instead routes the same partition into a token-level teacher signal, which appears to extract considerably more from the identical set of trajectories.

E Computational overhead

Computational overhead on long reasoning traces. Concatenating peer rollouts lengthens the teacher's input, which is a concern for tasks with very long traces. The breakdown in Table 15 shows

why the cost stays bounded: on LiveCodeBench the dominant per-step cost is reward computation (400s, since the verifier compiles and runs unit tests), and MOPD does not touch that phase at all — it lengthens only the update phase, from 78s to 90s, for a 9.1% increase in total step time. The 33.3% figure on QA-Chemistry comes from the opposite regime, where verification is essentially free (0.1s) so the update dominates. The implication runs against the usual intuition: for *verification-bound* long-horizon settings such as repo-level programming, where most wall-clock time goes to container builds, compilation, and test execution, peer conditioning is a small fraction of total cost. The overhead is mildest in exactly the long-horizon regime where the concern is sharpest.

How much peer context is useful? Table 10, Table 11 show that two successes plus one failure is the best compact context, and that supplying every available peer helps on ScienceQA but not on LiveCodeBench-v6. We read this as task-dependent rather than as an attention limit: ScienceQA-Chemistry is multiple-choice, so additional correct peers act as redundant confirming evidence, whereas on LiveCodeBench homogeneous successful programs dilute the success–failure contrast that carries the signal. How much peer context helps should therefore track how diverse the student's solutions are for a given task.

F Does Peer Information Improve Teacher–Student Distillation?

We next study whether peer information remains useful in a teacher–student setting, where the supervision signal is provided by a larger teacher model and used to train a smaller student. Our general knowledge distillation (GKD) pipeline follows an offline teacher-forcing paradigm: for each input prompt, a larger teacher model generates 8 completions, which are then converted into supervised fine-tuning examples. The student is trained by maximum-likelihood learning on these teacher responses. In on-policy distillation (OPD), the teacher provides supervision conditioned on on-policy student responses. Our teacher–student variant, MOPD (TS), further augments this teacher context with peer information: successful and failed peer rollouts.

Table 9: Teacher–student on-policy distillation results. All methods share the same rollout budget and hyperparameter.

	ToolUse		**LiveCodeBench**	
Method	mean@8	pass@8	mean@8	pass@8
Qwen3-4B (S)	53.27	58.58	28.80	49.36
Qwen3-14B (T)	56.02	60.89	46.75	65.16
GKD (TS)	55.91	60.05	46.05	56.67
OPD (TS)	59.63	61.58	60.92	67.30
SDPO (Self)	62.04	65.97	48.84	63.23
MOPD (TS)	**66.44**	**68.61**	**61.92**	**67.35**
MOPD (Self)	64.61	66.81	57.01	65.48

Table 9 reveals several trends in peer-conditioned distillation. 1) Teacher–student and self-distillation trade places by domain rather than one dominating: on LiveCodeBench the teacher–student pipeline is far ahead of self-distillation (60.92 vs. 48.84 mean@8), whereas on ToolUse self-distillation is ahead (62.04 vs. 59.63), consistent with a 14B teacher's prior helping most where the task is hardest for the 4B student. 2) A distilled student can exceed the teacher it is distilled from: on LiveCodeBench both OPD (TS) at 60.92 and MOPD (TS) at 61.92 beat the Qwen3-14B teacher's own 46.75 mean@8. We attribute this primarily to on-policy training on student-visited states rather than to peer conditioning, and we note that the privileged context is supplied to the teacher *during training only* — the student is unconditioned at inference. 3) Adding peer information helps in both settings, but the size of the gain is domain-dependent and should not be over-read: on ToolUse it is large (+6.81 mean@8 over OPD (TS)), whereas on LiveCodeBench the margin over OPD (TS) is +1.00 mean@8 and +0.05 pass@8 — the latter is well inside the run-to-run spread reported in Table 16 and Table 13 and we do not claim it as a gain. 4) On-policy methods outperform their off-policy counterparts (GKD) across both domains, consistent with the distribution-shift argument motivating on-policy training.

G TEACHER SIGNAL QUALITY METRIC

Mean Spearman Correlation. Spearman correlation measures the agreement between the rank ordering induced by teacher scores and the rank ordering induced by rewards. For a prompt, let $\mathrm{rank}(s_i)$ and $\mathrm{rank}(r_i)$ denote the average ranks under ties. The prompt-level Spearman correlation is

$$\rho_{\mathrm{S}} \;=\; \mathrm{corr}\big(\mathrm{rank}(s_1),\ldots,\mathrm{rank}(s_n),\ \mathrm{rank}(r_1),\ldots,\mathrm{rank}(r_n)\big),$$

where $\mathrm{corr}(\cdot,\cdot)$ is the Pearson correlation coefficient. We report the mean of this quantity across prompts. Higher values indicate that the teacher better preserves the reward-induced ordering.

Mean Kendall's τ. Kendall's τ evaluates pairwise ranking consistency between teacher scores and rewards. For each unordered pair (i, j) with $i < j$, let the pair be *concordant* if $(s_i - s_j)(r_i - r_j) > 0$ and *discordant* if $(s_i - s_j)(r_i - r_j) < 0$. Pairs tied in either score or reward are ignored. The prompt-level metric is

$$\tau \;=\; \frac{C - D}{C + D},$$

where C and D are the numbers of concordant and discordant pairs, respectively. We then average τ across prompts. Higher values indicate stronger ordinal agreement.

Success AUC. Success AUC treats the teacher score as a binary classifier score for success. Let $\mathcal{P} = \{i : r_i = 1\}$ and $\mathcal{N} = \{j : r_j = 0\}$ ($y^{(i)}$ denotes the token sequence throughout, r_i its verifier reward). The prompt-level AUC is computed by comparing all positive–negative pairs:

$$\mathrm{AUC} \;=\; \frac{1}{|\mathcal{P}||\mathcal{N}|}\sum_{i\in\mathcal{P}}\sum_{j\in\mathcal{N}}\left(\mathbf{1}[s_i > s_j] + \frac{1}{2}\mathbf{1}[s_i = s_j]\right).$$

This is the probability that a randomly chosen successful response receives a higher teacher score than a randomly chosen unsuccessful one, with ties receiving half credit. Higher is better.

Success Point-Biserial Correlation. To quantify the linear association between teacher scores and binary success labels, we compute the point-biserial correlation, which is simply the Pearson correlation between the continuous teacher scores and the binary labels:

$$\rho_{\mathrm{PB}} \;=\; \mathrm{corr}(s_1,\ldots,s_n,\ r_1,\ldots,r_n).$$

A larger positive value indicates that successful responses tend to receive higher teacher scores.

Success Brier Score (Sigmoid). To assess probabilistic calibration of teacher scores with respect to success, we first map each score to a pseudo-probability using the sigmoid transform

$$p_i \;=\; \sigma(s_i) \;=\; \frac{1}{1 + e^{-s_i}},$$

and then compute the Brier score

$$\mathrm{Brier} \;=\; \frac{1}{n}\sum_{i=1}^{n}(p_i - r_i)^2.$$

This metric measures the mean squared error between predicted success probabilities and observed binary outcomes. Lower values indicate better calibration.

Caveat: this Brier score is not a valid calibration ranking across conditions. Because q_ϕ is a probability distribution, every token log-probability is non-positive and hence $s_i \le 0$, so $p_i = \sigma(s_i) \in (0, \frac{1}{2}]$ for every candidate. A correct rollout ($r_i = 1$) therefore contributes at least $(\frac{1}{2} - 1)^2 = 0.25$ no matter how confident the teacher is, while an incorrect one contributes arbitrarily little as $s_i \to -\infty$. On a candidate pool dominated by failures, any condition that uniformly depresses teacher log-probabilities scores *better* on this metric without discriminating at all. This is exactly what the *Failure Sol.* row shows in Table 11: the best Brier value (0.173) alongside chance-level AUC (0.133). We report the number for completeness and read only the ranking and discrimination columns as evidence. A calibration comparison that would be meaningful here requires first mapping scores through an affine transform fit on held-out prompts (Platt or temperature scaling, $\sigma(as_i + b)$) before computing the Brier score or ECE; we leave that to future work rather than presenting the raw-sigmoid numbers as calibration evidence.

Condition	#Prompts	Mean Spearman ↑	Mean Kendall's τ ↑
Baselines			
Base (no peer)	131	-0.002 [-0.031, 0.026]	-0.003 [-0.030, 0.024]
Random	131	0.033 [0.005, 0.062]	0.022 [-0.004, 0.048]
Single-type peer context			
Failure Sol.	109	0.010 [-0.017, 0.037]	0.010 [-0.015, 0.034]
Primary Sol.	56	0.099 [0.070, 0.128]	0.126 [0.099, 0.154]
Second Sol.	51	0.119 [0.090, 0.148]	0.163 [0.136, 0.191]
Mixed success + failure context			
All Solutions	51	0.126 [0.097, 0.155]	0.162 [0.135, 0.190]
Verified-rank	51	0.148 [0.119, 0.177]	0.179 [0.152, 0.207]
Sol. + Failure	34	0.112 [0.083, 0.141]	0.133 [0.106, 0.160]
2 Suc. + 1 Failure	29	**0.160** [0.131, 0.188]	**0.188** [0.161, 0.215]

Table 10: Teacher-signal *ranking* metrics for all nine context conditions, with 95% prompt-level bootstrap confidence intervals in brackets ($10{,}000$ resamples over prompts). #Prompts is the number of LiveCodeBench-v6 prompts on which the condition's context can be instantiated, and it necessarily shrinks as a condition demands more peers of a given type; rows with different #Prompts are therefore not difficulty-matched against each other (see Section 5.4 for the difficulty-matched comparison). Prompts on which a metric is undefined — no reward-distinguishable rollout pair — are scored as 0 rather than dropped, which shrinks every value toward 0 in proportion to how many such prompts a row contains. Rows are grouped exactly as in Figure 3: *baselines* place no peer or label-agnostic peers in the context, *single-type* contexts contain only successful or only failed peers, and *mixed* contexts contain both. Higher is better.

Aggregation convention, and why $\mathrm{PairAcc} \neq (1+\tau)/2$ in our tables. Two conventions govern how the per-prompt quantities above become the numbers in Table 10 and Table 11, and both affect how they should be read. First, each condition is evaluated only on the prompts whose rollout group can instantiate its context, so the number of prompts n differs by condition (reported per row). Second, a prompt whose rollouts all carry the same verifier reward admits no reward-distinguishable pair, leaving τ and Success AUC undefined; we score such prompts as 0 and keep them in the average rather than dropping them. On LiveCodeBench-v6 this is not a rare event — 57% of prompts yield no successful rollout for Qwen3-4B — which is why the n=131 baseline row reports 0.128 rather than a value near 0.5. Note that under binary rewards and untied scores the per-prompt identity $\mathrm{PairAcc} = (1+\tau)/2$ holds exactly, yet it does not appear to hold between the *columns* of our tables. This is a direct consequence of the second convention rather than an inconsistency: scoring a degenerate prompt as 0 shrinks every metric toward 0, which leaves a metric centred at 0 (τ, Spearman ρ, point-biserial) almost unchanged while pulling a metric centred at 0.5 (Success AUC) far below chance. Writing f for the fraction of degenerate prompts in a row, the reported values satisfy $\overline{\tau} = (1-f)\,\tau^{\dagger}$ and $\overline{\mathrm{PairAcc}} = (1-f)\,(1+\tau^{\dagger})/2$ with $\tau^{\dagger}$ the mean over non-degenerate prompts, and the two columns of every row are consistent with this pair of equations. The comparison the tables support is therefore between conditions under a common convention, not against an absolute chance level of 0.5.

Tables 10 and 11 report prompt-level teacher-signal metrics across the nine context conditions, each with a 95% prompt-level bootstrap confidence interval. The two baseline rows bound the alternative explanation that peer conditioning merely adds context: the *Random* condition fills the same number of peer slots while ignoring verifier labels and barely moves away from the no-peer base (Success AUC 0.137 versus 0.128, overlapping intervals), so context alone is not sufficient. Within the mixed group, *Verified-rank* keeps the three highest-reward peers and is the strongest alternative to an explicit contrast — its peer set also contains both successes and high-scoring failures, but ordered implicitly by reward rather than labelled — reaching 0.420 Success AUC. The cleanest comparison it supports is against *Second Sol.* and *All Solutions*, which are scored on the same number of prompts (51): ranking peers by verifier score is worth roughly $+0.06$ Success AUC over an unranked success-only context of the same size. Its margin against *2 Suc.+1 Failure* (0.420 vs. 0.473) is *not* a controlled comparison, because the two rows are averaged over different prompt sets (51 vs. 29); the controlled version of that question is the matched-budget downstream comparison of Table 7, where verifier-score-ranked peers train on the same prompts as MOPD and still trail it by 5.6 mean@8 points. The *2 Suc.+1 Failure*

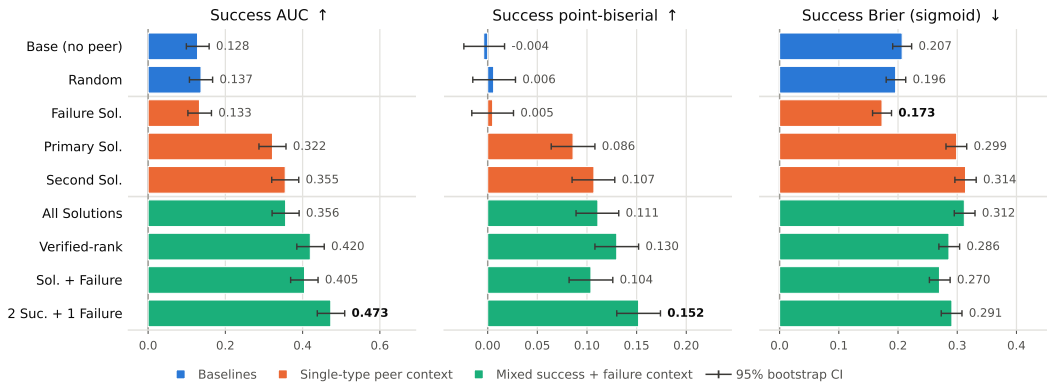


Figure 7: Self-teacher-signal *discrimination and calibration* metrics, the companion to Figure 3. Higher is better except for the Brier score, whose ordering is not meaningful across conditions for the reason given above.

Condition	#Prompts	Success AUC ↑	Success Point-Biserial ↑	Success Brier (Sig.) ↓
Baselines				
Base (no peer)	131	0.128 [0.099, 0.158]	-0.004 [-0.024, 0.017]	0.207 [0.191, 0.223]
Random	131	0.137 [0.107, 0.167]	0.006 [-0.015, 0.028]	0.196 [0.180, 0.213]
Single-type peer context				
Failure Sol.	109	0.133 [0.103, 0.164]	0.005 [-0.016, 0.026]	*0.173* [0.157, 0.189]
Primary Sol.	56	0.322 [0.287, 0.357]	0.086 [0.064, 0.108]	0.299 [0.281, 0.316]
Second Sol.	51	0.355 [0.320, 0.390]	0.107 [0.085, 0.128]	0.314 [0.296, 0.332]
Mixed success + failure context				
All Solutions	51	0.356 [0.321, 0.391]	0.111 [0.089, 0.132]	0.312 [0.295, 0.330]
Verified-rank	51	0.420 [0.385, 0.456]	0.130 [0.108, 0.152]	0.286 [0.269, 0.304]
Sol. + Failure	34	0.405 [0.369, 0.440]	0.104 [0.082, 0.126]	0.270 [0.253, 0.288]
2 Suc. + 1 Failure	29	**0.473** [0.438, 0.509]	**0.152** [0.130, 0.174]	0.291 [0.273, 0.308]

Table 11: Teacher-signal *discrimination and calibration* metrics for the same nine context conditions and the same prompt sets as Table 10, with 95% prompt-level bootstrap confidence intervals. Higher is better except for the Brier score. The best Brier value is italicised rather than bolded because, as explained below, the sigmoid Brier score of an unnormalized log-probability is minimised by a teacher that is uniformly pessimistic and is not a valid calibration ranking here.

condition attains the highest value of every ranking and discrimination metric in these tables — Mean Spearman (0.1595), Mean Kendall's τ (0.1879), Success AUC (0.4733), and Success Point-Biserial (0.1520) — on the subset of prompts where a two-success-plus-one-failure context can be built. In contrast, the *Failure Sol.* condition performs the worst across nearly all ranking and discrimination metrics, with a Mean Spearman of only 0.0097 and a Success AUC of 0.1332, suggesting that failure demonstrations alone offer little useful signal. *Failure Sol.* attains the lowest Brier score (0.1728), but its discrimination metrics also collapse to near-zero, so this calibration value reflects a degenerate near-uniform output rather than a useful signal. Among the single-type success conditions, *Second Sol.* slightly outperforms *Primary Sol.* across most metrics, while the mixed contexts that pair successes with an explicit failure (*Sol.+Failure*, *2 Suc.+1 Failure*) improve Success AUC over either of them.

G.1 Robustness and scope

Imperfect verifiers. MOPD partitions rollouts using the verifier, so verifier noise is a natural concern. We note first that this dependence is not specific to MOPD: every method we compare against reads the same verifier labels, and MOPD needs no verifier call beyond the one that already scores each rollout. The relevant question is therefore relative robustness. We flip each verifier label

Table 12: Robustness to verifier label noise on ScienceQA-Chemistry (mean@8). Labels are flipped independently with the stated probability for all methods.

Noise rate	0%	10%	20%	50%
GRPO	64.24	62.06	60.20	37.52
MOPD	**74.29**	**73.51**	**68.81**	**43.24**

Table 13: Random versus deterministic peer selection on LiveCodeBench-v6 with **Qwen3-8B**, mean and sample standard deviation over three independent runs (the same dispersion statistic as Table 16). The matched-budget controls in Tables 7 and 8 use Qwen3-4B; at the 8B scale used here the random-sampling row reproduces the Qwen3-8B MOPD entry of Table 4.

Peer selection	mean@8	Std. dev.
Random sampling (default)	61.82	1.8
Deterministic (max AST-Jaccard)	61.74	**1.6**

with probability 10%, 20%, and 50% before constructing Y^+ and Y^-, applying the same corruption to every method, and report mean@8 on ScienceQA-Chemistry.

Both methods tolerate mild noise: at 10% MOPD loses 0.78 points and GRPO loses 2.18. Degradation becomes substantial by 20%, and at 50% the labels are pure noise, so the collapse toward the unsupervised regime is expected rather than informative. MOPD retains its margin over GRPO at every noise level, so peer conditioning does not amplify label noise relative to the verifier dependence that RLVR already carries.

Variance and deterministic peer selection. Table 16 reports MOPD over three independent runs on Science QA and shows moderate variance, largest on Physics (± 2.8) and Chemistry (± 2.0). Part of this is specific to peer conditioning: because the peer partition is redrawn from freshly sampled rollouts at every step, the particular successes and failures that reach the teacher differ from run to run, on top of the instability inherent to on-policy RL. This suggests a mitigation: choose peers deterministically instead of sampling them. We select the failed peer with maximal AST-node Jaccard distance to the target rollout — the most structurally distinct failure, and therefore the most informative contrast — together with the two successes of minimal mean AST-node Jaccard distance to the other successes in $Y^+_{-i}(x)$, i.e. the medoids of the success set.

Two scope limits are worth stating plainly. First, this criterion is defined for code only: it needs a parseable syntax tree, so it does not transfer as written to ScienceQA or ToolUse, the very domains whose run-to-run spread motivated it. A text-compatible analogue — edit distance or embedding distance in place of AST-node Jaccard — is the natural substitute but we have not evaluated it. Second, the comparison in Table 13 is three runs per arm; a spread of 1.8 versus 1.6 over three runs is not a statistically meaningful difference ($F = 1.125$ on 2 and 2 degrees of freedom). What Table 13 supports is the weaker and still useful claim that deterministic peer selection *matches* random sampling on the mean (61.74 vs. 61.82), so removing peer-sampling stochasticity costs nothing; whether it actually reduces variance would need more runs than we can afford at this training scale. We therefore present it as an exploratory heuristic rather than a validated variance-reduction technique.

H Preliminary on Divergence-based distillation

Divergence-based distillation. A common approach to distillation is to minimize a divergence between the teacher distribution q_ϕ and the student distribution π_θ at each token position. In language model distillation, the direction of the KL divergence induces qualitatively different behavior.

The *forward KL* objective,

$$D_{\mathrm{KL}}(q_\phi(\cdot \mid c_t) \,\|\, \pi_\theta(\cdot \mid c_t)), \tag{10}$$

penalizes the student for assigning low probability to tokens favored by the teacher. It is therefore mode-covering: when the teacher assigns mass to multiple plausible continuations, forward KL encourages the student to preserve this diversity.

The *reverse KL* objective,

$$D_{\mathrm{KL}}(\pi_\theta(\cdot \mid c_t) \,\|\, q_\phi(\cdot \mid c_t)), \tag{11}$$

penalizes the student for placing probability mass on tokens that the teacher considers unlikely. This objective is mode-seeking and is often attractive in on-policy settings because the expectation is taken over the student distribution. However, reverse KL may over-concentrate probability mass on a subset of teacher-preferred modes, especially when the teacher distribution is uncertain or when the supervision signal is incomplete.

A symmetric alternative is the Jensen–Shannon divergence,

$$D_{\mathrm{JS}}(p \,\|\, q) = \frac{1}{2} D_{\mathrm{KL}}(p \,\|\, m) + \frac{1}{2} D_{\mathrm{KL}}(q \,\|\, m), \qquad m = \frac{1}{2}(p + q). \tag{12}$$

D_{JS} is symmetric and bounded. In principle, it can provide a more stable compromise between mode-covering and mode-seeking behavior. Equation (6) is the skewed generalization of this family: its α slides the mixture point between the two KL directions, recovering forward KL at α=0, the symmetric D_{JS} above at α=0.5, and reverse KL at α=1. In practice, however, computing such distribution-level objectives over the full vocabulary may be expensive for large language models, motivating approximations based on sampled tokens, top-k support, or teacher-provided log-probabilities.

I Case Studies

We present three complementary forms of evidence for MOPD's qualitative advantage. Case Studies 1 and 2 isolate checkpoint-level semantic failures: both models see the same task, but MOPD preserves the crucial constraints that SDPO drops. Case Study 1 additionally provides direct training-time evidence linking the checkpoint gap to MOPD's peer-conditioning mechanism. Case Study 3 examines training dynamics, showing that MOPD internalizes correct behavior earlier while SDPO remains unstable late in training.

I.1 Case Study 1: Identifier-space composition (checkpoint gap + training-time mechanism)

Candidate pair. The bib-mapping task provides a direct SDPO-versus-MOPD comparison. On this prompt, the SDPO checkpoint at training step 50 reaches accuracy 0.175, while the MOPD checkpoint at the same step solves it perfectly.

SDPO failure. The SDPO trajectory mixes up the role of the permutation arrays:

```
bib_to_person = [0] * (N + 1)
for i in range(N):
    bib_to_person[Q[i]] = P[i]
result = [str(bib_to_person[Q[i]]) for i in range(N)]
```

This is incorrect in two ways: `P[i]` is the stared-at person, not the wearer of bib `Q[i]`, and the final answer stops in person-ID space instead of projecting back to the target person's bib number. The program preserves the rough shape of the mapping task but composes the wrong relations.

MOPD success. The corresponding MOPD sample keeps the two identifier spaces separate throughout:

```
bib_to_person = [0] * (N + 1)
result = []
for i in range(N):
    bib_to_person[Q[i]] = i + 1
for i in range(1, N + 1):
    person = bib_to_person[i]
    target_person = P[person - 1]
    result.append(str(Q[target_person - 1]))
```

This trajectory performs the full chain correctly: `bib` → `wearing person` → `stared-at person` → `target bib`.

Training-time context: MOPD encounters the same failure pattern during rollout. The same bib-mapping prompt appears in the MOPD rollout stream at training step 1 with seven failed rollouts and one successful rollout; the failed rollouts exhibit the same identifier-space ambiguity that SDPO retains at checkpoint time. It is worth being precise about which updates carry the corrective signal here, since the leave-one-out rule of Equation (4) governs it. When the target $y^{(i)}$ is one of the *seven failed* rollouts, the leave-one-out pools are $|Y^+_{-i}(x)| = 1$ and $|Y^-_{-i}(x)| = 6$, so the third branch of Equation (9) applies and the teacher is conditioned on $T \oplus F$: the verified correct program together with a sibling failure carrying the same bug and an explicit "avoid this mistake" instruction. Seven of the eight token-level updates on this prompt are therefore contrastive, and they are precisely the updates applied to the trajectories that contain the bug. When the target is the *single successful* rollout, excluding it empties the positive pool ($Y^+_{-i}(x) = \varnothing$), the context is empty by the last branch, and that one update reduces to standard on-policy distillation — we do not self-condition on the target, and we do not fall back to a failure-only context. SDPO has no equivalent channel on any of the eight updates.

I.2 Case Study 2: Dropped middle-character constraint

Candidate pair. The evenly spaced `A-B-C` counting task gives a second clean contrast. On this prompt, the SDPO checkpoint at training step 50 reaches accuracy 0.825, while the MOPD checkpoint at the same step is fully correct.

SDPO failure. The SDPO code enforces equal spacing and checks the two endpoints, but drops the requirement that the middle character must be `B`:

```
for j in range(1, n-1):
    for i in range(j):
        if S[i] == 'A':
            k = 2 * j - i
            if k < n and S[k] == 'C':
                count += 1
```

This overcounts invalid triples such as `A-R-C` in the test case `ARC`, where the correct answer is 0. The failure is a local semantic omission: the positional structure is preserved, but one label constraint from the specification is forgotten.

MOPD success. The MOPD sample reinstates the missing check explicitly:

```
for j in range(1, n - 1):
    if s[j] == 'B':
        for i in range(j):
            if s[i] == 'A':
                k = 2 * j - i
                if k < n and s[k] == 'C':
                    count += 1
```

The program is structurally close to the SDPO attempt but preserves the full symbolic constraint set.

Takeaway. Case Studies 1 and 2 together show that MOPD's advantage in avoiding the failure pattern is not tied to a single error type. In Case Study 1, SDPO fails by composing the wrong identifier relations; here it fails by dropping one clause from the specification. MOPD succeeds in both settings; the contrast holds across two distinct error families rather than depending on a single failure mode.

I.3 Case Study 3: Training dynamics — MOPD stabilizes early, SDPO fails late

Candidate pair. The direction-opposite task illustrates the training-time gap. The SDPO checkpoint fails on this prompt at both step 50 and step 100 with accuracy 0.0: the step-50 sample is truncated, and the step-100 sample is flagged for a missing "`` ```python `` block. The MOPD checkpoint at step 50 already solves the same prompt with accuracy 1.0.

Early MOPD signal. In the MOPD rollout stream, a correct solution first appears at rollout step 10 with score 1.0—the earliest observed point at which the repaired behavior is present in training-time rollouts for this prompt.

Late SDPO failure. At rollout step 80, SDPO still fails with accuracy 0.0, receiving feedback: `Incorrect Format: Put your code inside a "'python ... "' block.`

Takeaway. The gap here is not algorithmic: MOPD internalizes basic output-format discipline by approximately step 10, while SDPO has not stabilized that behavior even late in training.

I.4 Overall Pattern

Taken together, the three case studies on LiveCodeBench-v6 reveal a consistent picture. The baseline's failures are narrow and consequential—a dropped constraint, a confused identifier space, an unresolved format requirement—rather than wholesale algorithmic breakdowns. These are precisely the errors that peer conditioning is designed to surface: by exposing the teacher to failed peers that exhibit the same bug, MOPD produces a sharper supervision signal at the critical decision points. Case Study 1 compared through training logs; Case Studies 2 and 3 provide corroborating evidence across a different failure type and a training-dynamics lens. The combined evidence supports the conclusion that MOPD's gains arise from a more targeted teacher signal, not from discovering fundamentally different solution strategies.

J Diversity

Distinct-n. Distinct-n (Li et al., 2016) is the ratio of unique n-grams to total n-grams across the concatenated token stream of all responses:

$$\text{Distinct-}n = \frac{|\{\text{unique } n\text{-grams}\}|}{\text{total } n\text{-grams}}. \tag{13}$$

We report Distinct-1 and Distinct-2.

AST node Jaccard distance (code only). Each response is parsed into a Python AST and its node-type multiset $\mathbf{a}_i$ is extracted. Pairwise Jaccard distance is

$$d_{ij}^{\text{AST}} = 1 - \frac{\sum_t \min(a_{it},\, a_{jt})}{\sum_t \max(a_{it},\, a_{jt})}, \tag{14}$$

where t ranges over all AST node types present in either response. Responses that fail to parse are excluded; the metric is omitted for a prompt if fewer than two responses parse successfully. The per-prompt score is the mean over all valid pairs. Because $\mathbf{a}_i$ is a flat multiset of node *types*, this quantity measures variation in the syntactic composition of two programs, not in their tree topology: two programs using identical counts of the same constructs arranged in different nesting or control-flow structures receive a distance of 0. It is a lower bound on structural diversity, and we read it as such.

Code Extraction for Diversity Measurement. For code-focused diversity analysis, we do not compute metrics on the raw response text directly. Instead, each response is first converted into a code view using a simple fenced-code-block extraction rule. Specifically, we match Markdown code blocks of the form "'`python ...` "' or "' `...` "' with a regular expression, and if multiple code blocks appear in one response, we keep only the last matched block. The extracted block is then stripped of leading and trailing whitespace and used as the canonical code content for subsequent diversity computation. If no fenced code block is found, the entire response text is used as a fallback. This procedure is intentionally lightweight: it does not perform full Markdown parsing, and only the `python` language tag is explicitly removed. As a result, responses with other language tags may retain the tag text inside the extracted content.

Results. Table 14 summarizes the diversity metrics across the three candidate sets evaluated on 131 shared prompts with up to eight rollouts each. On Distinct-1 and Distinct-2, SDPO remains close to the untrained base model (0.027 vs. 0.025; 0.110 vs. 0.111), suggesting that its training signal does not encourage lexical variation. MOPD, by contrast, achieves nearly double the Distinct-1 score of SDPO (0.052 vs. 0.027) and a notable gain in Distinct-2 (0.141 vs. 0.110). A similar pattern holds at the structural level: AST Jaccard distance increases from 0.110 (SDPO) to 0.129 (MOPD), indicating that MOPD generates solutions with more structurally distinct implementations. Peer conditioning therefore improves correctness and the lexical and structural diversity of the student's outputs together. We stop short of attributing the diversity gain specifically to the *failed* peers: Table 14 compares the full two-success-plus-one-failure context against SDPO and the base policy, so it cannot separate the effect of negative evidence from that of simply seeing several demonstrations. Measuring the same metrics for the positive-only condition of Table 6 would isolate this, and we leave it to future work.

Table 14: Diversity metrics (mean over prompts, n=131). Higher is better for all columns. **Bold** marks the best value per column.

Method	Distinct-1	Distinct-2	AST Jaccard dist.
MOPD	**0.052**	**0.141**	**0.129**
SDPO	0.027	0.110	0.110
Baseline	0.025	0.111	0.092

MOPD produces rollouts with substantially higher lexical and structural diversity: Distinct-1 is roughly $2\times$ that of SDPO and the baseline, and the mean pairwise AST node-type Jaccard distance is 17% higher than SDPO.

K Training Time Analysis Statistics

Compute Resources. All self-distillation experiments are conducted on a single node of 8×H100 GPUs and a single node of 8×A100 GPUs. Teacher–student distillation experiments are conducted on 2 nodes each equipped with 8×H100 GPUs.

Time Cost. MOPD introduces moderate computational overhead relative to SDPO, increasing the total per-step time by 9.1% on LCB and 33.3% on QA-Chemistry, as detailed in Table 15. On coding tasks such as LCB, reward computation dominates training time because each rollout requires compilation and unit-test execution. MOPD uses the same reward-evaluation procedure as GRPO and SDPO and therefore does not increase this dominant cost. Its additional overhead primarily comes from the model-update stage, which increases from 78 s to 90 s on LCB and remains small compared with the 400 s reward-computation cost. On QA-Chemistry, where reward evaluation is inexpensive, the increase in update time is more visible in the overall runtime.

Table 15: Per-step teacher-query cost comparison. Wall-clock time (s) is measured per gradient update step on 8×H100 GPUs with batch size 32 and 8 rollouts per prompt, averaged over training steps after the first five. "Post." is the reward post-processing stage between scoring and the update (`timing_s/adv` in `verl`); neither method estimates advantages, which is why it is negligible.

Dataset	Method	Time Gen	Time Reward	Time Post.	Time Update	Total
LCB	SDPO	37	400	1	78	550
	MOPD	37	400	2	90	600
QA-Chemistry	SDPO	5	0.1	1	75	90
	MOPD	5	0.1	2	110	120

Time Cost. We report several wall-clock timing metrics in seconds to break down the cost of each training iteration. To avoid startup overhead, we report the average timing measured after the first five training steps, since earlier steps may include additional one-time loading and initialization costs.

`timing_s/gen` measures the time spent generating sampled responses from the current policy. `timing_s/reward` measures the time required to obtain training rewards, including reward-model scoring or task-specific reward computation. `timing_s/update_actor` measures the wall-clock time of the actor optimization stage. In the SDPO setting, this stage includes the student forward pass, the teacher forward pass used to produce teacher logits, the self-distillation loss computation, and the subsequent backward pass and optimizer update. Finally, `timing_s/adv` measures the framework's post-processing stage between scoring and the update; for MOPD this is reward post-processing only, since the objective estimates no advantages. `timing_s/step` measures the total duration of one training step, covering the full pipeline from rollout generation to optimization. The reported stage times do not sum exactly to the full step time because they only cover the main timed components of the training loop. The total step time also includes additional overhead such as data movement, batch reorganization, mask construction, metric logging, coordination between workers, and other small bookkeeping operations that are not exposed as separate timing entries. As a result, the difference between the sum of the listed stage times and the total step time can be understood as uncategorized system and orchestration overhead.

Table 15 reports the per-step wall-clock time breakdown for SDPO and MOPD across two datasets. The dominant cost in both methods is reward computation and parameter update; rollout generation and reward post-processing are comparatively negligible. MOPD introduces additional overhead primarily in the update phase, as the teacher must process longer peer-conditioned contexts. Despite this, the total per-step time increases modestly: from 550s to 600s on LCB **(a 9.1% increase)** and from 90s to 120s on QA-Chemistry **(a 33.3% increase)**. These results indicate that the computational overhead of peer conditioning is small relative to the overall training cost, and that MOPD's consistent performance gains are not simply an artifact of increased compute budget.

Table 16: Science QA results with standard deviation over 3 runs. Bold marks the best result in each column.

Method	Biology	Chemistry	Physics	Materials
SDPO	$50.60_{\pm 2.1}$	$62.91_{\pm 2.5}$	$67.36_{\pm 1.9}$	$72.34_{\pm 2.6}$
MOPD	$\mathbf{55.69}_{\pm 1.6}$	$\mathbf{74.29}_{\pm 2.0}$	$\mathbf{76.25}_{\pm 2.8}$	$\mathbf{78.59}_{\pm 1.9}$

Training stability. Table 16 reports the mean and standard deviation of MOPD across 3 independent runs on the Science QA benchmarks. While MOPD consistently achieves the best performance across all four domains, the standard deviations reveal moderate variance, particularly in Physics (± 2.8) and Chemistry (± 2.0), suggesting that training is not fully stabilized across runs. We attribute this to the stochastic nature of on-policy rollout sampling and the sensitivity of peer-context construction to the specific success/failure partition observed in each run. Improving training stability through more robust peer selection or variance reduction techniques remains an avenue for future work.

L Limitations

MOPD relies on the availability of multiple rollouts and a verifier or reward function that can partition them into successful and failed trajectories. This makes the method most directly applicable to verifiable domains such as code, math, tool use, and structured QA, while extending it to open-ended generation tasks may require reliable preference models or process-level evaluators. Peer conditioning increases context length and teacher-query cost when raw rollouts are concatenated, especially for long reasoning traces; selecting the most informative peers under context-length constraints is an open question. Finally, the quality of the teacher signal depends on the correctness of the success/failure labels: noisy verifiers or mislabeled rollouts may introduce misleading positive or negative evidence. Future work should study more robust peer selection, better handling of noisy feedback, and extensions to less easily verifiable tasks.

M Societal Impacts

MOPD may have positive societal impacts by improving the efficiency of post-training and making stronger reasoning capabilities more accessible in smaller language models. Better distillation can

reduce inference cost and energy use, and may help deploy capable models in resource-constrained settings. In domains such as code generation, tool use, scientific question answering, and mathematical reasoning, more reliable small models could support education, research assistance, and productivity tools.

At the same time, improving the reasoning and tool-use capabilities of smaller models may also increase risks. More capable and cheaper models could be misused for generating harmful code, automating deceptive content, or executing tool-use workflows in unintended ways. Because MOPD learns from both successful and failed trajectories, poor verifier design or biased feedback could also reinforce undesirable behavior if incorrect success/failure labels are treated as reliable supervision. These risks suggest that peer-conditioned distillation should be paired with careful verifier design, safety evaluations, misuse monitoring, and appropriate deployment safeguards, especially when applied to open-ended, high-impact, or tool-using systems.

N Declaration of LLM Usage

We used large language models as assistance tools during the preparation of this paper. Specifically, LLMs were used to help draft, revise, and polish parts of the manuscript, including improving clarity, grammar, and organization. We also used LLM assistance for code writing and debugging, such as drafting utility scripts, checking implementation details, and improving code readability, and for retrieval and discovery of related work. Every reference surfaced this way was read and verified against its primary source by an author before being cited; no claim about prior work in this paper rests on an unverified model output. LLMs were not used to generate research ideas or any experimental result.