Self-Distillation Zero: Self-Revision Turns Binary Rewards into Dense Supervision

Yinghui He[1] **Simran Kaur**[1] **Adithya Bhaskar**[1] **Yongjin Yang**[2] **Jiarui Liu**[3]
Narutatsu Ri[1] **Liam Fowl**[1] **Abhishek Panigrahi**[1] **Danqi Chen**[1] **Sanjeev Arora**[1]
[1]Princeton University [2]University of Toronto [3]Carnegie Mellon University
yh0068@princeton.edu

Abstract

Current post-training methods in verifiable settings fall into two categories. Reinforcement learning (RLVR) relies on binary rewards, which are broadly applicable and powerful, but provide only sparse supervision during training. Distillation provides dense token-level supervision, typically obtained from an external teacher or using high-quality demonstrations. Collecting such supervision can be costly or unavailable. We propose **Self-Distillation Zero** (SD-ZERO), a method that is substantially more training sample-efficient than RL and doesn't require an external teacher or high-quality demonstrations. SD-ZERO trains a *single model* to play two roles: a *Generator*, which produces an initial response, and a *Reviser*, which conditions on that response and its binary reward to produce an improved response. We then perform on-policy self-distillation to distill the reviser into the generator, using the reviser's token distributions conditioned on the generator's response and its reward as supervision. In effect, SD-ZERO trains the model to *transform binary rewards into dense token-level self-supervision.*

On math and code reasoning benchmarks with Qwen3-4B-Instruct and Olmo-3-7B-Instruct, SD-ZERO improves performance by at least 10% over the base models and outperforms strong baselines, including Rejection Fine-Tuning (RFT), GRPO, and Self-Distillation Fine-Tuning (SDFT), under the same question set and training sample budget. Extensive ablation studies show two novel characteristics of our proposed algorithm: (a) *token-level self-localization*, where the reviser can identify the key tokens that need to be revised in the generator's response based on reward, and (b) *iterative self-evolution*, where the improving ability to revise answers can be distilled back into generation performance with regular teacher synchronization.

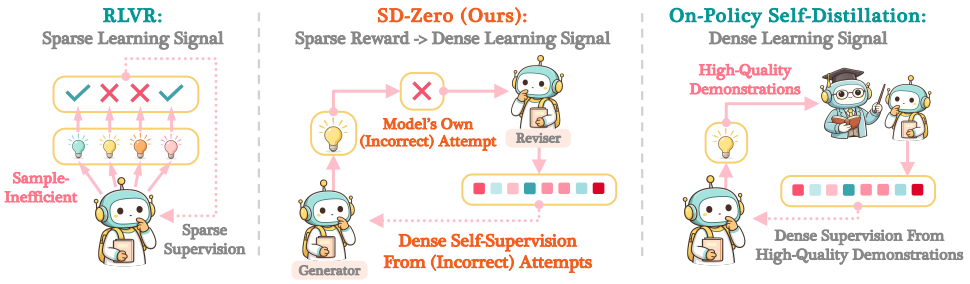

Figure 1: We introduce **SD-ZERO**, an on-policy self-distillation paradigm where the teacher (*reviser*) can condition on incorrect student (*generator*) attempts.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

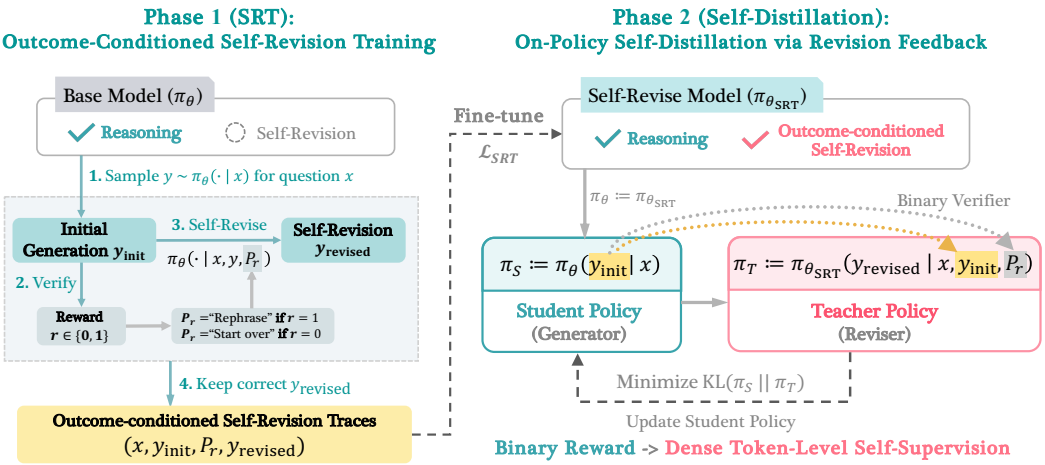

Figure 2: **Overview of SD-ZERO.** In **Phase 1 (SRT)**, we collect 6K outcome-conditioned self-revision traces by sampling an initial response from the base model, prompting the model to self-revise its incorrect response, and keeping the correct self-revision. In **Phase 2 (Self-Distillation)**, we conduct on-policy self-distillation with the self-revise (SRT) model acting as both student and teacher: the *student* generates an on-policy response, and the *teacher* generates a revised version conditioned on that response and its outcome reward. Throughout the Self-Distillation phase, the model bootstraps performance through internalizing self-revision behavior. See Section B.1 for details.

1 Introduction

Reinforcement learning (RL) has become the dominant approach for post-training language models on reasoning tasks in verifiable settings such as math and coding [Shao et al., 2024b, Guo et al., 2025, Chen et al., 2025a, Zheng et al., 2025]. These methods require only a binary reward, typically whether the final answer is correct, that makes them broadly applicable. However, a binary reward per response provides no information about which intermediate reasoning steps were sound. This sparse supervision makes training expensive: the model must discover good reasoning by comparing and contrasting many self-generated responses.

Distillation methods provide a more sample-efficient alternative by converting supervision into token-level feedback on student-generated responses. *On-policy distillation* methods [Agarwal et al., 2024, Gu et al., 2024, Boizard et al., 2025, Minixhofer et al., 2025, Lu and Lab, 2025] assume access to an external stronger teacher that can provide token-level feedback on the student's responses. More recent *self-distillation* methods, including OPSD [Zhao et al., 2026a], SDFT [Shenfeld et al., 2026a], and SDPO [Hübotter et al., 2026], remove the external teacher but still require high-quality demonstrations that are much better than the model's responses. These demonstrations are typically assumed to come from an external teacher (OPSD, SDFT) or repeated generation and filtering from the model itself (SDPO) [1]. Collecting such supervision can be unavailable or prohibitively expensive to collect. This raises the central question of our work:

> Can the model condition its own initial attempts (possibly incorrect) and their sparse rewards, and provide improved dense supervision to itself?

Self-Distillation Zero (SD-ZERO): Our method is built around a single model that plays two roles: a *generator* and a *reviser*. As a generator, the model produces candidate responses to a given question. As a reviser, the same model is conditioned on its own response and reward, and either corrects an *incorrect* response or rephrases a *correct* one.

SD-ZERO proceeds in two phases. In the first phase, we collect self-revision traces by sampling responses, checking for correctness, and prompting the model to revise incorrect ones. We retain only traces where revision succeeds. We fine-tune on these filtered traces, a procedure we call Self-Revision Training (SRT). In the second phase (self-distillation), we use the reviser as a teacher

[1] See Table 2 for a detailed comparison of post-training methods.

to provide token-level supervision over the generator's responses, thereby transforming outcome-level reward into dense token-level supervision.

Key Findings. Applying SD-ZERO on Olmo and Qwen models on math and code reasoning datasets shows the following:

- **Phase 1 (SRT) alone outperforms all baselines on the same data budget.** Training on 6K self-generated revision traces improves average accuracy by 7.8% for Qwen3-4B-Instruct and 9.2% for Olmo-3-7B-Instruct (Table 1).
- **Phase 2 (Self-Distillation) is critical for the training and inference efficiency of SD-ZERO.** SRT trained model produces overly long responses. Self-Distillation trains the model to internalize its *revision* behavior and generate stronger answers directly as the *generator*, reducing response length by roughly $2\times$ (Figure 3). It also improves performance beyond SRT, yielding an additional 2.7% gain for Qwen3-4B-Instruct and 1.2% for Olmo-3-7B-Instruct, for total gains of 10.5% and 10.4% over the base models. *Importantly*, Self-Distillation is also what makes SD-ZERO training sample-efficient during training: it requires only one response per question, whereas SRT must sample multiple responses to construct revision traces (Section 3.3)
- **SD-ZERO enables iterative self-evolution through regular teacher synchronization.** Because training also improves the model's revision capability, the updated model can serve as the reviser teacher in subsequent rounds of Self-Distillation. After one epoch of Self-Distillation, synchronizing the teacher with the updated model yields a further performance gain of at least 3% (Figure 5), suggesting that SD-ZERO can continue to improve over multiple rounds of Self-Distillation.

2 SD-ZERO: Turning Binary Rewards into Dense Self-Supervision

We consider a dataset $\mathcal{D} = \{(x_i, a_i)\}_{i=1}^{N}$, where x_i denotes an input problem and a_i its corresponding ground-truth final answer. Crucially, we do not assume access to gold solutions. Let π_θ denote the student policy with parameters θ, which generates a reasoning response y for input x according to $\pi_\theta(y \mid x)$. For each example (x, a) and generated response y, we assume access to a binary reward $r(y, a) \in \{0, 1\}$, where $r(y, a) = 1$ if the final answer extracted from y matches a, and $r(y, a) = 0$ otherwise.

The core design of SD-ZERO is around a single model that can play two roles: a *generator* and a *reviser*. We partition the training set into two disjoint subsets of sizes N_1 and N_2:

- **Phase 1, Self-Revision Training**: Using the first subset of N_1 examples, we train the model to perform both the generator and reviser roles.
- **Phase 2, Self-Distillation**: Using the remaining N_2 examples, we leverage the reviser to transform sparse outcome-level supervision into dense token-level supervision over the generator's responses.

In our experiments, we refer to the model obtained after Phase 1 as the SRT model, and the model obtained after Phase 2 as the SD-ZERO model.

2.1 Phase 1 (SRT): Self-Revision Training

Our base models exhibit strong *generator* performance but weak *reviser* performance (Figure 3). This phase strengthens the model as a *reviser*, while also improving it as a *generator*.

Outcome-Conditioned Self-Revision. For each input x, we sample multiple initial reasoning responses $y_{\text{init}} \sim \pi_\theta(\cdot \mid x)$. Given a sampled response y_{init}, we construct a revision prompt that conditions on the original problem, the sampled response, and a control phrase indicating whether the response should be revised or simply rephrased:

$$P_r = \begin{cases} \text{“Let me rephrase the above solution.”}, & \text{if } r(y_{\text{init}}, a) = 1, \\ \text{“Wait, this response is not correct, let me start over.”}, & \text{if } r(y_{\text{init}}, a) = 0. \end{cases}$$

We then use the same model to generate a revised response:

$$y_{\text{revised}} \sim \pi_\theta(\cdot \mid x, y_{\text{init}}, P_r).$$

For correct initial responses, this encourages the model to produce a rephrase of a correct solution, for incorrect initial responses, this encourages the model to critique on the previous attempt and regenerate a correct solution. We observe that when asked to rephrase a correct response, the model typically produces a shorter answer, which may additionally encourage more concise responses during the second phase. We defer further discussion to Section 3. We create a dataset $\mathcal{D}_{\text{REVISION}} = \{(x, y_{\text{init}}, P_r, y_{\text{revised}})\}$.

Self-Revision Objective. We train the model on two tasks simultaneously: given the input, initial attempt, and reward signal, produce a corrected or rephrased response (revision); and given only the input, produce the full correct response from scratch (generation). The corresponding loss $\mathcal{L}_{\text{SRT}}$ consists of two log-likelihood terms $\mathcal{L}_{\text{revision}}$ and $\mathcal{L}_{\text{generation}}$.

The *revision loss* trains the model to produce y_{revised} when conditioned on the input x, the first attempt y_{init}, and the reward prompt P_r:

$$\mathcal{L}_{\text{revision}}(\theta) = \mathbb{E}_{(x, y_{\text{init}}, P_r, y_{\text{revised}}) \sim \mathcal{D}_{\text{REVISION}}} \left[-\sum_{t=1}^{|y'|} \log \pi_\theta\left(y'_t \mid x, y_{\text{init}}, P_r, y'_{<t}\right) \right], \text{where } y' = y_{\text{revised}}. \quad (1)$$

The *generation loss* retains the model's ability to produce the full correct response conditioned only on input x:

$$\mathcal{L}_{\text{generation}}(\theta) = \mathbb{E}_{(x, y_{\text{init}}, P_r, y_{\text{revised}}) \sim \mathcal{D}_{\text{REVISION}}} \left[-\sum_{t=1}^{|y'|} \log \pi_\theta\left(y'_t \mid x, y'_{<t}\right) \right], \text{where } y' = [y_{\text{init}}, P_r, y_{\text{revised}}]. \quad (2)$$

The overall self-revision objective combines these two terms:

$$\mathcal{L}_{\text{SRT}}(\theta) = \mathcal{L}_{\text{revision}}(\theta) + \mathcal{L}_{\text{generation}}(\theta). \quad (3)$$

$\mathcal{L}_{\text{revision}}$ (Eq. 1) trains the model to self-revise, i.e., producing y_{revised} conditioned on the input x, the first attempt y_{init}, and the prompt P_r, while $\mathcal{L}_{\text{generation}}$ (Eq. 2) retains the model's generation capability, i.e., producing correct response y conditioned on input x. This combination implicitly teaches the model to actively evaluate its current response at inference time and generate a self-revision if necessary. As a result, the trained model often exhibits explicit self-revision behaviors that lead to extremely long responses (see Sections 3.2 and 3.3). In the next phase, we perform on-policy distillation on the SRT model, distilling this learned revision behavior back into the generator to enable sample-efficient self-improvement.

2.2 Phase 2 (Self-Distillation): On-Policy Self-Distillation via Revision Feedback

Self-Distillation phase distills the reviser's behavior back into the generator, so that the model can internalize revision and produce more compact and well-directed responses. Concretely, we leverage the self-revision capability learned in Phase 1 to perform on-policy self-distillation. Let θ_{SRT} denote the model obtained at the end of Phase 1. In Phase 2, we initialize the student parameters as $\theta := \theta_{\text{SRT}}$. The **generator** (student) uses the current parameters θ to produce on-policy responses $\pi_\theta(\cdot \mid x)$; its parameters are updated throughout Phase 2. The **reviser** (teacher) remains frozen at θ_{SRT} and provides token-level supervision by conditioning on the generator's response and its binary reward via $\pi_{\theta_{\text{SRT}}}(\cdot \mid x, y, P_r)$. The generator is trained to match the reviser's distribution with:

$$\mathcal{L}_{\text{Self-Distillation}}(\theta) = \mathbb{E}_{(x,a)\sim\mathcal{D}}\, \mathbb{E}_{y\sim\pi_\theta(\cdot|x)} \sum_{t=1}^{|y|} D_{\text{KL}}(\pi_\theta(\cdot \mid x, y_{<t}) \parallel \pi_{\theta_{\text{SRT}}}(\cdot \mid x, y, P_r, y_{<t})). \quad (4)$$

For each training example, a response is sampled from the model and checked against the ground-truth final answer for correctness. The Phase 1 reviser, conditioned on the model's response and whether it was correct, produces a next-token distribution at each token position. The model is trained to match this distribution via KL divergence loss.

	Math						Code		2*Avg.
	AIME24	AIME25	HMMT25	AMOBench	OpenR1	MATH	Codeforces	LCB	
Qwen3-4B-Instruct	59.6	45.8	26.7	9.8	55.8	91.0	48.0	61.8	49.8
SFT	61.7	46.3	31.3	7.3	56.1	91.3	49.1	57.2	50.0
RFT	64.2	52.1	37.1	11.3	59.3	91.9	50.9	68.0	54.3
GRPO	62.5	50.0	30.4	11.0	**62.9**	93.5	52.2	62.6	53.1
SDFT	63.3	47.9	32.9	9.0	57.0	91.1	49.3	59.2	51.2
SRT (Ours)	66.7	59.2	40.0	**16.0**	59.8	92.4	52.7	74.4	57.6
SD-ZERO (Ours)	**68.3**	**60.0**	**45.4**	**16.0**	60.4	**93.6**	**56.1**	**82.6**	**60.3**
Olmo-3-7B-Instruct	56.7	42.1	25.0	1.3	48.9	91.1	31.7	32.4	41.1
SFT	51.3	43.8	30.4	1.3	48.5	91.4	31.7	41.0	42.4
RFT	56.7	48.3	35.8	2.3	50.9	91.4	39.2	49.4	46.7
GRPO	54.6	43.8	25.8	4.8	**56.9**	91.8	37.1	43.6	44.8
SDFT	52.9	45.0	32.1	1.3	49.0	91.2	33.5	42.3	43.4
SRT (Ours)	59.2	52.9	39.6	3.5	52.4	92.3	42.8	**59.6**	50.3
SD-ZERO (Ours)	**61.7**	**53.8**	**40.4**	**5.5**	55.3	**94.0**	**43.5**	57.8	**51.5**

Table 1: **Performance comparison of SD-ZERO and SRT (Self-Revision Training, Phase 1) against baseline post-training methods** on math and code reasoning benchmarks, reported as avg@8. Across both Qwen3-4B-Instruct and Olmo-3-7B-Instruct, SRT and especially SD-ZERO achieve the strongest overall results; "LCB" denotes LiveCodeBench. All methods are compute equalized (see Appendix C.2 for more details).

3 Experiments

3.1 Experimental Setup

Models. We use Qwen3-4B-Instruct [Yang et al., 2025] and Olmo-3-7B-Instruct [Olmo et al., 2025] as base models. All sampling uses temperature 0.7 with a 16K token limit during training and a 32K limit at evaluation. We report avg@8 throughout.

Datasets. We train on two domains separately: (1) **OpenR1-Math** [Hugging Face, 2025], from which we select 15K competition- and olympiad-level problems with verified solutions, and (2) **Codeforces** [Penedo et al., 2025], from which we select 7.5K samples from the cpp (`solutions`) subset and 7.5K from the Python (`solutions_py`) subset. We include more details on training data curation in Appendix C.1.

Evaluation. We evaluate on eight benchmarks in math and code domains: competition math (AIME24 [Zhang and Math-AI, 2024], AIME25 [Zhang and Math-AI, 2025], HMMT25 [Balunović et al., 2025], MATH [Hendrycks et al., 2021]), olympiad math (AMOBench [An et al., 2025] and OpenR1-Math [Hugging Face, 2025]), and competitive programming (Codeforces [Penedo et al., 2025] and LiveCodeBench [Jain et al., 2024]). For the two in-distribution datasets we hold out 500 test questions each.

All baselines train on the same 15K questions: (1) **SFT** on high-quality demonstrations from DeepSeek-R1 [Guo et al., 2025], (2) **RFT**, rejection fine-tuning on correct self-generated traces [Yuan et al., 2023], (3) **GRPO** with binary correctness reward [Shao et al., 2024a] [2], and (4) **SDFT**, on-policy self-distillation with the model conditioned on high-quality demonstrations as its own teacher [Zhao et al., 2026b, Shenfeld et al., 2026b]. We attach a detailed comparison of sampling budgets across SD-ZERO and baseline methods in Appendix C.2, and training hyperparameters in Appendix C.3.

[2] We use DAPO (Yu et al. [2025]), an improved and commonly used variant of GRPO.

3.2 SRT Outperforms Training on Expert Solutions or Filtered Self-Generations

We first compare SRT against training on high-quality demonstrations (SFT) and filtered correct self-generations (RFT) (Table 1). Trained on only 6K self-revision responses, SRT improves average accuracy by 7.8% on Qwen3-4B-Instruct and 9.2% on Olmo-3-7B-Instruct, substantially outperforming both SFT and RFT trained on 15K examples. The baselines fail for different reasons: SFT on DeepSeek-R1 responses degrades Qwen3-4B-Instruct on benchmarks such as AMOBench and LiveCodeBench, while RFT shows minimal gains on harder tasks such as AMOBench.

SRT is effective because it supervises error correction: each example pairs an on-policy incorrect attempt with a self-corrected revision; we find that correctness filtering of revision traces is important (Appendix G). Unlike RFT, which removes incorrect reasoning entirely, SRT preserves the failed attempt as context, allowing the model to learn from its own mistakes. This structure yields especially strong gains on harder benchmarks such as AIME25, HMMT25, and LiveCodeBench.

SRT Significantly Boosts Self-Revision Capability. To measure directly, we run a Generate-then-Revise evaluation on 1K AIME24 questions using Qwen3-4B-Instruct: (1) *First Attempt*: sample one response per question, (2) *Revised Attempt*: prompt the model to generate a revision conditioned on the first attempt and its binary correctness reward. As shown in Figure 3 (detailed statistics in Table 9), while the base model gains only 1.1% from revision, SRT gains $+5.0\%$, indicating that the SRT model has learned to use outcome reward to correct its first attempt rather than simply resampling a new attempt. Notably, the revision responses produced by SRT are, on average, even shorter than the model's first attempts, suggesting more targeted and proactive reasoning during revision; in contrast, revisions from the base model tend to be longer and likely more redundant. More interestingly, the self-revision capability of the model improves further after the Self-Distillation phase. This observation is central to the self-evolving nature of SD-ZERO, which we discuss further in Section 3.4.

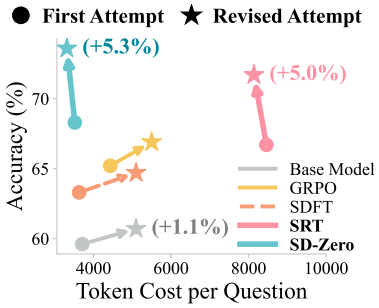

Figure 3: **Comparison of outcome-conditioned self-revision capability** on AIME24, Qwen3-4B-Instruct. SRT unlocks self-revision behaviors, while final SD-ZERO model preserves this advantage while improving token efficiency.

3.3 Self-Distillation Distills Revision into Token-Efficient Generation

After SRT, the model as a generator tends to produce longer responses (as shown in Figure 3) with significantly more self-revision behaviors. Self-Distillation phase distills the self-revision behavior of the model back into more proactive responses of the generator.

Self-Distillation Enables Stronger, Token-Efficient Generations. Table 1 shows that Self-Distillation phase adds 2.7% on Qwen3-4B-Instruct and 1.2% on Olmo-3-7B-Instruct beyond SRT, for total gains of $+10.5\%$ and $+10.4\%$ over the base models. For Qwen3-4B-Instruct, the gains concentrate on code benchmarks and HMMT25. For Olmo-3-7B-Instruct, the Self-Distillation phase mainly improves on math benchmarks such as AIME24 and AMOBench. We also report pass@8 results in Table 7. Interestingly, pass@8 improves substantially as well, suggesting that SD-ZERO does not merely sharpen the output distribution, a behavior often associated with RL-based methods such as GRPO [Yue et al., 2025].

Furthermore, as shown in Figure 3, SD-ZERO model generates only half as many tokens as SRT trained model and fewer tokens than all baselines, while achieving the strongest overall performance. We provide a deeper analysis in Section 4.2, where we show that distilling from the reviser helps the generator to produce more proactive and efficient responses.

Comparison to SRT. Since most of the performance gains in SD-ZERO arise during the SRT phase, one may question the importance of Self-Distillation. We argue that Self-Distillation is nevertheless a critical component of the SD-ZERO pipeline for two reasons. First, Self-Distillation reduces response length at inference by at least $2\times$, substantially improving inference-time efficiency (Figure 3). Second, Self-Distillation is highly sample-efficient during training. Constructing the SRT dataset requires collecting multiple sampled responses per question in order to obtain successful revisions, whereas Self-Distillation requires only a single response per question and uses the reviser's token-level feedback to provide dense supervision. We provide a detailed sample budget analysis in Appendix C.2.

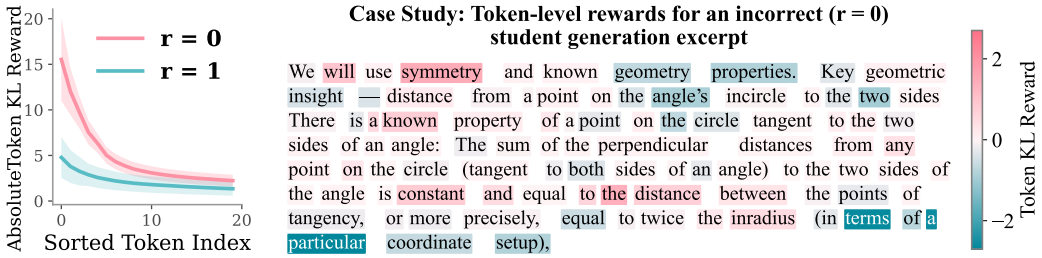

Figure 4: **Reviser converts binary outcome reward into dense token-level reward. Left:** Comparison of token-level KL reward distributions for correct ($\mathbf{r} = \mathbf{1}$) and incorrect ($\mathbf{r} = \mathbf{0}$) student generations. Incorrect trajectories concentrate larger rewards on a small number of tokens, whereas correct trajectories receive a flatter reward distribution that mainly preserves the response. **Right:** Visualization of token-level KL rewards for an incorrect student generation ($\mathbf{r} = \mathbf{0}$). The teacher policy converts binary reward $r = 0$ to dense rewards that localize mistake-relevant tokens and guide targeted correction.

Comparison to GRPO and SDFT. SD-ZERO outperforms by at least 4.8% on average across benchmarks. These comparisons are made under a matched generation budget, where all methods use the same set of questions and are normalized by the number of model-generated responses in their respective pipelines.

We emphasize that this improvement is substantial for two reasons. First, unlike SDFT, that require gold solutions for each problem, SD-ZERO requires only a scalar reward on the model's first attempt. In Table 8, we show that SDFT performs much worse when given access only to the final answer but not the gold solutions. Second, unlike GRPO, whose training requires a group of sampled responses per question, the Self-Distillation phase of SD-ZERO uses only a single response per question, substantially reducing generation cost. While we note that GRPO may benefit from additional training epochs, our comparison is under a matched single-epoch budget, where SD-ZERO achieves stronger performance with comparable total sample cost (see Appendix C.2 for detailed budget analysis).

Takeaway: SD-ZERO offers the best performance per model generations among the compared methods, while requiring neither an external teacher nor high-quality demonstrations.

3.4 Iterative Self-Evolution: Teacher Synchronization Enables Continued Gains

In SD-ZERO setup, the teacher is fixed as the SRT model throughout Phase 2. Performance eventually plateaus because the supervision signal is bounded by a stale teacher. But Self-Distillation phase also improves the model's revision capability: the SD-ZERO model achieves a 5.3% revision gain on the Generate-then-Revise evaluation, comparable to SRT's 5.0% (Figure 3). The improved student can therefore serve as a stronger teacher, enabling what we call ***iterative self-evolution***. The capability to revise answers is distilled back into generation, and regular teacher synchronization allows the loop to sustain.

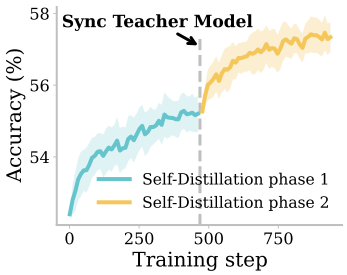

Figure 5: **Self-evolved reasoning through teacher synchronization in SD-ZERO.** SD-ZERO can iteratively improve the model by reusing its own learned self-revision behavior as supervision.

We test this by synchronizing the teacher with the trained student after one epoch and continuing training. Figure 5 shows the result on OpenR1-Math (Qwen3-4B-Instruct): the first Self-Distillation phase saturates around step 400; after teacher synchronization, a second phase yields at least 3 additional percentage points without signs of saturation. Once primed by SRT, the model can continue to self-evolve through iterated teacher synchronization, requiring only the initial attempt and corresponding binary reward in the teacher's context.

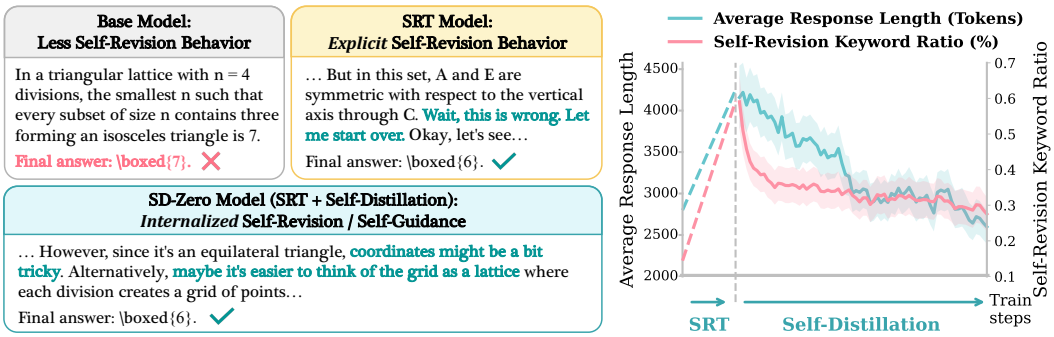

Figure 6: **Evolution of self-revision behavior across training phases** for Qwen3-4B-Instruct on OpenR1-Math. **Left:** Example reasoning traces from different model phases. The base model fails without revision, whereas the SRT model performs explicit self-revision and the SD-ZERO model exhibits more internalized self-guidance, identifying pitfalls and directing itself to the correct answer. **Right:** Training dynamics of self-revision behavior. Response length and explicit self-revision keywords rise during the SRT phase and fall during the Self-Distillation phase, indicating a shift from overt revision to more internalized, token-efficient reasoning.

4 Understanding How SD-ZERO Improves Reasoning

To understand why SD-ZERO improves reasoning, we provide three analyses: Section 4.1 shows that the reviser gives highly localized token-level feedback; Section 4.2 shows that the model gradually learns shorter, more concise reasoning during SD-ZERO; and Section 4.3 validates key design choices in SD-ZERO.

4.1 Token-Level Self-Localization: The Reviser Identifies Which Tokens to Correct

Although the reviser receives only a binary outcome $r \in \{0, 1\}$, its feedback concentrates on a small subset of tokens, a property we call *token-level self-localization*. We decompose the Self-Distillation loss into token-wise terms

$$D_{\text{KL}}^{(t)} \;:=\; D_{\text{KL}}\big(\pi_\theta(\cdot \mid x, y_{<t}) \;\big\|\; \pi_{\theta_{\text{SRT}}}(\cdot \mid x, y, P_r, y_{<t})\big),$$

which is approximately the log-probability gap between the generator and reviser at token t. We define **Token KL Reward** at token t as: $\log \pi_\theta(y_t \mid x, y_{<t}) - \log \pi_{\theta_{\text{SRT}}}(y_t \mid x, y, P_r, y_{<t})$.

To quantify how this signal is distributed, we sort tokens within each response by $D_{\text{KL}}^{(t)}$, divide them into 20 equal-sized buckets, and average within each bucket across 200 responses, separately for correct ($r = 1$) and incorrect ($r = 0$) generations. As shown in Figure 4, the distributions differ sharply: for incorrect responses, most of the KL mass is concentrated on a small fraction of tokens, whereas for correct responses it is much more uniform.

Figure 4 (right) illustrates this on one incorrect generation. Tokens corresponding to the faulty symmetry-based argument receive large positive KL, while tokens associated with the correct coordinate-based solution receive large negative KL. Thus, the reviser does more than penalize incorrect reasoning: it both localizes the error and redirects the model toward a better alternative, converting a scalar outcome into a two-sided token-level signal.

4.2 Evolution of Reasoning Behaviors: SD-ZERO Internalizes Self-Revision

As shown in Figure 3, the SD-ZERO model after Self-Distillation training produces substantially shorter generations, suggesting that it might have internalized some formerly explicit reasoning. Here, we take a closer look at how this generation behavior evolves over SD-ZERO training. Figure 6 (Right) tracks two statistics: average response length and frequency of explicit self-revision keywords (e.g., "Wait" "let me start over"; see Appendix E.1 for the full keyword list). During the SRT phase, both metrics rise substantially. The model learns to solve problems by actively judging its own response and applying self-correction. During Self-Distillation phase, the trend reverses: both response length

and self-revision keyword frequency fall steadily, while accuracy continues to improve. The resulting model only uses approximately half as many tokens as the SRT model at better accuracy.

We qualitatively illustrate this two-stage pattern via example traces from different models in Figure 6 (Left). The base model answers with a false claim. The SRT model reached the correct answer by backtracking with the exact transition phrase seen in training ("Wait, this is wrong. Let me start over."). By contrast, the SD-ZERO model, without any backtracking, anticipates the same pitfall and directs proactively to the correct answer. Note that Self-Distillation does not reduce the revision capability acquired in SRT. The model still achieves a 5.3% gain in the Generate-then-Revise evaluation (Figure 3), but it has successfully internalized some of that capability into a more directed, pitfall-aware attempt.

4.3 Ablations on SD-ZERO Design

We examine several design choices in SD-ZERO.

Loss Terms in SRT Objective. We ablate the two terms in $\mathcal{L}_{\text{SRT}}$ by training with only $\mathcal{L}_{\text{generation}}$ or only $\mathcal{L}_{\text{revision}}$ (Table 11). Neither alone matches full SRT: $\mathcal{L}_{\text{generation}}$ preserves stronger generation but weak self-revision, while $\mathcal{L}_{\text{revision}}$ improves revision but barely boosts generation. The two terms are therefore complementary: $\mathcal{L}_{\text{revision}}$ elicits self-revision, and $\mathcal{L}_{\text{generation}}$ transfers it to stronger generation.

Self-Distillation Phase without SRT. SD-ZERO requires the model to have certain self-revision ability, which is unlocked only after the SRT phase (Figure 3). To further demonstrate its necessity, we apply the Self-Distillation phase directly to Qwen3-4B-Instruct without prior self-revision training. Table 12 shows that this Phase-2-only variant yields only marginal improvements over the base model, and does not improve self-revision ability. Therefore, the SRT phase is necessary for SD-ZERO to be effective.

Data Split Between Phases. We study how to allocate a fixed training data budget between the SRT and Self-Distillation phases. As shown in Table 13, giving more data to SRT slightly improves the SRT model, but does not improve the final performance. Instead, the best result is obtained by assigning more data to the Self-Distillation phase, suggesting that once SRT has unlocked self-revision capability, Self-Distillation phase uses additional data more effectively to distill and refine this behavior into stronger generation.

5 Discussion

Concurrent and Related Works. We provide a detailed discussion of related work in Section A, focusing on on-policy distillation and self-improvement methods. Here, we briefly compare our method to some of the most recent concurrent works.

On-policy Self-Distillation (OPSD) [Zhao et al., 2026b] and Self-Distillation Fine-Tuning (SDFT) [Shenfeld et al., 2026a] the same model under training to provide dense token-level supervision to itself. However, both rely on access to gold solutions, possibly generated from a stronger teacher. In contrast, SD-ZERO requires no external teacher or gold solutions, and instead derives token-level supervision directly from sparse outcome reward. Empirically, we find that OPSD can substantially underperform SD-ZERO despite having access to gold solutions, suggesting that mismatches between reasoning style of gold solutions and the model's own responses can make self-distillation ineffective.

Self-Distillation Policy Optimization (SDPO) [Hübotter et al., 2026] also avoids an external teacher, but assumes access to finer-grained error signals indicating where the initial response fails. Such signals may come from successful self-generated responses filtered by reward, which can be expensive to collect and use as feedback, or from environment feedback available only in very structured domains such as coding, which can be difficult to create in general domains. By comparison, SD-ZERO requires only binary outcome supervision, whether the initial response is correct or incorrect, and still converts this signal into dense token-level guidance, making it substantially more lightweight and broadly applicable.

Conclusion. SD-ZERO shows that a model's revision ability, elicited with a small amount of supervised training on self-generated revisions, can transform sparse binary outcome reward into dense token-level supervision without requiring an external teacher or high-quality demonstrations. Across two model families and eight benchmarks, this yields consistent $10\%+$ gains over the base

models and outperforms all baselines under the same training sample budget (Table 1). We also show experiments on more broader tasks and models in Appendix G. Extensive ablations and analyses show that SD-ZERO also leads to improved revision capability which can lead to further gains on accuracy. We hope that by eliminating the need for high-quality demonstrations, we enable future work to apply self-distillation to broader domains.

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. In *International Conference on Learning Representations*, 2024.

Shengnan An, Xunliang Cai, Xuezhi Cao, Xiaoyu Li, Yehao Lin, Junlin Liu, Xinxuan Lv, Dan Ma, Xuanlin Wang, Ziwen Wang, and Shuang Zhou. Amo-bench: Large language models still struggle in high school math competitions, 2025. URL `https://arxiv.org/abs/2510.26768`.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. A general language assistant as a laboratory for alignment, 2021. URL `https://arxiv.org/abs/2112.00861`.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions, February 2025. URL `https://matharena.ai/`.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

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.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025a.

Howard Chen, Noam Razin, Karthik Narasimhan, and Danqi Chen. Retaining by doing: The role of on-policy data in mitigating forgetting, 2025b. URL `https://arxiv.org/abs/2510.18874`.

Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1607–1616. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/furlanello18a.html`.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars, 2025. URL `https://arxiv.org/abs/2503.01307`.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong

Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Alex Havrilla, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, and Roberta Robeyns. GLoRE: When, where, and how to improve LLM reasoning via global and local refinements. In *International Conference on Machine Learning*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

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

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL https://github.com/huggingface/open-r1.

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, 2024. URL https://arxiv.org/abs/2403.07974.

Ryo Kamoi, Yusen An, Nan Zhang, Jiaqi Gao, and Rui Zhang. When can LLMs actually correct their own mistakes? A critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 2024.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, 2016.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, JD Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024a.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning, 2024b. URL https://arxiv.org/abs/2409.12917.

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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Benjamin Minixhofer, Ivan Vulić, and Edoardo Maria Ponti. Universal cross-tokenizer distillation via approximate likelihood matching. In *Advances in Neural Information Processing Systems*, 2025.

Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces. https://huggingface.co/datasets/open-r1/codeforces, 2025.

Hejian Sang, Yuanda Xu, Zhengze Zhou, Ran He, Zhipeng Wang, and Jiachen Sun. On-policy self-distillation for reasoning compression, 2026. URL https://arxiv.org/abs/2603.05433.

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, 2024a. URL https://arxiv.org/abs/2402.03300.

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

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

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

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, 2025. URL `https://arxiv.org/abs/2505.09388`.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models, 2023. URL `https://arxiv.org/abs/2308.01825`.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation, 2019. URL `https://arxiv.org/abs/1905.08094`.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 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*, 2026a.

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, 2026b. URL `https://arxiv.org/abs/2601.18734`.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Appendix Content

A Related Work 16

B Method Design 17
- B.1 SD-ZERO Training Algorithm 17
- B.2 Comparison to Existing Methods 17

C Experiment Details 19
- C.1 Curating SD-ZERO training data 19
- C.2 Comparing Sampling Budgets 19
- C.3 Hyperparameters 20

D Additional Results 24
- D.1 Pass@8 Performance on Math Benchmarks 24
- D.2 SDFT Under Final-Answer-Only Supervision 24
- D.3 Detailed Self-Revision Statistics 24
- D.4 GRPO Configuration Comparison 24

E Analysis of SD-ZERO 27
- E.1 Self-Revision Keyword Analysis 27
- E.2 Ablation Studies on SD-ZERO Method Design 27

F SDFT with Thinking Models 30

G Self-Revision Design Choices 31
- G.1 Effect of Training Data Source 31
- G.2 Self-Revision as Initialization for RL 31

A Related Work

On-Policy Distillation and Self-Distillation. Knowledge distillation trains a student to match a teacher's output distribution either at the token level via soft targets [Hinton et al., 2015] or at the sequence level [Kim and Rush, 2016]. However, such off-policy training introduces a train-inference distribution mismatch that leads to compounding errors at test time [Bengio et al., 2015, Chen et al., 2025b]. On-policy distillation (OPD) [Agarwal et al., 2024, Gu et al., 2024, Boizard et al., 2025, Minixhofer et al., 2025] addresses this by using student rollouts with per-token supervision from the teacher.

A key limitation of OPD is the need for an external model during training to score student traces. Recent *self-distillation* methods [Furlanello et al., 2018, Zhang et al., 2019] eliminate this by conditioning the student on privileged information to serve as its own teacher [Askell et al., 2021, Snell et al., 2022], where the privileged context can include reasoning traces [Zhao et al., 2026a], in-context demonstrations [Shenfeld et al., 2026b], or conciseness prompts [Sang et al., 2026]. Still, such works assume access to high-quality signals from external sources that are often costly to obtain, and those that remove the need for an external teacher entirely require rich, dense environment feedback [Hübotter et al., 2026]. In contrast, SD-ZERO only requires binary correctness signals of the student's response and generates its own supervision through self-revision.

Self-Training and Self-Refinement. Self-training methods improve models by iteratively generating and filtering their own data. Prior work bootstraps reasoning by fine-tuning on self-generated rationales that yield correct answers [Zelikman et al., 2022, Yuan et al., 2023, Singh et al., 2024, Yuan et al., 2024]. In particular, STaR [Zelikman et al., 2022] iteratively generates and filters for correct rationales, discarding incorrect reasoning entirely. By contrast, SRT retains the incorrect attempt as context and trains on the full mistake-to-correction trajectory. Consistent with this, recent evidence suggests that productive reasoning behaviors (e.g., verification, backtracking, etc.) matter more for self-improvement than answer correctness [Gandhi et al., 2025]. Separately, prompting-based self-refinement methods show that LLMs are capable of critiquing and revise their own outputs at inference time [Madaan et al., 2023, Shinn et al., 2023], though such approaches yield limited gains without external feedback and do not update model weights [Huang et al., 2023, Kamoi et al., 2024]. Training-based approaches address this by using RL or supervised learning to internalize correction behavior [Kumar et al., 2024a, Havrilla et al., 2024], but still require multi-turn generation at inference time. SD-ZERO bridges both avenues by training the model to self-revise and then distilling the revision-informed distribution back into single-pass generation, eliminating multi-turn correction at test time.

Connection to Process Reward Models The per-token KL signal from the reviser is functionally analogous to process reward models (PRMs) [Lightman et al., 2024, Wang et al., 2024], in that both provide localized supervision over intermediate reasoning rather than only the final outcome. PRMs train a separate model to assign step-level scores to individual reasoning steps. In SD-ZERO, the token-level divergence $D_{KL}^{(t)}$ similarly concentrates on a sparse subset of tokens associated with errors in the response (Figure 4), so that the log-probability gap at each token both penalizes erroneous reasoning and redirects probability mass towards plausible alternatives. The main difference is cost and supervision: PRMs typically require either step-level correctness annotations [Lightman et al., 2024] or rollout-based estimates from search or sampling [Wang et al., 2024], together with a separately trained reward model, whereas our reviser's signal arises by conditioning the same model on its own response and a binary outcome, without step-level annotation or an auxiliary reward model. Whether this implicit token-level signal can complement or serve as a cheaper alternative to trained PRMs in settings such as guided tree search remains an interesting direction for future work.

B Method Design

B.1 SD-ZERO Training Algorithm

Section B.1 summarizes the full two-phase training procedure of SD-ZERO (Section 2). In Phase 1 (SRT), the model first learns outcome-conditioned self-revision: for each sampled response, we use the binary verifier to determine whether the response is correct, construct a control prompt that either triggers rephrasing (r=1) or restarting (r=0), and keep only revised responses that are verified as correct to form the self-revision dataset $\mathcal{D}_{\text{REVISION}}$. The model is then trained on this dataset with the SRT objective, which jointly improves its ability to revise and to generate stronger responses. In Phase 2 (Self-Distillation), we initialize the student from the SRT model and perform on-policy self-distillation: the current student generates a response, the fixed SRT reviser conditions on that response and its binary outcome to define a teacher distribution, and the student is updated to match this distribution via KL minimization. In this way, SD-ZERO first elicits explicit self-revision behaviors and then distills them back into more compact, token-efficient generation.

[htbp] [1] Base model π_θ, dataset $\mathcal{D} = \{(x, a)\}_{i=1}^{N}$, binary verifier $r(y, a) \in \{0, 1\}$
$\triangleright$ **Phase 1 (SRT): Self-Revision Training** each $(x, a) \in \mathcal{D}$ Sample initial response $y_{\text{init}} \sim \pi_\theta(\cdot \mid x)$ Compute reward $r \leftarrow r(y_{\text{init}}, a)$ Construct control prompt

$$P_r \leftarrow \begin{cases} \text{“Let me rephrase the above solution.”}, & r = 1, \\ \text{“Wait, this response is not correct, let me start over.”}, & r = 0 \end{cases}$$

Generate revised response $y_{\text{revised}} \sim \pi_\theta(\cdot \mid x, y_{\text{init}}, P_r)$ Add $(x, y_{\text{init}}, P_r, y_{\text{revised}})$ to $\mathcal{D}_{\text{REVISION}}$ **if** $r(y_{\text{revised}}, a) = 1$ **Train** π_θ **on** $\mathcal{D}_{\text{REVISION}}$ **with** $\mathcal{L}_{\text{SRT}}$ **to obtain** $\pi_{\theta_{\text{SRT}}}$

$\triangleright$ **Phase 2 (Self-Distillation): On-Policy Self-Distillation via Revision Feedback** Initialize $\pi_\theta \leftarrow \pi_{\theta_{\text{SRT}}}$ Sample $(x, a) \sim \mathcal{D}$ Sample student response $y_{\text{init}} \sim \pi_\theta(\cdot \mid x)$ Compute reward $r \leftarrow r(y_{\text{init}}, a)$ Construct control prompt P_r as in Phase 1 **Define student policy** $\pi_S(\cdot) \leftarrow \pi_\theta(\cdot \mid x)$ **Define teacher policy** $\pi_T(\cdot) \leftarrow \pi_{\theta_{\text{SRT}}}(\cdot \mid x, y_{\text{init}}, P_r)$ Update θ by minimizing

$$\mathcal{L}_{\text{Self-Distillation}}(\theta) = D_{\text{KL}}(\pi_S \,\|\, \text{StopGrad}(\pi_T))$$

convergence

B.2 Comparison to Existing Methods

Method	Sampling	Signal	Teacher	Teacher can condition on wrong attempt
SFT / Distillation	off-policy	dense	external	—
On-Policy Distillation	on-policy	dense	external	—
RLVR (e.g., GRPO)	on-policy	sparse	—	—
OPSD / SDFT / SDPO	on-policy	dense	self	
SD-ZERO (ours)	on-policy	dense	self	

Table 2: Comparison of post-training methods for LLMs. SD-ZERO is the only self-distillation method whose teacher can condition on incorrect student attempts, enabling it to convert sparse binary rewards into dense token-level supervision without requiring gold demonstrations.

Table 2 situates SD-ZERO among existing post-training methods. Standard SFT and distillation rely on dense supervision but are off-policy and require external demonstrations, while RLVR methods such as GRPO are on-policy but optimize only sparse outcome rewards. Recent self-distillation methods such as OPSD, SDFT, and SDPO combine on-policy sampling with dense self-supervision, but their teacher does not explicitly condition on the student's failed trajectory. In contrast, SD-ZERO is the only method in this comparison that is simultaneously on-policy, self-distilled, and able to condition the teacher on an incorrect attempt, allowing it to transform binary outcome rewards into targeted token-level supervision without requiring gold reasoning traces.

C Experiment Details

C.1 Curating SD-ZERO training data

In SD-ZERO, we trained the models on 15K training data from OpenR1-Math and 15K data from Codeforces separately, with 6K in Phase 1 (SRT) and 9K in Phase 2 (Self-Distillation).

In SRT phase, we curate 6K self-revision training data from 10K question-answer pairs in OpenR1-Math (or Codeforces), through the following pipeline:

1. **Initial sampling:** Select the first 10K questions in OpenR1-Math (or Codeforces), and sample 1 initial model responses y_{initial} per question x,
2. **Verification:** Verify the binary reward $r \in \{0, 1\}$ for each y_{initial}, and build self-revision prompt P_r. The 10K initial responses are roughly split into 5K correct and 5K incorrect responses.
3. **Self-Revision:** For each correct initial response, prompt the model to generate 3 rephrased responses y_{revised}; For each incorrect initial response, prompt the model to generate 3 corrected responses y_{revised},
4. **Filtering:** Keep traces $(x, y_{\text{initial}}, P_r, y_{\text{revised}})$ where y_{revised} reaches a correct final answer. The resulting training data contain 6K self-revision traces.

In Self-Distillation phase, we directly sample an additional 9K question-answer pairs from OpenR1-Math (or Codeforces) as training data.

C.2 Comparing Sampling Budgets

One feature of SD-ZERO is sample efficiency. In Table 3, we compare the sampling budget of SD-ZERO with baseline on-policy training methods, including rejection fine-tuning (RFT), GRPO, and SDFT. We calculate the sampling budget of each methods as follows:

- **RFT:** total # generations = 15K (questions) $\times$ 4 (response attempts per question) = 60K.
- **GRPO:** total # generations = 15K $\times$ 4 (rollout/question) = 60K.
- **SDFT:** total # generations = 15K $\times$ 4 (rollout/question) = 60K.
- **SRT phase:** total # generations = 10K (questions) $\times$ 1 (initial response attempts per question) + 5K (correct initial responses) $\times$ 3 (rephasing attempts per correct initial response) + 5K (incorrect initial responses) $\times$ 3 (self-correction attempts per incorrect initial response) = 40K.
- **Self-Distillation phase:** total # generations = 9K $\times$ 1 (rollout/question) = 9K.
- **SD-ZERO:** 40K (SRT phase) + 9K (Self-Distillation phase) = 49K.

We trained the GRPO baseline for 1 epoch on 15K training qesutions, with 4 generations per question [3].

We also estimate the total token budget. At approximately 3.7K tokens per response, the sampling budgets for RFT and GRPO are each roughly 222M tokens. For SD-ZERO, the SRT data collection phase generates approximately 148M tokens (10K initial responses plus 30K revisions), and the Self-Distillation phase generates at most 76.5M tokens (9K rollouts at up to 8.5K tokens each, though rollout length decreases during training). The total sampling budget for SD-ZERO is thus at most 225M tokens, comparable to baselines despite achieving substantially stronger performance.

Axis 1: Sampling budget (total completion tokens generated).

- **RFT:** $60\text{K} \times 3.7\text{K} \approx 222\text{M}$ tokens
- **GRPO:** $60\text{K} \times 3.7\text{K} \approx 222\text{M}$ tokens
- **SD-ZERO Phase 1 (SRT data collection):**

[3] We also compare to the 8-rollout GRPO variants in Table 10.

2*Method	# Training questions	# Training rollouts per question	# Training epochs	**Total # generations**
RFT	15K	-	1	**60K**
GRPO	15K	4	1	**60K**
SDFT	15K	4	1	**60K**
SRT phase	6K	-	1	**40K**
Self-Distillation phase	9K	1	1	**9K**
SD-ZERO	15K	-	1	**49K**

Table 3: Comparison of training data and generation budgets across different training paradigms. For RFT, GRPO, and SDFT, we assume 15K training questions and 4 sampled rollouts per question, resulting in 60K total generations before any filtering. In our method, the SRT phase starts from 10K seed questions, generates 1 initial response and 3 self-revisions per question (40K generations in total), and retains 6K correct self-revision traces for training. The Self-Distillation phase directly uses 9K additional question-answer pairs, requiring 9K generations. In total, SD-ZERO uses 15K training questions and 49K generations.

  - 10K initial responses $\times$ 3.7K $=$ 37M
  - 30K revisions $\times$ 3.7K $=$ 111M
  - Subtotal: 148M tokens
- **SD-ZERO Phase 2 (Self-Distillation):**
  - 9K rollouts $\times$ 8.5K $\leq$ 76.5M tokens (upper bound; rollout length decreases during this phase per Figure 4)
- **SD-ZERO total:** $\leq$ 224.5M tokens

Axis 2: Training budget (completion tokens in forward pass for loss computation).

- **GRPO:** 60K $\times$ 3.7K $\approx$ 222M tokens (probably more than this since avg response length increases to 4.4K by the end of GRPO)
- **RFT:** $\leq$ 222M (same sampling budget as GRPO but with incorrect samples dropped)
- **SD-ZERO SRT training** (6K traces, two losses per trace):
  - $\mathcal{L}_{\text{revision}}$: 6K $\times$ 3.7K $=$ 22M
  - $\mathcal{L}_{\text{generation}}$: 6K $\times$ 7.4K $=$ 44M (uses $2 \times 3.7K$ since $y' = [y_{\text{init}}, P_r, y_{\text{revised}}]$)
  - Subtotal: 66M tokens
- **SD-ZERO Self-Distillation** (student + teacher forward pass per step):
  - Student: 9K $\times$ 8.5K $=$ 76.5M
  - Teacher: 9K $\times$ 8.5K $=$ 76.5M
  - Subtotal: $\leq$ 153M tokens (probably less than this since avg response length goes down during self-distillation phase)
- **SD-ZERO total:** $\leq$ 219M tokens

C.3 Hyperparameters

We attach the hyperparameters used for Qwen3-4B-Instruct in SFT, RFT, and SD-ZERO SRT phase in Table 4, those used in GRPO in Table 5, and those used in SDFT and SD-ZERO Self-Distillation phase in Table 6.

Parameters	SFT
General	
Model	Qwen/Qwen3-4B-Instruct-2507
Thinking	False
Data	
Prompt format	Chat template (system + user)
Completion-only loss	True
Max. sequence length	32768
Max. response length	32768
Batching	
Per-device train batch size	1
Per-device eval batch size	1
Gradient accumulation steps	1
Effective global batch size	4
Optimization / Training	
Optimizer	AdamW
Learning rate	5×10^{-6}
Scheduler	Cosine
Warmup ratio	0.05
Num. epochs	3
Weight decay	1×10^{-4}
Adam β_1	0.9
Adam β_2	0.95
Precision	bfloat16
Gradient checkpointing	True
Gradient accumulation sync each batch	True
Use Liger kernel	True
Parallelism / FSDP	
FSDP mode	`full_shard auto_wrap`
FSDP wrapped layer	`Qwen3DecoderLayer`

Table 4: Hyperparameters used for SFT/ RFT baselines and SD-ZERO SRT phase on Qwen3-4B-Instruct.

Parameters	GRPO
General	
Model	Qwen/Qwen3-4B-Instruct-2507
Thinking	False
Data	
Max. prompt length	1024
Max. response length	8192
Train batch size	64
Validation batch size	–
Shuffle	True
Truncation	left
Batching	
Rollouts per prompt (n)	8
PPO mini-batch size	64
PPO micro-batch size per GPU	1
Logprob micro-batch size per GPU	1
Max. sequences per rollout batch	64
Max. batched tokens	16384
Rollout / Generation	
Inference engine	vLLM
Rollout mode	sync
Temperature	0.7
Top-p	1.0
Top-k	–
Sampling	True
Max new tokens	8192
Tensor parallel size	2
Enable chunked prefill	True
Actor / Policy Optimization	
Strategy	FSDP
PPO epochs	1
Clip ratio	0.2
Entropy coefficient	0.001
Use KL loss	True
KL loss coefficient	0.001
KL loss type	low-var KL
Gradient clip norm	1.0
Dynamic batch size	False
Reference Model	
Strategy	FSDP
Dynamic batch size	False
Sequence parallel size	1
Parallelism	
Number of GPUs	8
Number of nodes	1
Sequence parallel	1
Actor strategy	FSDP
Reference strategy	FSDP
Gradient checkpointing	True
Training	
Optimizer	AdamW
Learning rate	1×10^{-6}
Scheduler	Constant
Warmup steps ratio	0.0
Weight decay	0.01
Precision	bfloat16
NCCL timeout	1800

Table 5: Hyperparameters used in GRPO baseline on Qwen3-4B-Instruct.

Parameters	SD-ZERO	SDFT
General		
Policy Model	Qwen/Qwen3-4B-Instruct-2507	Qwen/Qwen3-4B-Instruct-2507
Teacher model	Qwen/Qwen3-4B-Instruct-2507	Qwen/Qwen3-4B-Instruct-2507
Thinking	False	False
Data		
Max. teacher prompt length	32768	32768
Max. response length	8192	8192
Batching		
Prompts per step	128	128
Generations per prompt	1	1
Train global batch size	128	128
Train micro batch size	1	1
Generation batch size	64	128
Logprob batch size	1	–
Max rollout turns	1	1
Rollout / Generation		
Inference engine	vLLM	vLLM
Temperature	1.0	1.0
Top-p	1.0	1.0
Top-k	–	–
Max new tokens	8192	8192
vLLM tensor parallel size	4	1
Validation		
Validation batch size	64	–
Validation period	20	–
Distillation loss		
Top-K distillation	64	–
Zero outside top-K	False	–
Parallelism		
Number of GPUs	4	4
Policy tensor parallel size	4	–
Policy context parallel size	1	–
Teacher tensor parallel size	4	–
Teacher context parallel size	1	–
Sequence parallel	False	False
Activation checkpointing	True	True
Training		
Optimizer	AdamW	AdamW
Learning rate	5×10^{-6}	5×10^{-6}
Scheduler	Linear warmup + constant	Cosine (warmup ratio $= 0.1$)
Warmup steps	20	20
Weight decay	0.01	–
Gradient clip norm	1.0	1.0
Precision	bfloat16	bfloat16

Table 6: Hyperparameters used for SD-ZERO Self-Distillation phase and SDFT on Qwen3-4B-Instruct.

D Additional Results

D.1 Pass@8 Performance on Math Benchmarks

Table 7 complements the main results in Table 1 by showing that the advantages of SRT and SD-ZERO persist under Pass@8 evaluation. The trend is consistent with the avg@16 results: across both Qwen3-4B-Instruct and Olmo-3-7B-Instruct, SRT already matches or outperforms strong baselines, and SD-ZERO further achieves the best overall average performance. These gains on Pass@8 suggest that our method is not merely sharpening the output distribution around a narrow set of solutions, but genuinely making the model's exploration more well-directed, so that correct reasoning paths are more likely to be discovered across multiple attempts. Overall, the Pass@8 results further reinforce the conclusion that explicit self-revision training improves reasoning quality, and that the second-stage SD-ZERO training strengthens this effect.

	AIME24	AIME25	HMMT25	AMOBench	OpenR1	MATH	**Average**
Qwen3-4B-Instruct	76.7	76.7	50.0	30.0	69.2	97.6	66.7
SFT	83.3	73.3	46.7	18.0	69.2	97.0	64.6
RFT	**86.7**	**80.0**	53.3	30.0	70.2	**98.4**	69.8
GRPO	80.0	70.0	46.7	28.0	70.0	98.0	65.4
SDFT	80.0	70.0	50.0	26.0	70.0	97.0	65.5
SRT (Ours)	**86.7**	**80.0**	60.0	28.0	70.2	97.6	70.4
SD-ZERO (Ours)	**86.7**	**80.0**	**63.3**	**36.0**	**72.0**	97.0	**72.5**
Olmo-3-7B-Instruct	80.0	73.3	46.7	16.0	60.0	**98.0**	62.3
SFT	83.3	66.7	46.7	18.0	55.0	96.0	60.9
RFT	80.0	73.3	60.0	22.0	62.0	97.0	65.7
GRPO	80.0	66.7	46.7	18.0	70.0	97.0	63.1
SDFT	80.0	73.3	56.7	20.0	58.0	97.0	64.2
SRT (Ours)	**83.3**	76.7	**66.7**	20.0	66.0	**98.0**	68.4
SD-ZERO (Ours)	**83.3**	**80.0**	**66.7**	**30.0**	**68.0**	97.0	**70.8**

Table 7: **Pass@8 performance comparison of SD-ZERO and SRT (Self-Revision Training, Phase 1) against baseline post-training methods** on math reasoning benchmarks. Across both Qwen3-4B-Instruct and Olmo-3-7B-Instruct, SRT and especially SD-ZERO achieve the strongest overall results.

D.2 SDFT Under Final-Answer-Only Supervision

In Table 8, we show that SDFT performs much worse when given access only to ground-truth final answers rather than gold solution traces. Its performance remains close to the base model and falls far behind SD-ZERO across all math benchmarks. These results highlight that existing self-distillation methods do not naturally benefit from final-answer-only supervision, whereas SD-ZERO is specifically designed to convert such sparse outcome feedback into effective token-level learning signals.

D.3 Detailed Self-Revision Statistics

Table 9 reports detailed statistics for the Generate-then-Revise evaluation on 1K AIME24 questions with Qwen3-4B-Instruct as the base model. For each method, we show the average token cost of the first attempt and the revision, together with the corresponding accuracies before and after revision. We also report the net wrong-to-correct rate, computed as $(\text{Revised Attempt Accuracy} - \text{First Attempt Accuracy})/(100 - \text{First Attempt Accuracy})$, which measures the fraction of initially incorrect answers that are corrected after revision. These statistics complement Figure 3 by quantifying both the accuracy gains and the token efficiency of revision across methods.

	AIME24	AIME25	HMMT25	AMOBench	OpenR1	MATH	**Avg.**
Base model	59.6	45.8	26.7	9.8	55.8	91.0	48.1
SDFT	63.3	47.9	29.9	9.0	57.0	91.1	49.7
SDFT (final answers only)	62.5	47.1	29.9	9.3	57.0	91.0	49.5
SD-ZERO (Ours)	**68.3**	**60.0**	**45.4**	**16.0**	**60.4**	**93.6**	**57.3**

Table 8: **SDFT with final-answer-only supervision** on Qwen3-4B-Instruct evaluated on math benchmarks. When given access only to final answers rather than gold solutions, SDFT performs only marginally better than the base model and remains far below SD-ZERO, showing that SD-ZERO is much more effective under outcome-only supervision.

D.4 GRPO Configuration Comparison

The GRPO baseline in Table 1 is run under 4 generations per question under a sampling budget matched with SD-ZERO. Here, we vary the number of generations per question and the number of training epochs in GRPO for Qwen3-4B-Instruct on the math domain.

As shown in Table 10, simply increasing the rollout budget in GRPO does not close the gap to our method. Moving from 4 to 8 generations per question (with 0.5 training epochs to match rollout budget) yields only marginal improvements, and in some settings even slightly hurts performance. Extending training to a full epoch with 8 generations ($2\times$ sampling budget as SD-ZERO) provides some recovery, but the gains remain modest overall.

2*Method	First Attempt Tokens	Revised Attempt Tokens	First Attempt Accuracy (%)	Revised Attempt Accuracy (%)	Correction Rate (%)
Base Model	3708	5098	59.6	60.7	2.7
GRPO	4432	5499	65.2	66.9	4.9
SDFT	3630	5099	63.3	64.7	3.8
SRT	8458	8137	66.7	71.7	**15.0**
SD-ZERO	3518	3314	68.3	73.6	**16.7**

Table 9: Comparison of self-revision effectiveness and token costs across different models on AIME24 with Qwen3-4B-Instruct as the base model. The Correction Rate is computed as $(\text{Revised Attempt Accuracy} - \text{First Attempt Accuracy})/(100 - \text{First Attempt Accuracy})$, i.e., the net fraction of initially incorrect answers corrected after revision.

	AIME24	AIME25	HMMT25	AMOBench	OpenR1	MATH	**Avg.**
GRPO (4 generations, 1 epoch)	62.5	50.0	30.4	11.0	62.9	93.5	51.7
GRPO (8 generations, 0.5 epoch)	58.8	50.0	30.4	9.8	61.5	93.0	50.6
GRPO (8 generations, 1 epoch)	61.7	52.1	31.3	12.0	**63.4**	93.1	52.3
SD-Zero (Ours)	**68.3**	**60.0**	**45.4**	**16.0**	60.4	**93.6**	**57.3**

Table 10: **Comparison with GRPO under larger rollout budgets** on Qwen3-4B-Instruct evaluated on math benchmarks. We compare SD-Zero against GRPO variants that use more sampled generations per prompt and different effective training budgets. Even when GRPO is given substantially more exploration through 8 generations, its gains remain limited, whereas SD-Zero achieves the best average performance and outperforms all GRPO settings on most benchmarks. This suggests that the advantage of our method comes not simply from sampling more trajectories, but from turning binary outcome feedback into more informative self-revision-based supervision.

E Analysis of SD-ZERO

E.1 Self-Revision Keyword Analysis

Below, we list the self-revision keywords used in our behavioral analysis. These keywords are used in the study described in Figure 6 to quantify the frequency of explicit self-correction language during training, complementing the analysis of response length and downstream performance. We construct the list by collecting common phrases that models frequently use when revising an earlier reasoning step, such as signaling doubt, detecting an error, restarting, rechecking, or correcting a previous claim. Concretely, the list includes expressions associated with hesitation (e.g., "wait," "hold on"), error acknowledgment (e.g., "my mistake," "this is wrong"), and explicit restart or verification behavior (e.g., "let me recheck," "let me try again," "let me start over"). While this keyword set is not intended to exhaustively capture all forms of self-correction, it provides a simple and interpretable proxy for tracking overt self-revision behavior across training stages.

List of Self-Revision Keywords Used in Behavioral Analysis (Section 4.2)

```
REVISION_KEYWORDS = [
    "wait",
    "hold on",
    "actually",
    "on second thought",
    "let me recheck",
    "let's recheck",
    "let me check",
    "let's check",
    "let me recalculate",
    "let's recalculate",
    "let me correct",
    "my mistake",
    "i made a mistake",
    "this is wrong",
    "that is wrong",
    "that's wrong",
    "incorrect",
    "re-evaluate",
    "let's re-evaluate",
    "let me rethink",
    "let's rethink",
    "let me double check",
    "let's double check",
    "wait a minute",
    "let me start over",
    "let me try again",
    "oops"
]
```

E.2 Ablation Studies on SD-ZERO Method Design

E.2.1 Ablating Loss terms in SRT

The SRT objective $\mathcal{L}_{\text{SRT}}$ (Section 2) combines two terms: the first term $\mathcal{L}_{\text{revision}}$ trains the model to produce a revision or rephrase conditioned on the outcome; the second term $\mathcal{L}_{\text{generation}}$ trains it to generate better responses by actively evaluating its current response and self-correcting. We ablate this by training with each term alone. If the self-correction term is redundant, removing it should not hurt Phase 2 performance; if the conditioned revision term is redundant, the model should still learn to revise from the self-correction objective alone.

As shown in Table 11, both terms are necessary, and neither alone recovers the full effect of SRT. Using only $\mathcal{L}_{\text{generation}}$ preserves relatively strong first-attempt generation, but its revised-attempt improvement is much smaller, with the correction rate dropping from 15.0% to 7.2%. This suggests that self-correction behavior does not emerge reliably from the generation objective alone. In contrast, using only $\mathcal{L}_{\text{revision}}$ yields a higher correction rate of 12.1%, indicating that the model does learn to revise when explicitly trained to do so, but this comes at a clear cost to generation quality: both average generation accuracy and first-attempt accuracy fall substantially relative to the full SRT objective. Taken together, these results show that the two terms play complementary roles. $\mathcal{L}_{\text{revision}}$ is important for directly eliciting revision behavior, while $\mathcal{L}_{\text{generation}}$ helps transfer that behavior back into stronger standalone generation. Their combination is what enables SRT to simultaneously improve first-attempt performance and unlock effective self-revision. This complementarity is consistent with previous work [Kumar et al., 2024b], which similarly finds that self-correction does not emerge from a correction-focused signal alone, but requires preserving the model's underlying generation ability while separately training revision behavior.

3***Method**	3***Average Generation**	**Generate-then-Revise on AIME24**		
	2***Accuracy (%)**	First Attempt Accuracy (%)	Revised Attempt Accuracy (%)	Correction Rate (%)
Base Model	49.8	59.6	60.7	2.7
SRT	57.6	66.7	71.7	15.0
SRT ($\mathcal{L}_{\text{generation}}$ Only)	56.4	65.4	67.9	7.2
SRT ($\mathcal{L}_{\text{revision}}$ Only)	52.2	62.1	66.7	12.1

Table 11: **Ablation of the two loss terms in the SRT objective.** We train Qwen3-4B-Instruct with SRT using only $\mathcal{L}_{\text{generation}}$ or only $\mathcal{L}_{\text{revision}}$, and evaluate both overall generation quality and self-revision ability. Both terms are important: using either term alone underperforms the full SRT objective in average generation accuracy and in generate-then-revise performance. In particular, $\mathcal{L}_{\text{generation}}$ alone yields relatively weak correction after revision, while $\mathcal{L}_{\text{revision}}$ alone improves correction rate but leads to substantially worse first-attempt and overall generation accuracy, showing that the two terms play complementary roles in strengthening both generation and self-revision.

E.2.2 Applying Self-Distillation Phase Directly to the Base Model

Table 12 provides an ablation study showing what happens when we remove the SRT phase and apply the Self-Distillation phase directly to the base model. Although this Phase-2-only variant slightly improves average generation accuracy from 49.8 to 51.4, its Generate-then-Revise (see Section 3.2) behavior remains weak: first-attempt accuracy increases only modestly from 59.6 to 61.2, revised-attempt accuracy rises from 60.7 to 62.2, and the correction rate is only 2.6%, which is nearly identical to the base model's 2.7%. In contrast, models that include the SRT phase achieve much stronger revision gains, with correction rates of 15.0% for SRT and 16.7% for SD-ZERO. These results indicate that directly applying preference optimization to a model without prior self-revision training does not meaningfully improve its ability to revise incorrect solutions. Instead, the SRT phase appears to be a necessary prerequisite that first elicits self-revision behavior, after which the Self-Distillation phase can effectively refine and strengthen it.

E.2.3 Ablating Data Split Between Phases

Our method uses a fixed overall data budget that must be divided between the SRT phase and the Self-Distillation phase. This creates a natural trade-off: allocating more data to SRT may produce a

3*Method	3*Average Generation 2*Accuracy (%)	Generate-then-Revise on AIME24		
		First Attempt Accuracy (%)	Revised Attempt Accuracy (%)	Correction Rate (%)
Base Model	49.8	59.6	60.7	2.7
SRT	57.6	66.7	71.7	15.0
SD-ZERO	60.3	68.3	73.6	16.7
SD-ZERO (Phase 2 Only)	51.4	61.2	62.2	2.6

Table 12: **Ablation of applying SD-ZERO without the SRT phase.** We train a Phase-2-only variant on Qwen3-4B-Instruct by running SD-ZERO directly on the base model, without first inducing self-revision through SRT. This variant shows only marginal improvement over the base model in both overall generation accuracy and generate-then-revise performance, and its correction rate remains nearly unchanged. In contrast, the full two-phase method substantially improves both first-attempt generation and revision effectiveness. These results suggest that the SRT phase is necessary to first elicit self-revision capability, which SD-ZERO can then refine and distill into stronger reasoning.

stronger reviser, but leaves fewer examples for Phase 2 distillation; allocating too little data to SRT may instead yield a reviser that is not strong enough to provide useful supervision. To study this trade-off, we vary the split between the two phases while keeping the total amount of training data fixed, and report both the intermediate SRT model accuracy and the final SD-ZERO model accuracy in Table 13.

Data Split		Average Performance (%)	
SRT	**Self-Distillation**	**SRT Model**	**SD-ZERO Model**
6K	9K	57.6	60.3
9K	6K	57.8	59.1
7.5K	7.5K	57.8	59.8

Table 13: **Ablation of data split between SRT and Self-Distillation phases.** On Qwen3-4B-Instruct, we vary how the training data is allocated across the two phases while keeping the total data budget fixed. Allocating more data to SRT slightly improves SRT model performance, but does not lead to the best final SD-ZERO accuracy. The best overall result is achieved by assigning more data to Self-Distillation, suggesting that once SRT has sufficiently elicited self-revision capability, additional data is more effectively used in Self-Distillation phase to distill and refine this behavior into stronger generation.

F SDFT with Thinking Models

In Section 5 we noted that extending self-distillation to *thinking models*—where the student generates long, exploratory chains of thought before reaching an answer—is a natural but non-trivial next step. Below, we provide concrete evidence for this claim by applying SDFT to Qwen3-4B with thinking enabled during the student's on-policy rollouts.

Setup. We use the `Ashkchamp/Openthoughts_math_filtered_30K` dataset, taking the `solution` column as the privileged information supplied to the teacher. Training runs for one epoch with learning rate $5{\times}10^{-6}$ and batch size 64. The sole experimental variable is the `enable_thinking` flag, which controls whether the student generates with Qwen3's native thinking mode turned on or off *during SDFT training*. At evaluation, thinking is always enabled for all methods; the model may generate up to 38K completion tokens, and we sample with temperature 0.6 and top-$p = 0.95$, reporting avg@16.

Results. Table 14 shows that enabling thinking during training degrades the base model on every benchmark we tested: -9.8 points on AIME24, -10.8 on AIME25, and -8.3 on HMMT25. Disabling thinking during training largely preserves (and on HMMT25 slightly improves) the base model's accuracy.

Method	**AIME24**	**AIME25**	**HMMT25**
Qwen3-4B	0.735	0.647	0.458
+ SDFT (`enable_thinking=True`)	0.637	0.539	0.375
+ SDFT (`enable_thinking=False`)	0.733	0.616	0.466

Table 14: Effect of the student's thinking mode on SDFT training (Qwen3-4B, avg@16). `enable_thinking` refers to whether thinking is active during training; all models are evaluated with thinking enabled. Red entries denote degradation relative to the base model.

G Self-Revision Design Choices

We present complementary experiments on Countdown, a constrained arithmetic reasoning task, using Qwen2.5-7B. Although the setup differs from the main experiments (different base model, task domain, and pipeline configuration), three findings align with and corroborate the design principles of SD-ZERO: (1) on-policy self-revision data outperforms off-policy teacher data, (2) correctness filtering of revision traces is essential, and (3) self-revision training provides a stronger foundation for subsequent training than standard SFT does.

G.1 Effect of Training Data Source

We compare several training data sources for supervised fine-tuning on Countdown. For each of the 8K original SFT questions, we sample N solutions per question and optionally filter for correctness. Table 15 reports the results.

- **LLaMA-70B**: $N = 16$, pass@16 $= 81\%$, filtered $\rightarrow$ 6.5K pairs.
- **Qwen2.5-7B** (self-generated): $N = 16$, pass@16 $= 78.7\%$, filtered $\rightarrow$ 6.2K pairs.
- **GPT-4o**: $N = 5$, pass@5 $= 67.5\%$, filtered $\rightarrow$ 5.4K pairs.
- **SFT on GRPO-Qwen2.5-7B data** (on-policy): GRPO reaches pass@1 $= 91\%$ after RL; we distill its generations into 6.5–8K SFT pairs without filtering.
- **Self-Revision**: Qwen2.5-7B critiques and refines the previous response attempt; unfiltered (6.5K) and filtered (5.9K) variants.

Training Source	pass@1	pass@2	pass@4
No training	0.410	0.538	0.639
LLaMA-70B (6.5K, filtered)	0.605	0.725	0.814
Qwen2.5-7B (6.2K, filtered)	0.552	0.676	0.768
GPT-4o (5.4K, filtered)	0.617	0.711	0.775
SFT on GRPO data (8K, unfiltered)	0.878	0.904	0.922
SFT on GRPO data (6.5K, unfiltered)	0.883	0.910	0.919
Self-Revision (6.5K, unfiltered)	0.529	0.639	0.717
Self-Revision (5.9K, filtered)	0.630	0.737	0.810

Table 15: Countdown pass@k after fine-tuning Qwen2.5-7B on different data sources. On-policy sources (SFT on GRPO data, self-revision) consistently outperform off-policy ones (LLaMA-70B, GPT-4o). Among methods that do not require a prior RL stage, filtered self-revision achieves the highest pass@1.

Findings. Three patterns emerge. First, **on-policy data dominates off-policy data**: SFT on data generated by the GRPO-trained checkpoint ($\sim$0.88) substantially outperforms SFT on LLaMA-70B (0.605) or GPT-4o (0.617) data, despite comparable data sizes and correctness filtering. Note that GRPO itself reaches $\sim$0.91 via RL alone; the $\sim$0.88 figure reflects SFT distillation of its outputs. Distribution match matters more than teacher quality—the on-policy 6.5K subset performs comparably to the 8K variant. SD-ZERO also uses the student's own reviser as the teacher.

Second, **self-revision outperforms first-attempt sampling**: filtered self-revision (pass@1=0.630) exceeds the model's own correctness-filtered first attempts (pass@1=0.552). Both are on-policy, so the gain isolates the value of the revision step itself.

Third, **correctness filtering is essential**: unfiltered self-revision (0.529) underperforms the base model's first attempts, while filtering raises it to 0.630—a 19% relative gain. Noisy revision traces actively hurt. SD-ZERO's Phase 1 also retains only successful revisions (Section 2.1).

G.2 Self-Revision as Initialization for RL

We evaluate whether self-revised targets (A′) improve OOD generalization, especially when combined with RL. We fine-tune Qwen2.5-7B on 8K Countdown questions with either LLaMA-70B–generated

answers (A) or self-revised answers (A′), optionally followed by GRPO. Neither A nor A′ are filtered for correctness. Note that A uses off-policy teacher answers; Table 15 shows this is a conservative baseline, since off-policy data underperforms on-policy data even in-distribution. All evaluations use a 4096-token generation budget.

	AIME24		**AIME25**		**Countdown**		**IFBench**		**IFEval**		**MMLU**		**MMLU-Pro**	
Method	@1	@128	@1	@128	@1	@128	@1	@128	@1	@128	@1	@128	@1	@128
Base	.07	.27	.07	.43	.41	.90	.27	.58	.71	.90	.74	.91	.56	.82
GRPO (Countdown)	.08	.30	.08	.40	.91	.98	.27	.61	.72	.91	.74	.92	.56	.84
Countdown (A)	.05	.27	.05	.43	.39	.91	.23	.54	.52	.85	.69	.96	.46	.91
Countdown (A) + GRPO	.06	.23	.07	.40	.92	.98	.24	.55	.53	.84	.71	.97	.50	.93
Countdown (A′)	.07	.30	.06	.47	.40	.88	.19	.62	.60	.91	.71	.97	.50	.94
Countdown (A′) + GRPO	.07	**.40**	.07	**.47**	.91	.98	.20	.58	.62	.91	.72	.97	.52	.93

Table 16: OOD evaluation (pass@1 / pass@128) after training on Countdown with LLaMA-70B answers (A) or self-revised answers (A′), optionally followed by GRPO. A′+GRPO achieves the highest OOD scores.

The key result is that **self-revised targets provide a better initialization for RL on OOD benchmarks.** Countdown (A′)+GRPO yields pass@128 of 0.40 on AIME24 and 0.47 on AIME25, compared to 0.23 and 0.40 for Countdown (A)+GRPO, and 0.30 and 0.40 for standalone GRPO. In-distribution Countdown performance is identical across all GRPO variants (0.91/0.98).

Notably, this comparison is conservative: neither A nor A' were filtered for correctness, and A uses off-policy answers rather than the model's own attempts. Despite this, self-revised targets still yield a meaningfully stronger RL initialization on OOD benchmarks (Table 16). This suggests that the revision process itself — even without correctness filtering — carries useful signal, and that the gains would likely be larger with filtering (Table 15 shows a 19% relative gain from filtering in-distribution). Moreover, these experiments use GRPO rather than SD-ZERO's distillation objective as the second stage, suggesting the value of self-revision as initialization is not specific to a particular second-stage method.

More broadly, the fact that unfiltered self-revision provides useful signal raises the question of whether self-revision training can extend to settings without verifiable rewards, where correctness filtering is unavailable. We leave this to future work.