The Role of Feedback Alignment in Self-Distillation

Semih Kara
Gensyn
semih@gensyn.ai

Oğuzhan Ersoy
Gensyn
oguzhan@gensyn.ai

Abstract

Language models often perform better when their prompts include helpful context, such as feedback on a previous attempt. Self-distillation trains the model to achieve this performance without the context at inference time. It matches the model's output distribution across two settings: a student that sees only the question, and a self-teacher that also sees the context. What the model learns depends on the context that the self-teacher receives, yet its design (e.g. its structure, granularity, and alignment with the student's reasoning) remains largely unexplored.

We study feedback design by training a solver on feedback from a frozen critic. We compare three conditions: (i) a binary reward (GRPO), (ii) the reference solution, and (iii) a step-by-step critique aligned to the solver's reasoning trace.

Step-aligned critique yields the largest gains, outperforming GRPO by 16.11 points and outperforming reference-solution-conditioned self-distillation by 5.27 points (in Avg@12). Per-token advantage analysis explains why: step-aligned feedback concentrates distributional shifts at the tokens where reasoning fails, acting as implicit process supervision. In contrast, the reference solution spreads signal across all tokens, because an alternative derivation diverges from the solver's trace even at correct steps. Our results suggest that structural alignment between feedback and the solver's reasoning is one of the main drivers of self-distillation effectiveness.

1 Introduction

Post-training with reinforcement learning has become the dominant recipe for improving LLM reasoning. The standard approach is reinforcement learning from verifiable rewards (RLVR) [Shao et al., 2024], which learns from a single scalar reward per rollout. This reward says whether the final answer is correct, but not where in the reasoning trace the model went wrong, making credit assignment difficult. Alternatively, distillation [Hinton et al., 2015] provides dense, token-level supervision. However, it requires access to the logits of a strong teacher, which may be hidden behind an API or may be too costly to transfer at scale.

Self-distillation [Hübotter et al., 2026, Zhao et al., 2026, Shenfeld et al., 2026] sidesteps both limitations. The same model plays two roles: a *student* conditioned on the question alone, and a *self-teacher* conditioned on the question plus some additional context c (e.g., execution traces, follow-up prompts, reference solutions, feedback from another model, etc.). Training minimizes a divergence between these two distributions, distilling capabilities that emerge in-context [Brown et al., 2020] into the context-free policy.

Prior work has used various forms of context for self-distillation: code execution traces [Hübotter et al., 2026], ground-truth solutions [Zhao et al., 2026], and feedback from other models [Song et al., 2026] or users [Wang et al., 2026, Buening et al., 2026]. In all cases, the context is treated as a fixed choice. No prior work, to our knowledge, studies how the design of the context affects what the model learns. Nonetheless, when the context is feedback from another model, the practitioner can

International Conference on Machine Learning (ICML), Seoul, South Korea, 2026. Workshop: RLxF: Reinforcement Learning from World Feedback.

design its structure. This raises the question: **what form of feedback produces the most effective self-teacher?**

We study this in the mathematical reasoning domain, using a solver–critic setup. We compare three conditions: (i) a binary reward (GRPO), (ii) the reference solution (REFSOL), and (iii) a step-by-step critique aligned to the solver's reasoning trace (STEPALIGNFB). Our main findings are:

- **STEPALIGNFB outperforms GRPO and REFSOL.** Step-aligned critique outperforms both GRPO and self-distillation with the reference solution on all aggregation-style accuracy metrics in our evaluation (see Section 4).
- **STEPALIGNFB provides implicit process supervision.** Per-token advantage analysis shows that step-aligned feedback concentrates distributional shifts at error-adjacent tokens, while solution-level feedback produces a more diffuse signal. The localization mirrors what a PRM provides [Lightman et al., 2024, Uesato et al., 2022], but is obtained without training a reward model or collecting per-step scalar labels.
- **Feedback alignment matters as much as feedback quality.** A complete, correct reference derivation is a strong signal, but in self-distillation it diffuses across the solver's rollout because the derivation diverges from the solver's trace in surface form even at correct steps. Step-aligned critique, by addressing the solver's actual trace, concentrates the signal at the tokens where reasoning goes wrong. Self-distillation already provides a form of process-level signal through its token-level advantages. *Step-aligned feedback amplifies this by concentrating the signal at the tokens where reasoning goes wrong.*

2 Background

Distillation. Knowledge distillation [Hinton et al., 2015] trains a student model π_θ to match a stronger teacher π_T by minimizing the per-token forward KL divergence:

$$\mathcal{L}_{\text{KD}} = \mathbb{E}_{y \sim \pi_T(\cdot|x)} \left[\text{KL}\big(\pi_T(y \mid x) \,\big\|\, \pi_\theta(y \mid x)\big) \right]. \tag{1}$$

This provides dense, token-level supervision but requires access to the logits of a strong teacher model (which is often not available). Standard distillation is also off-policy: the student trains on the teacher's rollouts[1], not its own. Thus, errors compound as the student's distribution drifts from the teacher's at inference time [Ross et al., 2011]. On-policy distillation [Agarwal et al., 2024, Lu and Thinking Machines, 2025] addresses this by having the student generate its own rollouts and optimizing a reverse-KL-style distillation objective using token-level supervision from the teacher, but it still requires access to the logits of a stronger teacher.

RLVR. Reinforcement learning with verifiable rewards removes the need for a teacher. Given a question x, the model samples G rollouts $\{y_i\}_{i=1}^G \sim \pi_\theta(\cdot \mid x)$, each scored by a binary reward $r_i \in \{0, 1\}$. GRPO [Shao et al., 2024] estimates per-rollout advantages via group normalization:

$$A_i^{\text{GRPO}} = \frac{r_i - \bar{r}}{\sigma(r)}, \tag{2}$$

where $\bar{r}$ and $\sigma(r)$ are the mean and standard deviation of rewards within the group, respectively. The policy is updated by maximizing[2]

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\{y_i\}_{i=1}^G \sim \pi_\theta(\cdot|x)} \left[\sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \log \pi_\theta(y_{i,t} \mid x, y_{i,<t})\, A_{i,t} \right].$$

Notice that $A_{i,t} = A_i^{\text{GRPO}}$ is constant across all tokens within a rollout.

Self-distillation. Self-distillation [Hübotter et al., 2026, Zhao et al., 2026, Shenfeld et al., 2026] unifies the dense supervision of distillation with the teacher-free and on-policy properties of RL. The same model serves as both student and teacher under different prompting contexts. The *student* is

[1] In next-token prediction, a rollout is the model's complete response to a prompt.

[2] We present the simplified, on-policy expression (without clipping) and remove the KL penalty. We refer to Shao et al. [2024] for the full details.

conditioned on the question alone, $\pi_\theta(\cdot \mid x, y_{<t})$, whereas the *self-teacher* is conditioned on the question augmented with some additional context c, giving the distribution $\pi_\theta(\cdot \mid x, c, y_{<t})$.

Training minimizes a per-token divergence (forward KL, reverse KL, or Janson-Shannon divergence) between these two distributions, yielding the loss function:

$$\mathcal{L}_{\text{SD}} = \mathbb{E}_{y \sim \pi_\theta(\cdot|x)} \left[D\big(\pi_\theta(y \mid x) \,\big\|\, \text{sg}\left[\pi_\theta(y \mid x, c)\right]\big) \right], \tag{3}$$

where D is a divergence and $\text{sg}[\cdot]$ denotes stop-gradient. Hübotter et al. [2026] show that, when D is the KL divergence, the gradient of $\mathcal{L}_{\text{SD}}$ has the same form as the gradient of $-\mathcal{J}_{\text{GRPO}}$, but with per-token advantages given by

$$A_t^{\text{SD}}(\hat{y}_t) = \log \pi_\theta(\hat{y}_t \mid x, c, y_{<t}) - \log \pi_\theta(\hat{y}_t \mid x, y_{<t}). \tag{4}$$

A_t^{SD} quantifies how much the context shifts the model's next-token prediction. Unlike the GRPO advantage, the self-distillation advantage varies at every token position, providing dense credit assignment automatically.

Self-distillation uses the *same* model π_θ in both roles, distinguished only by the context c. Thus, the quality of the resulting advantage (Eq. 4) depends heavily on c. When c is feedback from another model, the practitioner has control over its structure. The topic of this paper is the design of this structure.

3 Methodology

3.1 Setup

Solver–critic setup. We implement self-distillation in a solver–critic setup (Figure 1). For each math question x, a trainable *solver* π_θ generates step-tagged reasoning traces

$$y = \langle \texttt{step}_1 \rangle \ldots \langle \texttt{step}_N \rangle \langle \texttt{answer} \rangle.$$

A frozen *critic* π_{critic} produces feedback f on the solver's response. The solver is then trained with self-distillation (Eq. 3) using f as the context c.

The only variable across our experimental conditions is f. The solver, loss function, divergence, and all hyperparameters remain fixed, isolating feedback structure as the independent variable.

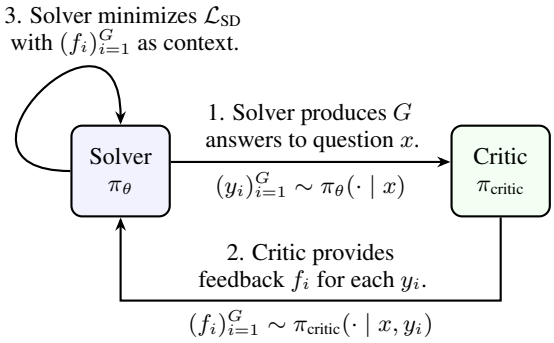


Figure 1: Solver–critic training loop. The critic produces feedback given the question x and each response y_i. Only the solver is updated.

Feedback conditions. We compare three forms of context/feedback:

- **GRPO (no feedback.)** Standard RLVR baseline. The solver generates G rollouts per question, each scored with a binary reward. Advantages are group-normalized rewards (Eq. 2). No critic, no self-distillation.
- **REFSOL (reference solution).** A chain-of-thought reference solution produced by a stronger model, either generated on-the-fly by the critic, or precomputed during dataset construction as by Li et al. [2024], Moshkov et al. [2025]. REFSOL mirrors the setup of Zhao et al. [2026].
- **STEPALIGNFB (step-aligned critique).** The critic receives both the solver's step-tagged response and the dataset's reference solution, then produces per-step feedback. We prompt it to copy correct steps verbatim and fix incorrect or incomplete steps while staying as close to the solver's reasoning trace as possible. We observed that copying correct steps verbatim activates the model's in-context copying behavior [Olsson et al., 2022], which sharpens advantage estimates, especially for correct steps. We give more details in Section 4.2.

3.2 Training Details

Our training recipe follows Zhao et al. [2026] with three modifications: no per-token KL clipping, increased student rollout cap, and a critic-conditioned teacher in the STEPALIGNFB experiments.

Self-distillation recipe. We use Qwen3-1.7B [Yang et al., 2025] as the solver. All rollouts are sampled on-policy, at temperature T=1.1 with a maximum token cap of 2048. We use a group size of $G = 1$ for self-distillation experiments, and set $G = 8$ for GRPO). Divergence measure is forward KL. We use the Thinking-Mode-Off student / Thinking-Mode-On teacher pairing identified as optimal by Zhao et al.. The teacher is held fixed at the initial (LoRA-free) base policy by disabling adapters at the teacher forward; no separate teacher checkpoint is loaded. LoRA [Hu et al., 2021] adapters are trained on all attention and MLP projections (r=64, α=128) with AdamW [Loshchilov and Hutter, 2019], learning rate 5×10^{-6}, effective batch size 32, bf16 precision, gradient checkpointing, and Flash Attention 2 [Dao et al., 2022]. Distributed training uses DeepSpeed ZeRO-2 with CPU optimizer offload [Rasley et al., 2020] across 4 H100 GPUs; on-policy generation is served in-process by vLLM [Kwon et al., 2023] in colocate mode. Details of the training configuration are in Appendix A.2.

Step-aligned feedback variant (STEPALIGNFB). In the step-aligned feedback variant, the teacher conditions on the critique f (for a group size of 1, we denote $f := f_1$) instead of the raw reference solution. The critique is produced once per rollout by a frozen reasoning model (Qwen/QwQ-32B; greedy decoding, T=0, Thinking-Mode-On) served via vLLM. We send a single user-message prompt that contains schema-driven grading instructions, the problem, the reference solution, and the student rollout; we strip the reasoning trace (`<think>...</think>`) from the returned content and splice only the structured grader output into the teacher context. The detailed critic prompt template is in Appendix A.1. The critic prompt is long, by design, to keep its feedback actionable and aligned step-by-step with the solution. Nonetheless, vLLM's automatic prefix caching substantially mitigates the resulting overhead.

Dataset. We train on a difficulty and formatting-filtered subset of OpenMathReasoning [Moshkov et al., 2025]. Prior work suggests that self-distillation's advantage over RLVR methods is most pronounced on harder problems. Since our focus is on how the *form* of feedback affects self-distillation, we deliberately target this harder-question regime, where self-distillation is the more relevant choice over RLVR. We additionally require that the problems be tractable for the trained Qwen3-1.7B, so that the model receives *actionable feedback* rather than failing outright and being handed a full reference-like solution by the critic (which would have collapsed REFSOL and STEPALIGNFB into the same setup). We also require that the critic itself can solve the problem, ensuring that its feedback is coherent and high quality. Concretely, we start from the subset of OpenMathReasoning with `pass_rate_72b_tir` > 0.85 (`pass_rate_72b_tir` is provided within the OpenMathReasoning dataset), then compute Avg@16 and average formatting accuracy over 16 attempts from the 1.7B model, keeping only problems where Avg@16 < 5/16 and the formatting accuracy is greater than 0.9. This yields 312 samples, of which we reserve 30 for evaluation and use the remaining 282 for training. We train for up to 7 epochs, saving checkpoints every 10 optimizer steps.

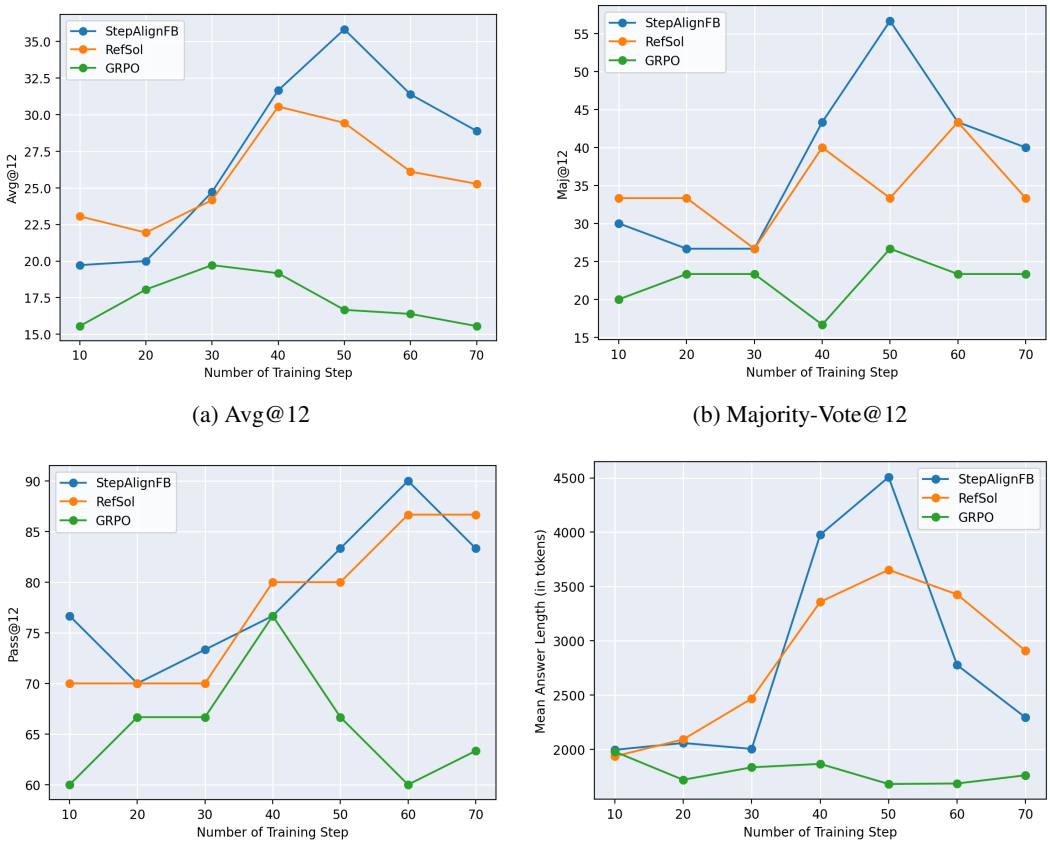
4 Results

4.1 Accuracy Results

We evaluate the three feedback variants on a held-out 30-sample test split from OpenMathReasoning. We report accuracy metrics over $n = 12$ samples per problem, averaged across the test set: Avg@12 (mean fraction of correct samples), Majority-Vote@12 (correctness of the majority answer), and Pass@12 (fraction of problems with at least one correct sample), alongside mean answer length.

We compare GRPO against the self-distillation variants (REFSOL and STEPALIGNFB) at matched generation compute. GRPO requires G=8 rollouts per prompt for its group-relative advantage estimate, while self-distillation uses a single rollout (G=1); at equal compute, GRPO thus consumes 8× fewer prompts per step. Nonetheless, each method trains for 7 epochs, ruling out the reduced dataset exposure as a confound.

Figure 2 plots each metric as a function of optimization step, showing how the three methods evolve through training. Table 1 summarizes the *per-metric best value attained by each method across checkpoints* (steps 10–70), with the step at which the maximum (or minimum, for answer length) occurred annotated as *s*. Because the methods reaches peak evaluation performance at different steps, we select the best checkpoint independently for each metric rather than reporting end-of-run values or fixing a single checkpoint per method.



(a) Avg@12

(b) Majority-Vote@12

(c) Pass@12

(d) Mean answer length

Figure 2: Self-distillation matches or exceeds GRPO across training on OpenMathReasoning. REFSOL conditions the teacher on the reference solution provided in the OpenMathReasoning dataset; STEPALIGNFB substitutes step-level feedback; GRPO is the reward-only baseline.

Table 1: **Per-metric best evaluation results on OpenMathReasoning.** For each (method, metric) pair we report the best value across all training checkpoints and annotate the step (s) at which the maximum (or minimum, for answer length) was attained. Bold marks the per-column winner.

	Pass@12	Maj@12	Avg@12	Mean Answer Length
GRPO	76.67 (s=40)	26.67 (s=50)	19.72 (s=30)	**1681.49 (s=50)**
REFSOL	86.67 (s=60)	43.33 (s=60)	30.56 (s=40)	1935.83 (s=10)
STEPALIGNFB	**90.00 (s=60)**	**56.67 (s=50)**	**35.83 (s=50)**	1996.07 (s=10)

Observations. We see that using step-aligned feedback as context outperforms using the reference solution. Despite never seeing the ground-truth derivation, STEPALIGNFB outperforms REFSOL on the Avg@12 metric at the per-metric best values: +2.33 Pass@12, +13.33 Maj@12, and +5.27 Avg@12 (Table 1). The large Majority-Vote gain in particular suggests STEPALIGNFB's policy concentrates probability on correct answers more sharply, rather than merely covering them, which is the regime that benefits most from test-time aggregation.

As with other post-training methods [Stollenwerk, 2022, Zhao et al., 2026], self-distillation benefits from early stopping. We trained the model for 7 epochs but found that 5–6 sufficed to reach peak performance across all metrics. This is promising for compute efficiency, however carries a methodological caveat: a fixed end-of-run evaluation can systematically understate self-distillation's ceiling, so per-checkpoint selection on a held-out validation set is necessary for a fair comparison.

Finally, both distillation methods dominate GRPO throughout training at all fronts except answer length. While GRPO maintains token efficiency, both REFSOL and STEPALIGNFB stay above GRPO in terms of accuracy throughout training, with a final Avg@12 gap on the order of 8 points.

4.2 Token-Level Credit Assignment

In this section, we identify the mechanism by which STEPALIGNFB outperforms REFSOL. The self-distillation advantages (Eq. 4) tell a clear story: step-aligned feedback causes the self-distillation signal to behave like a process reward model, reinforcing correct steps in the solver's trace while suppressing erroneous ones. Feeding the reference solution as feedback, by contrast, suppresses even fully correct solver trajectories. We present sample problems, student and reference solutions, critiques, and the resulting per-token advantage plots in Appendix B.

STEPALIGNFB concentrates signal at errors and reinforces correct steps. At incorrect steps, the self-teacher diverges sharply from the student, producing large negative advantages (Figure 4). At correct steps, including those preceding the error, the critic faithfully preserves the solver's reasoning path. The result is a targeted learning signal that penalizes errors while reinforces the valid trace that surrounds them. This is analogous to a process reward model, but derived entirely from natural-language feedback, with no reward model training required.

Solution-level feedback produces diffuse suppressive signal. Under REFSOL, the self-teacher sees a complete alternative derivation: one that reaches the right answer but follows a different reasoning path, uses different notation, and phrases steps differently. Even at steps where the solver was correct, the alternative solution diverges in surface form, creating diffuse negative advantages at almost all tokens (see Figures 3 and 4). The self-teacher thus mixes genuine error correction with stylistic disagreement, diluting the learning signal.

Connection to process vs. outcome supervision. Lightman et al. [2024] and Uesato et al. [2022] show that per-step scalar rewards outperform per-problem rewards for math reasoning. Our finding is a distributional generalization: step-aligned textual feedback provides richer-than-scalar process supervision, and the self-distillation mechanism converts it into per-token credit, all without training a reward model.

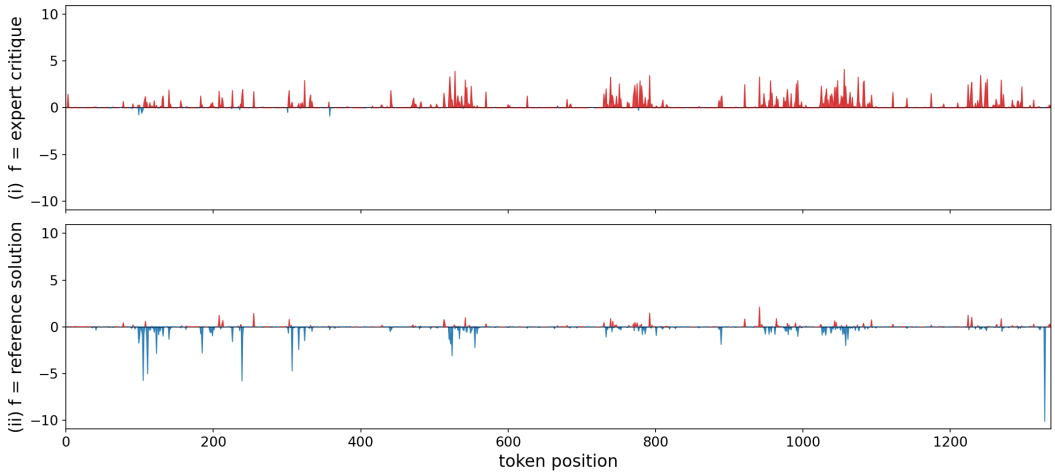


Figure 3: **Fully-correct rollout.** Per-token advantages $A_t^{\mathrm{SD}}(\hat{y}_t)$ (see Eq. 4) along a single student trajectory that reaches the correct boxed answer. STEPALIGNFB (top) produces positive advantages throughout the trace, reinforcing the student's correct reasoning trace. REFSOL (bottom) produces diffuse negative advantages across the entire rollout: the teacher prefers the alternative derivation's surface form even though the student's answer is correct, mixing stylistic disagreement into the gradient.

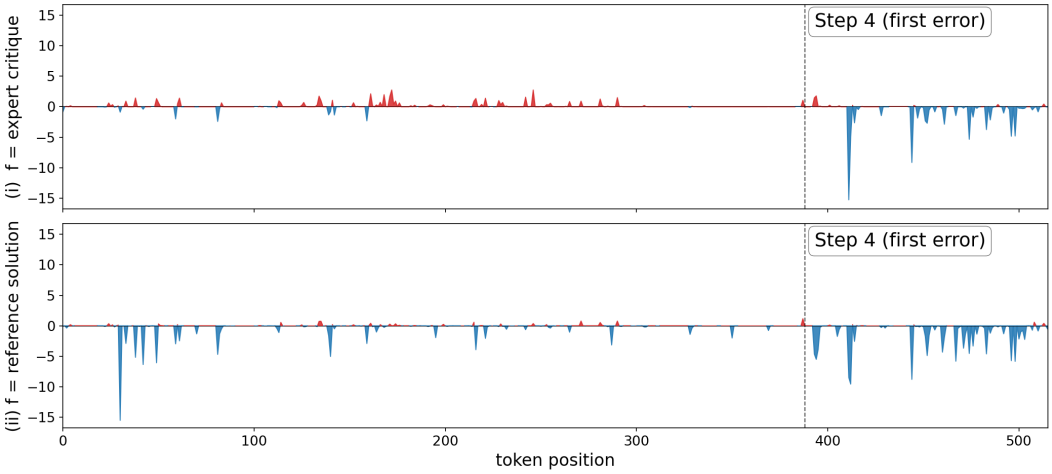


Figure 4: **Incorrect step.** Per-token advantages for a rollout containing an error at the step marked by the dashed line. STEPALIGNFB (top) produces a sharp negative shift at the tokens after the erroneous step, while reinforcing correct reasoning in the preceding steps. REFSOL (bottom) produces broadly negative advantages across the entire trace, conflating the error with everywhere the student's path diverges from the canonical solution.

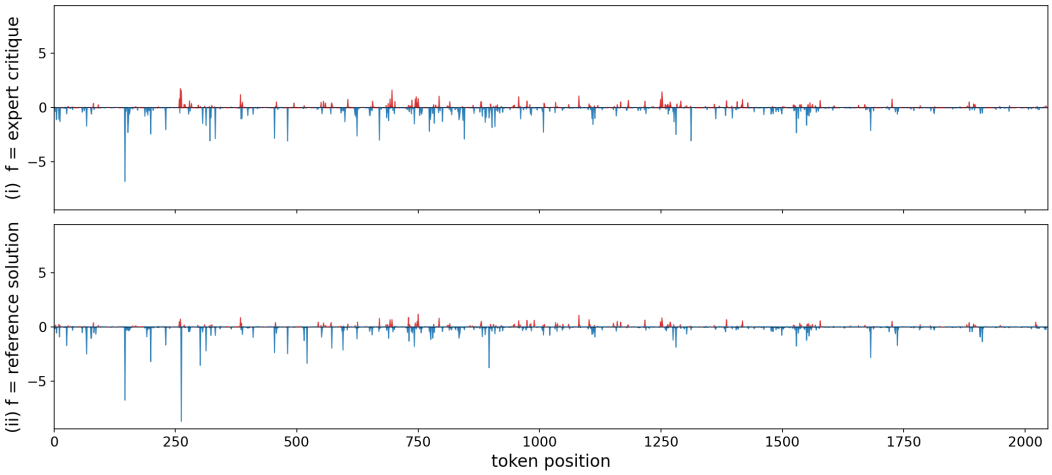


Figure 5: **Incomplete solution.** Per-token advantages for a rollout that exhausted the token budget without producing a final answer. The student's reasoning trace is unpromising from the start and quickly devolves into guessing values for the variable before running out of budget. The critic responds by producing a complete alternative solution, yielding update signals similar to those of the reference solution.

4.3 Verbatim Repetition and the Role of Induction-Head Copying

Our experiments showed that PRM-like credit assignment requires careful design of what the critic includes in its feedback. How much of the solver's output the critic repeats verbatim determines the quality of the advantage signal. We identify three regimes in the advantage plots. These led us to the "faithful-scribe convention": the critic repeats correct steps verbatim and rewrites only the incorrect ones (Hard Rules 3 and 4 in Appendix A.1).

Full verbatim repetition overrides the corrective signal. A natural first question is whether to expose the full original attempt in the feedback: *"Here is your previous attempt: `<previous_attempt>`. Step N is incorrect; the correct step should be `<corrected_step>`."* Consistent with Hübotter et al. [2026], we found this consistently fails. The verbatim quote, placed

earlier in the context, anchors the model at the erroneous step: the teacher reproduces tokens from the original (incorrect) attempt, and the per-token advantages are diffusely positive (Figure 6). The corrective text appended after the quote does not override the in-context copy, and the gradient do not see the error.

Omitting the solution suppresses correct reasoning. The dual failure mode arises when we don't provide any of the correct steps from the original attempt (verbatim). A natural shorthand is to acknowledge correctness without reproducing the trace, e.g. *'This step is correct."* However, without the student's tokens in the context, the self-teacher distribution can drift away from the student's at the positions the student got right. The per-token advantages on the correct trace come out diffusely negative (Figure 7), echoing the diffuse-suppression pattern REFSOL produces in Figure 3 despite the feedback endorsing the student.

Partial verbatim repetition up to the first error preserves both properties. The configuration that works is to repeat the student's solution verbatim only up to (but not including) the erroneous step, replace that step with the correct claim, and continue from there. The teacher's distribution then aligns with the student's tokens on the correct prefix (producing positive advantages), while the absence of an in-context anchor at the erroneous position lets the corrected continuation govern the teacher's predictions there (producing the sharp negative cluster). This is the pattern shown in Figure 4, and it is precisely the behavior our critic prompt is engineered to elicit.

A mechanistic hypothesis: induction-head copying. These three regimes mirror the prefix-matching-and-copying behavior of induction heads characterized by Olsson et al. [2022]. An induction head implements the pattern $[A][B] \ldots [A] \rightarrow [B]$: when a token sequence appears earlier in the context, attending back to it increases the logit of the token that followed it, *biasing the model toward repeating the in-context continuation*. Under this account, a full verbatim quote of the student's incorrect step installs the erroneous continuation as the in-context $[B]$; subsequent corrective text arrives too late and too disconnected from the matching prefix to override the copy. Partial repetition is the only configuration that uses copying *selectively* In effect, the faithful-scribe convention recruits induction-head copying as a tool for credit assignment rather than a source of error reinforcement.

Hübotter et al. [2026] report a closely related effect in a different setting. Ablating whether the student's original attempt y is included in the teacher template, they find that "including it biases the teacher towards the student's attempt … thereby reducing exploration" (their Table 6), and that "naively including only solutions or initial attempts y significantly reduces diversity in the teacher and student." Our findings are consistent with this and we hypothesize that in-context copying via induction heads is the underlying mechanism. The practical implication, however, is sharper than "do not repeat the attempt." Repeating the *correct prefix* while withholding the erroneous step turns the copying bias into a mechanism that concentrates the self-distillation signal at errors.

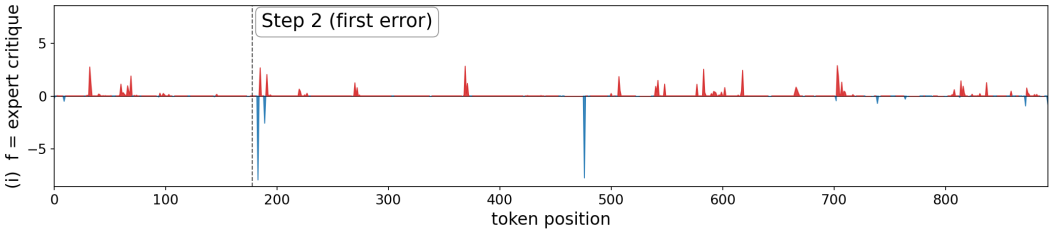


Figure 6: **Full verbatim repetition with appended correction.** The critic includes the student's incorrect step verbatim and follows it with a textual correction. Induction-head copying of the original tokens dominates the teacher's distribution; per-token advantages are overwhelmingly positive.

5 Related Work

Self-distillation for LLMs. Main references for self-distillation are Hübotter et al. [2026], Shenfeld et al. [2026], Song et al. [2026], Zhao et al. [2026]. Hübotter et al. [2026] introduced SDPO and used

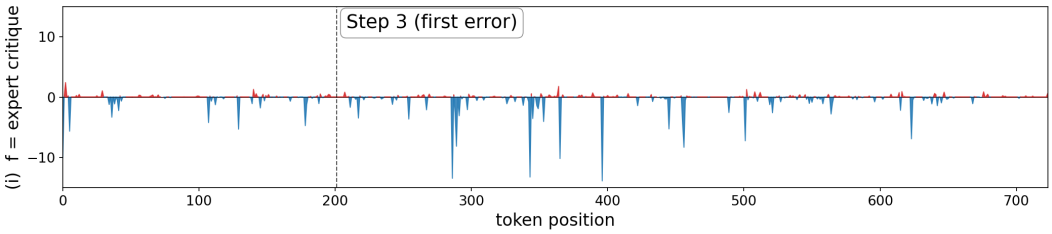


Figure 7: **Solution omitted from the teacher context.** For a fully correct student solution, the critic acknowledges correctness without reproducing the trace. With no in-context anchor on the student's tokens, the teacher's distribution drifts toward alternative surface forms, producing negative advantages across the correct trajectory.

code execution trace, reference solutions from datasets, and stronger models' solutions as feedback. OPSD Zhao et al. [2026] used ground-truth solutions as teacher context. RLTF [Song et al., 2026] formalized learning from text feedback using both self-distillation and feedback-modeling objectives, though they re-sample in-context rollouts from the solver. SDFT [Shenfeld et al., 2026] applied self-distillation to continual learning from demonstrations.

Process vs. outcome supervision. PRMs [Lightman et al., 2024] and process-based feedback [Uesato et al., 2022] assign per-step scalar rewards and consistently outperform outcome supervision [Cobbe et al., 2021] for math. The advantage is twofold: step-level signals localize errors to the exact reasoning step where they occur and they discourage models from reaching correct answers via flawed intermediate steps. To avoid the cost of human step-level annotation, follow-up work has automated the labeling process [Wang et al., 2024]. PRMs are widely used both as verifiers for best-of-N selection and as dense reward signals for RL.

Multi-agent debate and LLM-as-judge. Multi-agent debate [Du et al., 2024] and LLM-as-judge [Zheng et al., 2023] introduced inter-model interaction for improving reasoning at inference time. Since then, LLM-generated judgments have been widely adopted as training signals [Yuan et al., 2024, Chan et al., 2023, Zhao et al., 2025]. Our solver–critic setup follows this paradigm.

6 Conclusion

We demonstrate that the structure of feedback is a central determinant of self-distillation effectiveness. Among the three training signals we study, step-aligned feedback (StepAlignFB) consistently performs best on all accuracy metrics, outperforming both sparse binary reward (used in GRPO) and self-distillation with reference solution (RefSol). This comparison highlights a clear trade-off. Firstly, as also observed by prior work, GRPO is simple and inexpensive, but its scalar reward provides only weak credit assignment over long reasoning traces. Secondly, RefSol offers richer supervision than GRPO, yet because it is not explicitly aligned to the solver's intermediate steps, its learning signal remains diffuse. By contrast, StepAlignFB preserves the token-level benefits of self-distillation while adding structural alignment to the solver's reasoning process, yielding an implicit form of process supervision that concentrates updates around error-prone parts of the trace. This stronger signal comes with an important practical drawback: producing high-quality step-aligned critiques requires a capable critique model, which can substantially increase training cost. Reference solutions with chain-of-thought are also commonly sampled from strong models. Nonetheless, unlike solution-specific feedback, the same reference solution can be reused across different training runs.

Our findings suggest that the main advantage of self-distillation is not merely access to additional context, but access to context whose structure matches the reasoning trajectory the student must learn to correct. More broadly, they point to feedback design as a key optimization dimension for post-training beyond the usual choice between reinforcement learning and distillation. Our study, however, is limited in scope: all experiments are conducted on the OpenMathReasoning dataset, so it remains unclear how well these conclusions transfer to other benchmarks and broader reasoning domains. Future work should therefore test whether the benefits of step-aligned critique persist across datasets, model sizes, and tasks, and whether similarly effective but cheaper critique-generation strategies can narrow the cost gap that currently limits the practicality of StepAlignFB.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. *arXiv preprint arXiv:2306.13649*, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Thomas Kleine Buening, Jonas Hübotter, Barna Pásztor, Idan Shenfeld, Giorgia Ramponi, and Andreas Krause. Aligning language models from user interactions. *arXiv preprint arXiv:2603.12273*, 2026.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023.

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

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *arXiv preprint arXiv:2205.14135*, 2022.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *ICML*, 2024.

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

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

Jonas Hübotter, Idan Shenfeld, Andreas Krause, and Bernhard Schölkopf. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

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

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. `https://huggingface.co/AI-MO/NuminaMath-CoT`, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2019.

Kevin Lu and Thinking Machines. On-policy distillation, 2025. URL `https://thinkingmachines.ai/blog/on-policy-distillation/`. Blog post.

Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

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.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

Yuda Song, Lili Chen, Fahim Tajwar, Remi Munos, Deepak Pathak, J. Andrew Bagnell, Aarti Singh, and Andrea Zanette. Expanding the capabilities of reinforcement learning via text feedback. *arXiv preprint arXiv:2602.02482*, 2026.

Felix Stollenwerk. Adaptive fine-tuning of transformer-based language models for named entity recognition. *arXiv preprint arXiv:2202.02617*, 2022.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

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

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. Openclaw-rl: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026.

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, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *ICML*, 2024.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.

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*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *NeurIPS*, 2023.

A Experiment details

A.1 Prompt templates

Solver Prompt

```
Problem:
{problem}

Solve the problem step by step. Format each step as:

### Step N: <topic>
<reasoning for step N>

Please reason step by step, and put your final answer within \boxed{}.
```

Figure 8: **Solver prompt.** The solver sees only the problem statement, with the same step-formatted instruction used by the evaluation harness so that on-policy training and inference share a prompt template.

Teacher Prompt — RefSol (reference-solution conditioning)

```
Question: {problem}

A reference solution is given below. Use it only to ensure correctness.

Reference solution:
{reference_solution}

Instructions:
- Produce a fresh, self-contained solution to the original problem.
- Use the reference solution only to ensure correctness; do not mention or refer to it.

Let's think step by step and produce a final answer in the format \boxed{}.
```

Figure 9: **Teacher prompt for RefSol.** The reference solution provided in the dataset is the privileged context.

Teacher Prompt — StepAlignFB (step-aligned feedback conditioning)

```
Question: {problem}

Expert feedback on a prior attempt at this problem is given below. The feedback diagnoses
     where the attempt went wrong (by step number) and carries the corrected continuation.

Expert feedback:
{expert_critique}

Instructions:
- Produce a fresh, self-contained solution to the original problem.
- Use the feedback only to ensure correctness; do not mention or refer to it.

Let's think step by step and produce a final answer in the format \boxed{}.
```

Figure 10: **Teacher prompt for StepAlignFB.** The reference solution is replaced by a critic-generated critique f produced by Qwen/QwQ-32B (Figure 11).

Critic Prompt (Qwen/QwQ-32B)

```
You are a math grader producing feedback to a student's solution.
    Your default behavior is faithful scribe: when the student's work is correct up to
     some point, reproduce that portion. The exceptions:
    - In Case D, at the erroneous step only, replace the student's claim with the correct
     one.
    - In Case C, you may compress the student's correct steps because the student ran out
     of room -- this exception applies ONLY to Case C and nowhere else.

    # Decision procedure

    **Step 1: Classify into ONE case.** Check in order, stop at first match:
    - No final answer (no \\boxed{{}}, trails off, stops mid-derivation) -> Case C
    - Boxed answer doesn't match reference (allow equivalent forms: 1/2 == 0.5) -> Case D
    - Boxed answer matches but a non-routine (non-routine = named theorem, substitution,
     key equation, or non-algebraic claim) step is unjustified or invalid -> Case B
    - Else -> Case A

    **Step 2: Identify the pivotal step N.**
    - B: step with the unjustified claim.
    - C: last step before stopping.
    - D: earliest erroneous step (not inherited errors).

    **Step 3: Output the matching schema below. No preamble, no postamble, no commentary
     outside the schema.**

    # Schemas

    Every schema begins with a '### Summary' block giving a one-line verdict per student
     step. The body that follows uses headers and '---' delimiters to make the structure
     of the feedback explicit.

    ## Case A

    ### Summary
    Step 1: Correct.
    Step 2: Correct.
    ...
    Step <final>: Correct.

    # Your solution is fully correct.
    ---
    <Reproduce the student's solution, including the \\boxed{{}} answer.>
    ---

    ## Case B

    ### Summary
    Step 1: Correct.
    Step 2: Correct.
    ...
    Step <N>: Correct, but missing justification -- <one phrase naming what's missing>.
    Step <N+1>: Correct.
    ...
    Step <final>: Correct.

    # Your solution reaches the correct answer, but Step <N> is missing justification.
    # This is the part of your solution before the gap:
    ---
    <Student's steps 1 through N-1, reproduced.>
    ---
    # Step <N> needs the following justification added:
    ---
    <Student's step N copied, with the missing justification -- named theorem, formula, or
      equation -- appended.>
```

```
---
# The remainder of your solution is correct:
---
<Student's steps N+1 through final, reproduced, including the \\boxed{{}} answer.>
---

## Case C

### Summary
Step 1: Correct.
Step 2: Correct.
...
Step <N>: Correct, but stopped here without producing a final answer.

# Your solution was correct up to Step <N> but ran out of room before finishing. Below
  is a condensed version of your work followed by the completion.
# This is the correct part of your solution, condensed:
---
<Student's steps 1 through N, condensed.>
---
# Here is the rest of the solution:
---
<Continue the derivation in the student's notation, concisely. One operation per step
 .>
---
# Final answer
\\boxed{{<answer>}}

## Case D

### Summary
Step 1: Correct.
Step 2: Correct.
...
Step <N>: Incorrect -- <one phrase naming the error>.
Step <N+1>: <verdict>.
...
Step <final>: <verdict>.

# Your solution has an error at Step <N>. Below is your correct work, the corrected
 step, and the remainder of the solution.
# This is the correct part of your solution:
---
<Student's steps 1 through N-1, reproduced.>
---
# You made an error at Step <N>. Here is the corrected step:
---
Step <N>: <Write the correct claim in the student's style -- same notation, same step
 granularity, same level of detail as the surrounding correct steps. Do NOT reproduce
 the student's incorrect step.>
---
# Continuing the solution from the corrected step:
---
<Continue the derivation correctly from the corrected step N, in the student's
 notation and style. Include the final \\boxed{{}} answer.>
---

# Hard rules

1. Output ONE schema only. Every schema starts with '### Summary'. No preamble before
 '### Summary', no postamble after the final delimiter.
2. The '### Summary' block contains one line per student step in the format 'Step <i>:
  <verdict>' where verdict is "Correct" or "Incorrect -- <one phrase>" or a similar
 terse marker.
```

```
3. In Cases A, B, and D, your reproduction of the student's correct steps should
 follow same or similar equation, notations, wording. If the student's reasoning trace
  correct, do not deviate from the student's solution.
4. In Case D, at step N only, replace the student's incorrect claim with the correct
 claim in the student's style. Correct steps before and after are copied exactly per
 rule 3.
5. Every hypothesis in the problem must appear in the steps at a named step.
6. No advisory phrases ("be more rigorous", "needs justification") unless immediately
 followed by the specific content.
7. Equivalent approaches reaching the same answer are CORRECT. The reference is an
 answer key, not a required path.
8. In Case D, continuations after step N match the student's style: same notation,
 same step granularity, same level of formality. In Case C, continuations are written
 concisely -- one operation per step, no exploration.
9. Don't invent formulas absent from the reference or not derivable from the problem.

Problem:
{problem}

Reference answer:
{reference_solution}

Student's solution:
{student_sol}
```

Figure 11: **Critic prompt.** A single user-turn message sent to a frozen Qwen/QwQ-32B. The grader is instructed to classify the student rollout into one of four cases (A: fully correct; B: correct answer with an unjustified non-routine step; C: ran out of tokens; D: incorrect answer) and emit feedback in a schema specific to that case. The returned reasoning trace (`<think>...</think>`) is stripped before the structured grader output is spliced into the final feedback f for the STEPALIGNFB teacher prompt (Figure 10). Unicode characters in the deployed prompt have been transliterated here for printability.

A.2 Training configuration (full)

This appendix lists complete hyperparameters for the self-distillation (with reference solution and) and GRPO runs reported in the main paper. See Tables 2, 3, 4, and 5 for the values.

B Sample solution, critique, and self-distillation advantage plots

B.1 Example correct student solution

Problem

```
Let $\mathbb{Z}_{>0}$ be the set of positive integers. Find all functions $f :
     \mathbb{Z}_{>0} \rightarrow \mathbb{Z}_{>0}$ such that
$$f(2m+2n)=f(m)^{2021}f(n)^{2021}$$
for all $m,n \in \mathbb{Z}_{>0}$.
```

Reference Solution

```
To solve the functional equation \( f(2m + 2n) = f(m)^{2021} f(n)^{2021} \) for all
     positive integers \( m \) and \( n \), we proceed as follows:

1. **Initial Substitution and Simplification:**
   - Set \( m = n \). Then the equation becomes:
     \[
     f(4m) = f(m)^{4042}
     \]
   - This indicates that \( f \) at even arguments can be expressed in terms of \( f \)
     at smaller arguments.
```

```
2. **General Form Analysis:**
   - For any positive integers \( m \) and \( n \), the argument \( 2m + 2n = 2(m + n)
     \). Thus, \( f(2(m + n)) = f(m)^{2021} f(n)^{2021} \).
   - Consider \( m = 1 \) and vary \( n \):
     - For \( n = 1 \):
       \[
       f(4) = f(1)^{4042}
       \]
     - For \( n = 2 \):
       \[
       f(6) = f(1)^{2021} f(2)^{2021}
       \]

3. **Constant Function Hypothesis:**
   - Suppose \( f(k) = c \) for all \( k \). Substitute into the original equation:
     \[
     c = c^{4042}
     \]
   - The only positive integer solution to this equation is \( c = 1 \).

4. **Verification:**
   - Verify that \( f(k) = 1 \) satisfies the original equation:
     \[
     f(2m + 2n) = 1 \quad \text{and} \quad f(m)^{2021} f(n)^{2021} = 1^{2021} \cdot
     1^{2021} = 1
     \]
   - Both sides are equal, confirming that \( f(k) = 1 \) is a solution.

5. **Uniqueness:**
   - To show that \( f(k) = 1 \) is the only solution, consider the necessity for \(
     f(m) f(n) \) to be constant for each \( t \geq 2 \) where \( m + n = t \). This
     implies \( f \) must be constant.
   - If \( f \) were not constant, the recursive relationships derived would lead to
     inconsistencies, as shown by the recursive nature of the equation.

Thus, the only function that satisfies the given equation is:

\[
\boxed{1}
\]
```

Student Solution

```
We are given the functional equation:

$$
f(2m + 2n) = f(m)^{2021} f(n)^{2021}
$$

for all $ m, n \in \mathbb{Z}_{>0} $, and we are to find all functions $ f :
     \mathbb{Z}_{>0} \to \mathbb{Z}_{>0} $ satisfying this.

---

### Step 1: Try simple functions

Let's try simple functions like constant functions, linear functions, or exponential
     functions.

#### Try: $ f(n) = c $, a constant.

If $ f(n) = c $, then:

$$
f(2m + 2n) = c = c^{2021} \cdot c^{2021} = c^{4042}
$$
```

```
This implies $ c = c^{4042} $, which is only true if $ c = 1 $, since $ c \in
     \mathbb{Z}_{>0} $.

So $ f(n) = 1 $ is a possible solution.

Check:

$$
f(2m + 2n) = 1, \quad f(m)^{2021} f(n)^{2021} = 1^{2021} \cdot 1^{2021} = 1
$$

So this satisfies the equation.

#### Try: $ f(n) = n^k $ for some exponent $ k \in \mathbb{Z} $.

Then:

$$
f(2m + 2n) = (2m + 2n)^k = (2(m + n))^k = 2^k (m + n)^k
$$

On the other hand:

$$
f(m)^{2021} f(n)^{2021} = (m^k)^{2021} (n^k)^{2021} = (mn)^{2021k}
$$

So the equation becomes:

$$
2^k (m + n)^k = (mn)^{2021k}
$$

This is not generally true for all $ m, n $. So $ f(n) = n^k $ is not a solution unless
     $ k = 0 $, which gives $ f(n) = 1 $.

---

### Step 2: Consider functional equations with exponents

We notice that the equation has a multiplicative structure. Let's try to analyze the
     form of $ f $.

Define:

$$
f(n) = g(n)^{2021}
$$

Then:

$$
f(2m + 2n) = g(2m + 2n)^{2021}
$$

On the other hand:

$$
f(m)^{2021} f(n)^{2021} = [g(m)^{2021}]^{2021} [g(n)^{2021}]^{2021} = g(m)^{2021^2}
     g(n)^{2021^2}
$$

So the equation becomes:

$$
```

```
g(2m + 2n)^{2021} = g(m)^{2021^2} g(n)^{2021^2}
$$

This is a functional equation for $ g $. It's similar in form to the original one, but
     with the exponent $ 2021^2 $ instead of $ 2021 $. So the structure is consistent.

We can iterate this and try to find a function $ g $ such that this holds.

---

### Step 3: Try $ g(n) = 1 $ for all $ n $

Then:

$$
g(2m + 2n) = 1, \quad g(m)^{2021^2} g(n)^{2021^2} = 1^{2021^2} \cdot 1^{2021^2} = 1
$$

So this works. Then:

$$
f(n) = g(n)^{2021} = 1^{2021} = 1
$$

Thus, $ f(n) = 1 $ is a solution.

---

### Step 4: Uniqueness

Suppose there is another function $ f $ satisfying the condition.

Suppose $ f(n) \neq 1 $ for some $ n $.

Let's try $ f(1) \neq 1 $. Then we can define $ f(2m + 2n) = f(m)^{2021} f(n)^{2021} $,
     which is a multiplicative function in $ m, n $, but not necessarily additive.

If we consider the form of the function, we can try to look for functions that are
     multiplicative or power functions.

Suppose $ f(n) = c^{\alpha(n)} $, with some function $ \alpha(n) $.

Then the equation becomes:

$$
f(2m + 2n) = c^{\alpha(2m + 2n)} = f(m)^{2021} f(n)^{2021} = (c^{\alpha(m)})^{2021}
     (c^{\alpha(n)})^{2021} = c^{2021\alpha(m) + 2021\alpha(n)}
$$

So:

$$
\alpha(2m + 2n) = 2021(\alpha(m) + \alpha(n))
$$

This is a recursive functional equation. It implies that $ \alpha(n) $ must be linear in
     the binary representation of $ n $. This is possible only if $ \alpha(n) = k \cdot
     \log_2 n $, which is not in $ \mathbb{Z}_{>0} $ unless $ k = 0 $.

So the only solution is:

$$
\alpha(n) = 0 \Rightarrow f(n) = 1
$$

---
```

```
### Final Answer:

$$
\boxed{f(n) = 1}
$$<|im_end|>
```

Step-level Critique

```
### Summary
Step 1: Correct.
Step 2: Correct.
Step 3: Correct.
Step 4: Correct.
Step 5: Correct.

# Your solution is fully correct.
---
We are given the functional equation:

$$
f(2m + 2n) = f(m)^{2021} f(n)^{2021}
$$

for all $ m, n \in \mathbb{Z}_{>0} $, and we are to find all functions $ f :
     \mathbb{Z}_{>0} \to \mathbb{Z}_{>0} $ satisfying this.

---

### Step 1: Try simple functions

Let's try simple functions like constant functions, linear functions, or exponential
     functions.

#### Try: $ f(n) = c $, a constant.

If $ f(n) = c $, then:

$$
f(2m + 2n) = c = c^{2021} \cdot c^{2021} = c^{4042}
$$

This implies $ c = c^{4042} $, which is only true if $ c = 1 $, since $ c \in
     \mathbb{Z}_{>0} $.

So $ f(n) = 1 $ is a possible solution.

Check:

$$
f(2m + 2n) = 1, \quad f(m)^{2021} f(n)^{2021} = 1^{2021} \cdot 1^{2021} = 1
$$

So this satisfies the equation.

#### Try: $ f(n) = n^k $ for some exponent $ k \in \mathbb{Z} $.

Then:

$$
f(2m + 2n) = (2m + 2n)^k = (2(m + n))^k = 2^k (m + n)^k
$$

On the other hand:

$$
```

```
f(m)^{2021} f(n)^{2021} = (m^k)^{2021} (n^k)^{2021} = (mn)^{2021k}
$$

So the equation becomes:

$$
2^k (m + n)^k = (mn)^{2021k}
$$

This is not generally true for all $ m, n $. So $ f(n) = n^k $ is not a solution unless
    $ k = 0 $, which gives $ f(n) = 1 $.

---

### Step 2: Consider functional equations with exponents

We notice that the equation has a multiplicative structure. Let's try to analyze the
    form of $ f $.

Define:

$$
f(n) = g(n)^{2021}
$$

Then:

$$
f(2m + 2n) = g(2m + 2n)^{2021}
$$

On the other hand:

$$
f(m)^{2021} f(n)^{2021} = [g(m)^{2021}]^{2021} [g(n)^{2021}]^{2021} = g(m)^{2021^2}
    g(n)^{2021^2}
$$

So the equation becomes:

$$
g(2m + 2n)^{2021} = g(m)^{2021^2} g(n)^{2021^2}
$$

This is a functional equation for $ g $. It's similar in form to the original one, but
    with the exponent $ 2021^2 $ instead of $ 2021 $. So the structure is consistent.

We can iterate this and try to find a function $ g $ such that this holds.

---

### Step 3: Try $ g(n) = 1 $ for all $ n $

Then:

$$
g(2m + 2n) = 1, \quad g(m)^{2021^2} g(n)^{2021^2} = 1^{2021^2} \cdot 1^{2021^2} = 1
$$

So this works. Then:

$$
f(n) = g(n)^{2021} = 1^{2021} = 1
$$

Thus, $ f(n) = 1 $ is a solution.
```

```
---

### Step 4: Uniqueness

Suppose there is another function $ f $ satisfying the condition.

Suppose $ f(n) \neq 1 $ for some $ n $.

Let's try $ f(1) \neq 1 $. Then we can define $ f(2m + 2n) = f(m)^{2021} f(n)^{2021} $,
     which is a multiplicative function in $ m, n $, but not necessarily additive.

If we consider the form of the function, we can try to look for functions that are
     multiplicative or power functions.

Suppose $ f(n) = c^{\alpha(n)} $, with some function $ \alpha(n) $.

Then the equation becomes:

$$
f(2m + 2n) = c^{\alpha(2m + 2n)} = f(m)^{2021} f(n)^{2021} = (c^{\alpha(m)})^{2021}
     (c^{\alpha(n)})^{2021} = c^{2021\alpha(m) + 2021\alpha(n)}
$$

So:

$$
\alpha(2m + 2n) = 2021(\alpha(m) + \alpha(n))
$$

This is a recursive functional equation. It implies that $ \alpha(n) $ must be linear in
     the binary representation of $ n $. This is possible only if $ \alpha(n) = k \cdot
     \log_2 n $, which is not in $ \mathbb{Z}_{>0} $ unless $ k = 0 $.

So the only solution is:

$$
\alpha(n) = 0 \Rightarrow f(n) = 1
$$

---

### Final Answer:

$$
\boxed{f(n) = 1}
$$
```

B.2 Example incorrect student solution

Problem

```
How many solutions are there to the equation \(2x + 3y = 763\)?
```

Reference Solution

```
To determine the number of non-negative integer solutions to the equation \(2x + 3y =
     763\), we proceed as follows:

1. **Check for the existence of solutions**: The greatest common divisor of 2 and 3 is
     1, and since 1 divides 763, there exist integer solutions to the equation.

2. **Express \(x\) in terms of \(y\)**:
   \[
```

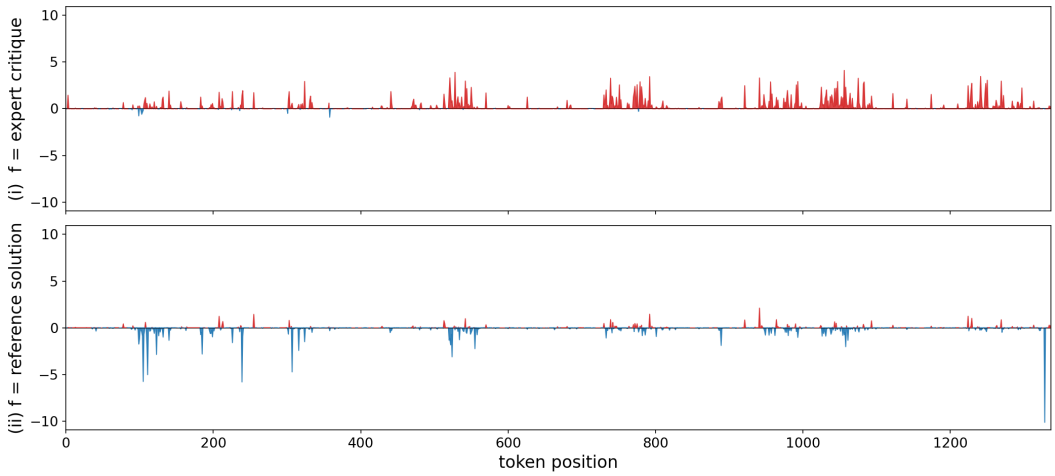


Figure 12: Per-token advantage under each teacher context for correct student solution.

```
   x = \frac{763 - 3y}{2}
   \]
   For \(x\) to be an integer, \(763 - 3y\) must be even. Since 763 is odd, \(3y\) must
     also be odd, implying that \(y\) must be odd. Let \(y = 2k + 1\) where \(k\) is a
     non-negative integer.

3. **Substitute \(y = 2k + 1\) into the equation**:
   \[
   x = \frac{763 - 3(2k + 1)}{2} = \frac{763 - 6k - 3}{2} = \frac{760 - 6k}{2} = 380 - 3k
   \]

4. **Ensure \(x\) is non-negative**:
   \[
   380 - 3k \geq 0 \implies 380 \geq 3k \implies k \leq \frac{380}{3} \approx 126.67
   \]
   Since \(k\) must be an integer, \(k\) ranges from 0 to 126, inclusive.

5. **Count the number of possible values for \(k\)**:
   The number of integers from 0 to 126 is:
   \[
   126 - 0 + 1 = 127
   \]

Therefore, the number of non-negative integer solutions to the equation \(2x + 3y =
     763\) is \(\boxed{127}\).
```

Student Solution

```
We are given the equation:

$$
2x + 3y = 763
$$

We are to find the number of **integer solutions** $(x, y)$ that satisfy this equation.

---

### Step 1: Express $x$ in terms of $y$

We solve the equation for $x$:

$$
2x = 763 - 3y
```

```
$$

$$
x = \frac{763 - 3y}{2}
$$

For $x$ to be an integer, the expression $763 - 3y$ must be even.

---

### Step 2: Determine conditions on $y$

We require:

$$
763 - 3y \text{ is even}
$$

Let's consider the parity (even or odd nature) of the numbers involved:

- $763$ is **odd**
- $3y$ is **odd** if $y$ is **odd**, and **even** if $y$ is **even**

So, odd minus odd = even
odd minus even = odd

Therefore, for $763 - 3y$ to be **even**, $3y$ must be **odd**, which means $y$ must be
     **odd**.

---

### Step 3: Let $y = 2k + 1$, where $k$ is an integer

Substitute $y = 2k + 1$ into the equation:

$$
x = \frac{763 - 3(2k + 1)}{2}
$$

$$
x = \frac{763 - 6k - 3}{2}
$$

$$
x = \frac{760 - 6k}{2}
$$

$$
x = 380 - 3k
$$

---

### Step 4: Find all integer values of $k$ such that $x$ and $y$ are integers

We have:

$$
x = 380 - 3k
$$
$$
y = 2k + 1
$$

There are no restrictions on $k$, so $k$ can be any integer.
```

```
---

### Step 5: Count the number of integer solutions

Since $k$ can be any integer, there are **infinitely many integer solutions** to the
     equation.

---

### Final Answer:

$$
\boxed{\text{Infinitely many}}
$$<|im_end|>
```

Step-level Critique

```
### Summary
Step 1: Correct.
Step 2: Correct.
Step 3: Correct.
Step 4: Incorrect -- failed to consider non-negative constraints on x and y.
Step 5: Correct.

# Your solution has an error at Step 4. Below is your correct work, the corrected step,
     and the remainder of the solution.
# This is the correct part of your solution:
---
We are given the equation:

$$
2x + 3y = 763
$$

We are to find the number of **integer solutions** $(x, y)$ that satisfy this equation.

---

### Step 1: Express $x$ in terms of $y$

We solve the equation for $x$:

$$
2x = 763 - 3y
$$

$$
x = \frac{763 - 3y}{2}
$$

For $x$ to be an integer, the expression $763 - 3y$ must be even.

---

### Step 2: Determine conditions on $y$

We require:

$$
763 - 3y \text{ is even}
$$

Let's consider the parity (even or odd nature) of the numbers involved:

- $763$ is **odd**
- $3y$ is **odd** if $y$ is **odd**, and **even** if $y$ is **even**
```

```
So, odd minus odd = even
odd minus even = odd

Therefore, for $763 - 3y$ to be **even**, $3y$ must be **odd**, which means $y$ must be
     **odd**.

---

### Step 3: Let $y = 2k + 1$, where $k$ is an integer

Substitute $y = 2k + 1$ into the equation:

$$
x = \frac{763 - 3(2k + 1)}{2}
$$

$$
x = \frac{763 - 6k - 3}{2}
$$

$$
x = \frac{760 - 6k}{2}
$$

$$
x = 380 - 3k
$$
---

# You made an error at Step 4. Here is the corrected step:
---
### Step 4: Find all integer values of $k$ such that $x$ and $y$ are non-negative
     integers

We need both \(x\) and \(y\) to be non-negative integers.

For \(x = 380 - 3k \geq 0\):
\[
380 - 3k \geq 0 \implies k \leq \frac{380}{3} \approx 126.67 \implies k \leq 126
\]

For \(y = 2k + 1 \geq 0\):
\[
2k + 1 \geq 0 \implies k \geq -\frac{1}{2}
\]
Since \(k\) is an integer, the smallest possible \(k\) is \(0\).

Thus, \(k\) must satisfy \(0 \leq k \leq 126\).

---

# Continuing the solution from the corrected step:
---
### Step 5: Count the number of integer solutions

The integer values of \(k\) range from \(0\) to \(126\) inclusive. The number of
     integers in this range is:
\[
126 - 0 + 1 = 127
\]

Therefore, the number of non-negative integer solutions to the equation \(2x + 3y =
     763\) is \(\boxed{127}\).
```

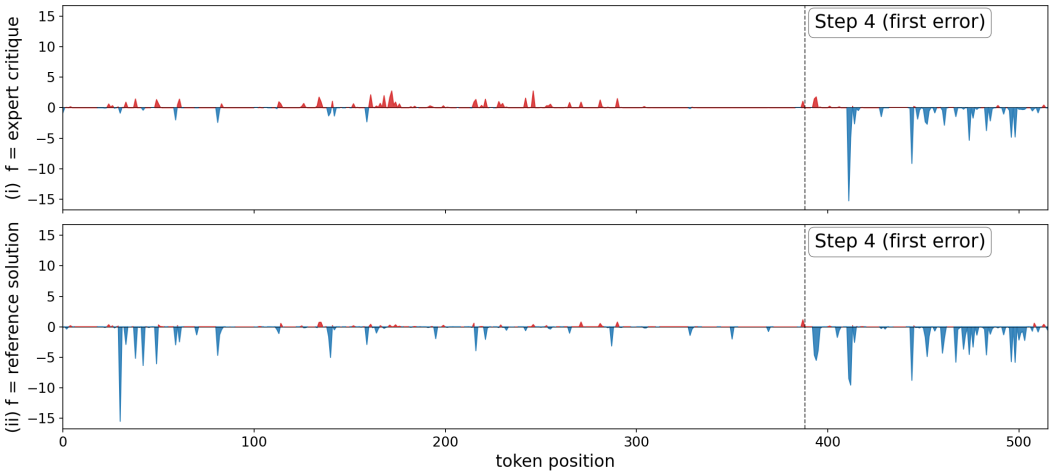


Figure 13: Per-token advantage under each teacher context for incorrect student solution.

Table 2: Self-distillation training configuration. The step-aligned feedback block lists the additional critic-server settings.

Parameter	**Value**
Model and adapters	
Base model	Qwen3-1.7B
Precision	bfloat16
Attention implementation	Flash Attention 2
LoRA rank r	64
LoRA α	128
LoRA dropout	0.0
LoRA target modules	`{q,k,v,o,gate,up,down}_proj`
Optimization	
Optimizer	AdamW
Learning rate	5×10^{-6}
LR schedule	constant
Weight decay	0.0
Gradient clipping (global)	0.1
Per-device train batch size	2
Gradient accumulation steps	4
GPUs (data-parallel)	4
Effective batch size	**32**
Gradient checkpointing	Enabled
DeepSpeed ZeRO stage	2 (CPU optimizer offload)
Epochs (max)	7
Save / eval interval	every 10 optimizer steps
Sequence lengths	
Max student completion $\lvert\hat{y}\rvert_{\max}$	2048
Max packed sequence length	20,000
On-policy sampling	
Sampling backend	vLLM (colocate)
Temperature	1.1
Top-p	0.95
Top-k	20
Rollouts per problem	1
Student thinking mode	OFF
Teacher thinking mode	ON
Distillation objective	
Objective	Full-vocabulary generalized JSD
β (forward KL)	0
Per-token KL clip τ	disabled
Teacher policy	Fixed = initial (LoRA-disabled) base
Teacher temperature	1.0
λ (student-mixing)	1
STEPALIGNFB *only: critic server*	
Critic model	Qwen/QwQ-32B (frozen)
Critic inference backend	vLLM (separate node, OpenAI-compat HTTP)
Critic sampling	greedy (T=0), top-p 0.95, top-k off
Critic max output tokens	8,000
Critic request timeout	240 s
Fallback on truncation	Enabled
Reasoning-trace handling	Strip everything up to and including `</think>`

Table 3: GRPO baseline configuration [Shao et al., 2024]. Effective batch size and learning rate are matched to self-distillation experiments (REFSOL and STEPALIGNFB); the larger generation cap and group size are the standard GRPO configuration.

Parameter	Value
Base model	Qwen3-1.7B
Learning rate	5×10^{-6}
Per-device batch size	1
Gradient accumulation	8
GPUs	4
Effective batch size	32
LoRA r / α	64 / 128
LoRA target modules	`{q,k,v,o,gate,up,down}_proj`
Max prompt length	2,048
Max completion length	8,000
Rollouts per prompt G	8
Sampling temperature	1.2
Loss type	`grpo`
Reward scaling	group-normalized
KL coefficient β	0
PPO inner iterations	2
Reward signal	binary, verified by `math_verify`

Table 4: Deltas of our setup from Zhao et al. [2026]. Items not listed are unchanged.

Item	Zhao et al.	Ours
Training corpus	OpenThoughts math subset	OpenMathReasoning (difficulty and format-filtered)
Teacher conditioning	Reference solution (REFSOL)	Critic feedback f (STEPALIGNFB)
Student rollout cap	1,024 tokens	2,048 tokens
Per-token KL clip τ	Enabled (untuned)	Disabled
Training budget	100 optimizer steps	Up to 7 epochs

Table 5: Evaluation configuration.

Parameter	Value
Inference backend	vLLM
Tensor parallel size	4
Samples per problem (n)	12
Temperature	1.0
Top-p	0.95
Top-k	disabled
Max new tokens	38,912
Thinking mode	Enabled
Benchmarks	Left-out 30 problem test split from difficulty and format-filtered OpenMathReasoning