COMPUTATION MODULES FOR GENERATION: A DESIGN SPACE FOR LABEL-FREE SELF-DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-distillation offers a promising route to language model self-improvement, but its effectiveness depends on the supervision a model can provide for itself. While ground-truth labels are not accessible, existing approaches often construct this supervision through heuristic generation procedures, leaving open how additional computation should be organized to make the model a more informative teacher. We introduce COMPOUND COMPUTATION FOR AMPLIFICATION, a framework for organizing self-improvement into generation and learning. At its core is a *computation module for generation*, which organizes model calls and programmatic operations as a directed acyclic graph to construct supervision from unlabeled input problems. By connecting the computation used for generation, the content it produces, and subsequent learning, the framework enables generation quality and downstream training outcomes to be studied separately. Within this design space, we compare different strategies for scaling computation and find that higher answer accuracy does not consistently lead to better self-distillation. Our observations suggest that changes in response coverage and reasoning patterns may also contribute to differences in supervision quality. Motivated by this finding, we develop COMPUTATION-AUGMENTED SELF-DISTILLATION (CASD), a concrete label-free on-policy self-distillation method within our framework, whose computation module for generation is designed to construct more effective supervision. Experiments on mathematical reasoning show that CASD achieves performance comparable to or better than baselines trained with ground-truth labels and outperforms all evaluated label-free baselines. These results demonstrate the potential of organizing generation-time computation to construct more effective supervision for self-improvement.

1 INTRODUCTION

Enabling language models to continually self-improve is an important direction toward more capable AI systems. Self-improvement seeks to use a model's own capabilities and experience to support further learning. Within this direction, self-distillation offers a promising route: the model acts as both learner and teacher, using its own guidance to improve its behavior. By turning existing capabilities into a resource for subsequent training, self-distillation provides a foundation for continued self-improvement (Zhao et al., 2026a; Shenfeld et al., 2026).

Effective self-distillation requires informative supervision. However, high-quality reference solutions and external feedback are not always available, and directly generated trajectories may offer limited guidance. Recent work therefore constructs richer training experience through the model's own generation and reasoning processes. For example, He et al. (2026) uses initial responses and binary correctness feedback to support self-revision, then distills the reviser's capabilities back into the generator. Li et al. (2026) instead constructs pseudo-reference solutions from agreement among independently generated responses, enabling self-supervision without ground-truth labels. These approaches demonstrate how additional computation can produce supervision beyond what a single generation provides.

The generation procedures in these approaches are typically organized around specific heuristics, such as revising an initial response or selecting supervision through consensus among candidates. Although individually well motivated, these heuristics offer limited guidance on how to allocate and

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

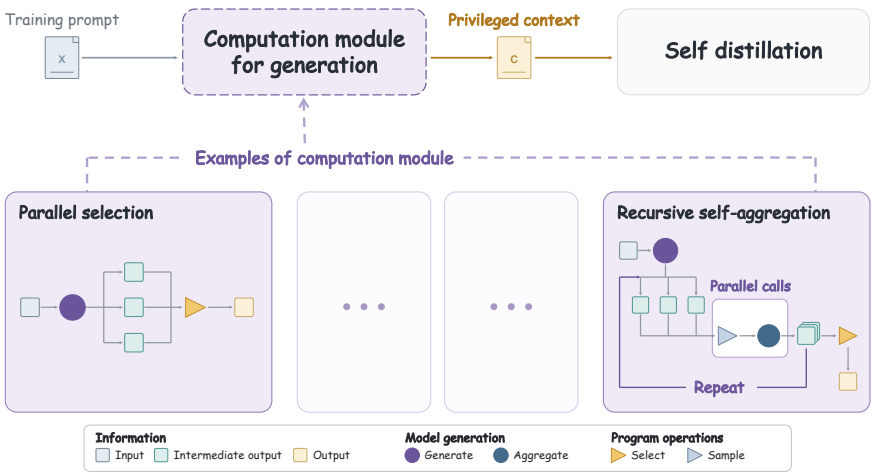

Figure 1: A self-distillation framework with computation modules for generation: compound computation for amplification.

organize additional computation, or which generation procedures yield more effective supervision for self-distillation. We therefore seek a unified perspective on their shared design problem: how can additional computation be converted into more valuable self-generated training data? This perspective makes the generation process itself an object of study, connecting how computation is organized, what it produces, and how those outputs affect subsequent learning. Our central question is:

How can a model become a more informative teacher of itself?

To study this question, we introduce COMPOUND COMPUTATION FOR AMPLIFICATION, a framework that connects data construction and subsequent learning in self-distillation (Fig. 1). At its core is a computation module for generation, which, in the label-free setting considered here, takes an input problem and combines model calls with programmatic operations to produce material for supervision. The module is represented as a directed acyclic graph, whose nodes perform operations such as generation, revision, aggregation, selection, or conditional branching, and whose edges specify data and control dependencies. The module's output serves as privileged context for the teacher, which provides dense supervision along the student's own trajectories.

The framework accommodates different compositions of model calls and programmatic operations. We focus on two representative structures: parallel scaling, which explores independent reasoning paths, and sequential scaling, which repeatedly conditions on earlier attempts. We compare their generation behavior and study how their composition affects downstream self-distillation.

Building on this comparison and prior work on aggregating candidate solutions (Venkatraman et al., 2026), we construct a computation module for generation that combines parallel exploration with iterative information integration. We then incorporate the generated material into a common on-policy self-distillation procedure to evaluate its effect on student learning. The resulting CASD student achieves performance comparable to or better than baselines trained with ground-truth labels and outperforms all evaluated label-free baselines. These results support the value of designing computational structures for supervision generation, while treating generation quality and learning outcomes as related but separately evaluated objectives.

Our contributions are threefold:

- We introduce COMPOUND COMPUTATION FOR AMPLIFICATION, a framework for studying how generation-time computation constructs supervision for self-distillation.
- We compare parallel, sequential, and hybrid computation modules to investigate the conditions on the modules for successful distillation. In particular, the study reveals that higher generation accuracy does not necessarily yield better downstream learning.
- We instantiate the framework as a label-free on-policy self-distillation method that outperforms evaluated label-free baselines and is competitive with label-supervised methods.

2 Related Work

2.1 Test-Time Scaling

Test-time scaling improves model predictions by organizing additional computation during inference. Parallel methods such as self-consistency sample diverse reasoning paths and select an answer through agreement (Wang et al., 2023), whereas sequential methods such as Self-Refine iteratively generate feedback and revise an earlier response (Madaan et al., 2023). Snell et al. (2024) study verifier-guided search and adaptive response refinement, showing that the effectiveness of inference strategies depends on problem difficulty. Hybrid methods combine exploration across candidates with information reuse across stages: RSA repeatedly aggregates subsets of a candidate population (Venkatraman et al., 2026), Parallel-Distill-Refine compresses parallel drafts into a textual workspace for subsequent refinement (Madaan et al., 2025), and MatryoshkaThinking recursively combines reasoning, verification, and summarization (Chen et al., 2025). These methods motivate the compositional design space of our computation modules for generation. Additional inference computation can also produce material for subsequent training. Yu et al. (2024) distill outputs of multi-step reasoning procedures into direct model responses, while MAGDi distills multi-agent interaction graphs into smaller models (Chen et al., 2024). Our focus is on how generation structure affects the usefulness of privileged context for label-free on-policy self-distillation.

2.2 On-Policy Distillation and Self-Distillation

On-policy distillation (OPD) trains a student on its own rollouts using dense token-level supervision from a teacher (Agarwal et al., 2024; Lu & Lab, 2025). However, standard OPD typically requires an external teacher during training. On-policy self-distillation removes this requirement by conditioning the model on privileged information, such as reference solutions, demonstrations, or feedback, to serve as its own teacher (Hübotter et al., 2026; Zhao et al., 2026a; Shenfeld et al., 2026). Recent work has improved OPD and self-distillation along several axes, including reducing supervision requirements (He et al., 2026), selecting or weighting token-level supervision (Xu et al., 2026; Liu et al., 2026b), modifying distillation targets and objectives (Yang et al., 2026; Heo et al., 2026; Shen et al., 2026), and refining training trajectories (Jiang et al., 2026). Recent label-free approaches further construct privileged context from the model's own generations, leveraging consensus across multiple samples or in-context learning from self-generated examples rather than a single unassisted generation (Li et al., 2026; Gkountouras et al., 2026; Guo et al., 2026; Chen & Li, 2026). Related approaches construct teacher guidance through error-localized reflection in ROSD (Zhao et al., 2026d), reflection-conditioned scores in SRPO (Liu et al., 2026a), and draft–verify or plan–solve workflows in on-policy harness self-distillation (Zhao et al., 2026c).

3 Compound Computation for Amplification

Framework. We describe self-improvement through two components: a *computation module for generation* $\mathcal{G}$ that constructs generated material, and a learning procedure $\mathcal{U}$ that uses this material to update the model. Let $\mathcal{D} = \{x_i\}_{i=1}^{n}$ be a dataset of input problems and π_θ a policy that generates a response $y \sim \pi_\theta(\cdot \mid x)$. The module $\mathcal{G}$ uses this policy to produce generation records $\mathcal{B}$, which $\mathcal{U}$ uses to obtain updated parameters θ'. We call this composition *compound computation for amplification* (Algorithm 1).

Computation modules for generation. A computation module for generation combines policy calls with programmatic operations. We represent these operations as a finite directed acyclic graph: nodes specify operations, and edges specify data and control dependencies. Operations may generate or revise responses, aggregate candidates, extract answers, or select generated material; conditional nodes determine which operations are executed and when generation terminates. The records $\mathcal{B}$ retain intermediate

Algorithm 1 Compound Computation for Amplification

Input: Problem dataset $\mathcal{D}$, policy π_θ, computation module for generation $\mathcal{G}$, learning procedure $\mathcal{U}$
Output: Updated policy $\pi_{\theta'}$

1: $\mathcal{B} \sim \mathcal{G}(\cdot \mid \mathcal{D}; \pi_\theta)$
2: $\theta' \leftarrow \mathcal{U}(\theta; \mathcal{B})$
3: **return** $\pi_{\theta'}$

and final outputs, which may be filtered, reordered, combined, or rewritten within $\mathcal{G}$ to construct material for learning. This interface accommodates different learning procedures, including supervised fine-tuning (Chen et al., 2024), reinforcement learning (Zuo et al., 2025), and on-policy self-distillation (Zhao et al., 2026a).

Our research question. Within this framework, we study how the design of $\mathcal{G}$ affects the usefulness of generated material for a fixed learning procedure $\mathcal{U}$. Our study considers unlabeled problems: crucially, we do not assume access to ground-truth answers or solutions. During generation, the policy is fixed, and every operation uses only the input problem and internally generated information, without external correctness feedback. We examine two questions separately: whether a computation module for generation produces more accurate responses, and whether its outputs provide more effective supervision for student learning. Higher generation accuracy need not imply better learning outcomes; their relationship is part of the empirical question.

Instantiation with on-policy self-distillation. To study these questions, we instantiate $\mathcal{U}$ by optimizing the on-policy self-distillation objective of Zhao et al. (2026a), using generated privileged contexts in place of ground-truth solutions. For each problem x, the computation module for generation constructs a context c from $\mathcal{B}$; this selection or transformation remains part of $\mathcal{G}$. In our instantiation, the teacher and student start from the same checkpoint, and the teacher parameters $\bar{\theta}$ remain fixed during training. The student samples $y \sim \pi_\theta(\cdot \mid x)$, while the teacher additionally receives c and supplies token-level supervision along the same response. Writing $(x, c) \sim \mathcal{B}$ for sampling a problem and its constructed context from the records, we use the forward KL objective

$$\mathcal{L}_{\mathrm{OPSD}}(\theta; \mathcal{B}, \bar{\theta}) = \mathbb{E}_{\substack{(x,c)\sim\mathcal{B} \\ y\sim\pi_\theta(\cdot|x)}} \left[\sum_{t=1}^{|y|} \mathrm{KL}(\pi_{\bar{\theta}}(\cdot \mid x, c, y_{<t}) \,\|\, \pi_\theta(\cdot \mid x, y_{<t})) \right]. \tag{1}$$

Only the student parameters are updated. This fixes the learning objective while leaving the computation module for generation open to investigation: Section 4 studies generation quality and downstream learning, and Section 5 presents a concrete method within this framework.

4 An Empirical Study of Computation Modules for Generation

Within *compound computation for amplification*, how should a computation module for generation construct material that is useful for self-distillation? A natural starting point is answer correctness: additional generation computation may produce more accurate responses, which can then provide privileged information to the teacher, as in consensus-based self-distillation (Li et al., 2026; Gkountouras et al., 2026). However, a generation procedure can change not only correctness, but also the coverage and content of its outputs. We therefore examine generation quality and downstream learning separately. We first compare parallel sampling with sequential revision and select a promising hybrid computation module for generation through generation-level evaluation. We then study how its outputs evolve across stages and whether these changes translate into better self-distillation.

4.1 Parallel Sampling Outperforms Sequential Revision in Success Coverage

We first investigate whether additional generation is more effective at discovering correct solutions through independent sampling or through revision of existing responses. These two patterns appear in consensus-based and revision-based self-distillation, respectively (Li et al., 2026; He et al., 2026). Our comparison measures success coverage: the fraction of problems for which the procedure produces at least one correct answer.

We start from one or four independently sampled responses and compare sequential revision with a fourfold increase in parallel sampling. Following the revision approach of Self-Distillation Zero (He et al., 2026), we append a correction prompt when an evaluation checker identifies an incorrect response, asking the model to reconsider the problem, identify errors in its previous reasoning, and produce a corrected answer in the required format. We continue revision until a correct answer is found or the round or context limit is reached; the full prompt is provided in Appendix C.1. This use of correctness feedback makes revision a diagnostic comparison rather than an instance of our label-free computation module for generation. Including the initial response, revision uses an average

of 5.43 model calls for Qwen3-4B and 3.86 for Qwen3-4B-Instruct-2507. These counts describe the evaluated configurations and do not imply an exact match in computational cost.

Table 1 shows that independent sampling produces substantially larger coverage gains in these settings. For Qwen3-4B, revising four initial responses increases coverage from 22.37% to 22.73%, whereas expanding to sixteen independent responses raises it to 37.27%. The corresponding results for Qwen3-4B-Instruct-2507 are 52.44%, 52.73%, and 64.55%. Thus, continuing existing attempts provides relatively little additional coverage, while exploring new trajectories more often discovers a correct solution. This finding motivates computation modules for generation that retain parallel exploration and raises the next question: how can information from multiple candidates be combined?

Table 1: Success coverage (%) of direct sampling, sequential revision, and expanded parallel sampling.

Initial samples	Direct	+ Revision	4× Parallel
Qwen3-4B			
1	11.82	12.95	**22.37**
4	22.37	22.73	**37.27**
Qwen3-4B-Instruct-2507			
1	34.09	39.32	**52.44**
4	52.44	52.73	**64.55**

4.2 Selecting a Computation Module for Generation through Generation Accuracy

Hybrid computation modules for generation combine parallel exploration with sequential information reuse. As an initial selection criterion, we compare the generation accuracy of three representative structures: Recursive Self-Aggregation (RSA) (Venkatraman et al., 2026), Parallel-Distill-Refine (PDR) (Madaan et al., 2025), and MatryoshkaThinking (Chen et al., 2025).

These computation modules for generation differ in how information passes between stages. RSA constructs a new candidate population by aggregating subsets of the preceding population. PDR compresses parallel drafts into a textual workspace that conditions subsequent refinement. MatryoshkaThinking recursively combines generation, verification, and summarization. Figure 2 illustrates the configurations evaluated in our study.

Table 2: Generation accuracy (%) for three hybrid computation modules for generation.

Module	AIME 2025	HMMT 2025	Overall
RSA	66.04	39.79	52.92
PDR	60.00	36.67	48.33
MatryoshkaThinking	53.33	36.67	45.00

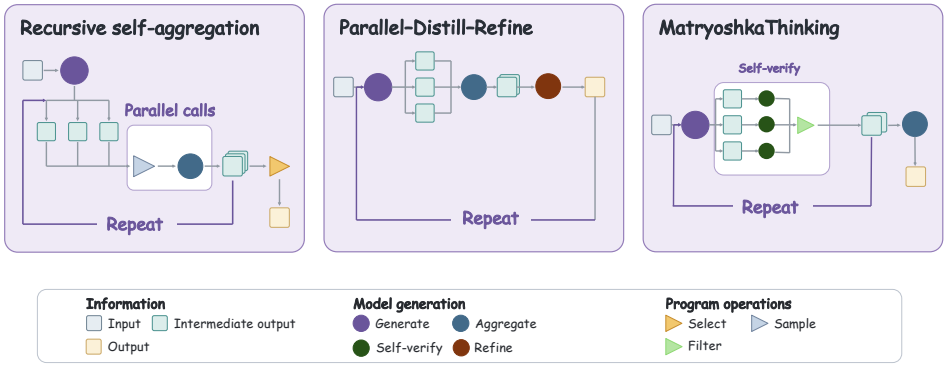

Figure 2: Hybrid computation modules for generation. Left: Recursive Self-Aggregation. Middle: Parallel-Distill-Refine. Right: MatryoshkaThinking.

We use sixteen initial candidates for each computation module for generation. RSA aggregates subsets of four candidates for five recursive rounds; PDR applies distill–refine transitions with widths $16 \to 8 \to 4$; and MatryoshkaThinking samples sixteen candidates per loop for three verify–summarize rounds. These configurations retain each method's own aggregation or compression rule.

Among these configurations, RSA achieves the highest reported generation accuracy on both AIME 2025 and HMMT 2025, with an overall score of 52.92%, compared with 48.33% for PDR and 45.00% for MatryoshkaThinking (Table 2). This comparison identifies RSA as a promising concrete instance for further investigation. Its sequence of candidate populations also allows us to examine how generated material changes across aggregation stages. We therefore use RSA for the following analyses, while leaving its value for self-distillation to be evaluated separately.

4.3 Aggregation Improves Average Accuracy but Reduces Coverage

A higher average correctness rate does not fully characterize a generated population. We examine two complementary properties: avg@16, the average correctness of sixteen responses per problem, and pass@16, the fraction of problems with at least one correct response. We additionally measure response length and the frequency of reflection markers to characterize changes in the generated trajectories.

We sample 200 problems from each of OpenR1 (Lozhkov et al., 2025), OpenThoughts (Guha et al., 2025), and MATH (Hendrycks et al., 2021), and track these properties from the initial population through three RSA aggregation stages. Reflection-marker frequency is normalized per 10,000 generated tokens; the marker list is provided in Appendix A.1.

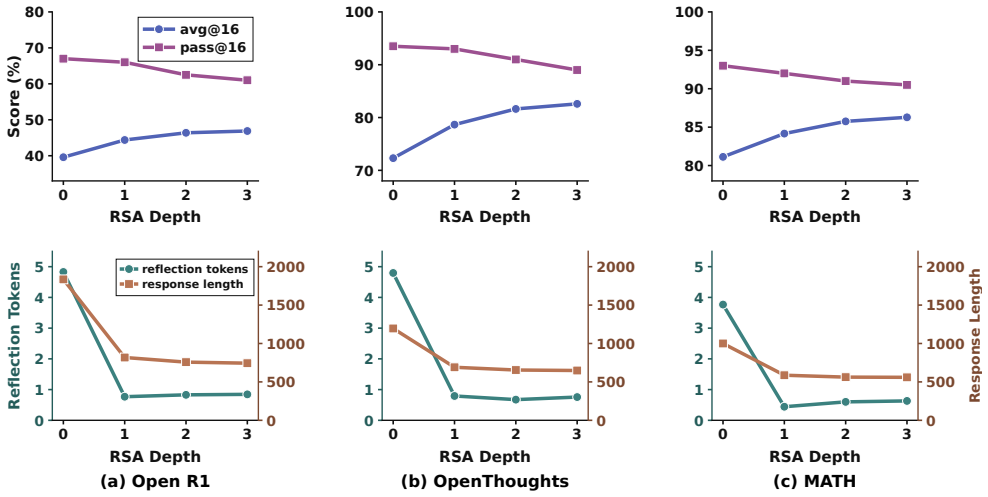

Figure 3: Generation properties across RSA stages on OpenR1, OpenThoughts, and MATH. Top: average population accuracy (avg@16) rises while success coverage (pass@16) declines. Bottom: reflection-marker frequency per 10,000 generated tokens and average response length fall sharply after the initial generation stage.

Figure 3 reveals a consistent divergence: avg@16 increases with aggregation depth, while pass@16 decreases. Later populations therefore contain a higher average proportion of correct responses, yet cover fewer problems with at least one correct solution. Aggregation also changes the form of the responses: length and reflection-marker frequency fall sharply after the first stage, with smaller subsequent changes.

These observations show that aggregation improves one aspect of generation quality while reducing another and substantially altering trajectory content. The observed changes in length and reflection markers do not by themselves establish a loss of reasoning capability. They nevertheless motivate evaluating whether later-stage material, despite its higher average correctness, provides better supervision for learning.

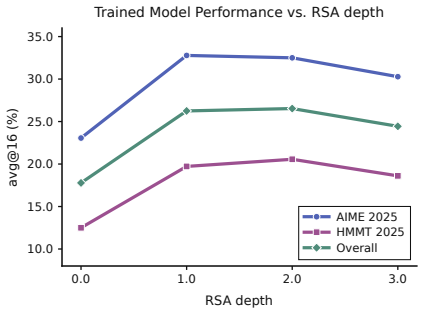

Figure 4: Student avg@16 accuracy (%) after self-distillation with privileged context from different RSA depths.

4.4 Higher Generation Accuracy Does Not Consistently Improve Self-Distillation

We next examine how the stage from which privileged context is extracted affects the trained student. Using 10,000 OpenR1 problems (Lozhkov et al., 2025), we construct RSA generation records and compare self-distillation with contexts selected from different depths through majority-based selection. Generation stops early when all candidate responses yield valid and identical final answers. The downstream training procedure is held fixed across these variants.

Figure 4 shows that student performance initially improves and then declines. Thus, the deepest context is not the most effective, while the initial population also provides substantially weaker supervision than the intermediate stages. We also measure the accuracy of the responses selected as privileged context for each depth. Figure 5 compares this context accuracy with the resulting student's overall avg@16. The selected contexts become more accurate at every successive depth, whereas student performance peaks at depth 2 and declines at depth 3. Thus, the decline occurs even as the accuracy of the privileged context itself continues to improve. This comparison shows that selecting more accurate contexts alone does not consistently yield more effective self-distillation.

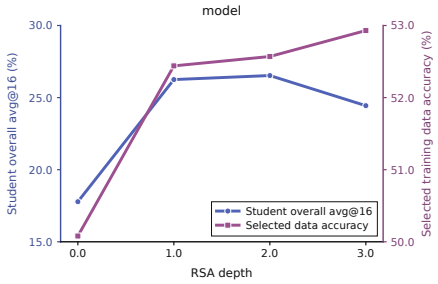

Figure 5: Accuracy of selected privileged contexts for training (right axis) and student overall avg@16 (left axis) across RSA depths. Both axes show percentages.

The generation-level analysis in Section 4.3 suggests possible contributing factors, including reduced success coverage and changes in trajectory content. Related evidence that self-distillation can suppress uncertainty expression and degrade reasoning further motivates examining these behavioral changes (Kim et al., 2026).

Implication. These findings motivate separating two decisions: which answer the generated evidence supports, and which trajectory should serve as the teacher's privileged context. In Section 5, we instantiate this idea by using the terminal population to select a pseudo-answer and retrieving a matching trajectory from the initial population, with a fallback to the terminal population when no initial trajectory matches.

5 Computation-Augmented Self-Distillation

Section 4 shows that higher generation accuracy does not consistently translate into better self-distillation. Motivated by this finding, we introduce *Computation-Augmented Self-Distillation* (CASD), a concrete instance of *compound computation for amplification* that separates answer selection from privileged-context selection. Algorithm 2 in the appendix summarizes the complete procedure. Here we first describe its computation module for generation and then evaluate the resulting student on mathematical reasoning benchmarks.

5.1 Decoupling Answer Selection from Context Selection

CASD uses aggregated responses to select a pseudo-answer and preferentially uses a matching response from the initial population as the teacher's privileged context. The motivation is that aggregation changes both answer accuracy and the content of the reasoning trajectory, as observed in Section 4. These two roles could be served by different responses: the terminal population can determine which answer to retain, while a matching initial response provides a different reasoning path to that answer.

Constructing the privileged context. For each problem x, the computation module for generation first samples 16 direct responses to form the initial population T_0 and then applies Recursive Self-Aggregation (RSA) (Venkatraman et al., 2026) for at most three additional stages. Each aggregation stage generates 16 responses, with each response conditioned on a subset of four responses from the preceding stage. Generation stops when all 16 responses yield valid and identical final answers or

when the maximum of four stages is reached. We select a pseudo-answer by majority voting over the terminal population.If multiple answers tie for the most votes, we select one uniformly at random. We uniformly select the privileged context c from responses in T_0 whose final answer matches the terminal pseudo-answer. If T_0 contains no matching response, we instead uniformly select a matching response from the terminal population. The only candidate sources for context selection are T_0 and the terminal population; intermediate stages are not searched.

Distilling into the student. Given the selected contexts, we apply the on-policy self-distillation procedure from Section 3 using Eq. (1), following the training objective match (Zhao et al., 2026a). The fixed teacher receives the privileged context with thinking enabled; the student receives only the problem and operates without thinking during both training and evaluation.

5.2 Main Evaluation

Table 3: Avg@12 accuracy (%) for Qwen3 models on mathematical reasoning benchmarks in non-thinking mode. Without ground-truth supervision, CASD achieves performance comparable to or better than baselines that use ground-truth labels and outperforms all evaluated label-free baselines. Bold and underlined entries denote the highest and second-highest scores, respectively.

	Method	AIME 2025	AIME 2026	HMMT 2025	HMMT 2026	AMC 2023	MATH 500	Apex-Shortlist
Qwen3-8B	Base	20.56	15.56	9.72	17.42	69.38	86.83	3.19
w. label	SFT	28.06	32.78	17.50	17.93	75.83	85.17	0.71
	GRPO	<u>46.67</u>	50.83	28.06	<u>35.35</u>	<u>91.25</u>	**95.75**	4.96
	RFT	21.39	18.33	10.56	17.17	68.75	85.58	4.96
	OPSD	46.39	<u>51.11</u>	<u>29.44</u>	31.31	88.96	91.42	<u>5.85</u>
w.o. label	TTRL	21.94	20.83	13.61	20.96	72.08	89.08	3.72
	Intuitor	15.28	12.78	7.50	10.86	59.38	80.25	3.19
	CASD	**57.78**	**67.50**	**42.50**	**42.42**	**93.75**	<u>94.83</u>	**9.04**
Qwen3-4B	Base	17.22	18.06	11.94	17.42	67.71	85.25	4.08
w. label	SFT	25.83	25.28	11.94	14.90	73.13	79.50	0.53
	GRPO	<u>50.83</u>	<u>47.50</u>	<u>31.94</u>	<u>31.06</u>	**90.00**	**94.00**	<u>4.96</u>
	RFT	18.89	13.61	12.50	15.91	67.71	85.33	3.19
	OPSD	35.00	37.78	21.67	25.76	66.04	80.92	3.72
w.o. label	TTRL	20.28	15.56	12.50	15.66	67.71	86.17	3.19
	Intuitor	15.83	10.56	9.72	6.57	56.88	68.25	1.95
	CASD	**54.17**	**53.06**	**33.06**	**34.60**	<u>89.38</u>	<u>91.83</u>	**6.03**

Models and training data. We evaluate Qwen3-4B and Qwen3-8B (Yang et al., 2025), initializing the teacher and student from the same checkpoint for each model size. SFT, RFT, GRPO, OPSD, and CASD use the same subset of 10,000 problems from Open R1 (Lozhkov et al., 2025) as their source of prompts. CASD uses only the problem statements and constructs its privileged contexts without accessing the dataset's answers or solutions.

Baselines and supervision. We compare against the base model and four baselines that use ground-truth supervision: SFT, RFT, GRPO, and OPSD (Zhao et al., 2026a). For SFT, we sample three trajectories per problem from DeepSeek-V4-Flash, filter them by correctness, and retain at most one trajectory per problem, yielding approximately 6,000 training trajectories. GRPO uses binary outcome rewards verified against the ground-truth answer, while OPSD provides the ground-truth solution as the teacher's privileged context. We also compare against two label-free baselines, TTRL (Zuo et al., 2025) and Intuitor (Zhao et al., 2026b). These two groups test whether internally generated supervision can compete with ground-truth supervision and whether CASD improves over existing methods that likewise do not require ground-truth labels.

Evaluation and benchmarks. We evaluate on seven mathematical reasoning benchmarks: AIME 2025 (Zhang & Math-AI, 2025), AIME 2026 (Zhang & Math-AI, 2026), HMMT 2025 (Balunović

et al., 2026), HMMT 2026 (Dekoninck et al., 2026), AMC 2023 (Mathematical Association of America, 2023), MATH500 (Hendrycks et al., 2021), and Apex-Shortlist (Dekoninck et al., 2026). For MATH500, we use a fixed random subset of 100 problems. All evaluations use non-thinking mode, and we report Avg@12 accuracy, averaged over 12 sampled responses per problem. We train SFT for three epochs and evaluate after each epoch; all other methods are trained for 200 steps and evaluated every 50 steps. For each method and model size, we select a single checkpoint with the highest Avg@12 accuracy on AIME 2025 and then evaluate that checkpoint on the remaining benchmarks. Table 3 reports results from this single selected checkpoint for each method and model size.

Results. Without using ground-truth labels, CASD achieves performance comparable to or better than label-supervised baselines and outperforms all evaluated label-free baselines (Table 3).

Against ground-truth-supervised baselines, CASD achieves the highest accuracy on six of seven benchmarks for Qwen3-8B and five of seven for Qwen3-4B, while outperforming ground-truth-conditioned OPSD on all seven benchmarks at both model sizes. On AIME 2026, for example, CASD improves over the strongest label-supervised baseline from 51.11% to 67.50% for Qwen3-8B and from 47.50% to 53.06% for Qwen3-4B. On the few remaining benchmarks, it remains close to GRPO, with gaps of at most 0.92 points for Qwen3-8B and 2.17 points for Qwen3-4B.

Among label-free methods, CASD outperforms both TTRL and Intuitor at both model sizes. Together, the two comparisons show that this concrete instance of compound computation for amplification produces effective supervision from the model's own computation: it improves over existing label-free approaches while achieving performance competitive with, and often superior to, methods that access ground-truth labels.

5.3 Ablation Study

We compare CASD with two coupled selection strategies under the same training and evaluation protocol. Initial consensus selects both the pseudo-answer and a matching context from the initial population, while terminal consensus selects both from the terminal population. CASD and terminal consensus share the same generation records and terminal pseudo-answers, differing only in how the privileged context is selected. CASD outperforms both alternatives on all seven benchmarks.

Table 4: Avg@12 accuracy (%) for Qwen3-4B across mathematical reasoning benchmarks in non-thinking mode. We compare supervision constructed from the initial and terminal consensus with CASD. Bold and underlined entries denote the highest and second-highest scores, respectively.

Method	AIME 2025	AIME 2026	HMMT 2025	HMMT 2026	AMC 2023	MATH 500	Apex-Shortlist
Initial consensus	24.17	24.44	14.44	15.66	75.42	85.42	<u>4.43</u>
Terminal consensus	<u>26.94</u>	<u>29.17</u>	<u>18.89</u>	<u>20.71</u>	<u>77.50</u>	<u>89.42</u>	4.08
CASD	**54.17**	**53.06**	**33.06**	**34.60**	**89.38**	**91.83**	**6.03**

6 Conclusion

We introduced Compound Computation for Amplification, a framework that separates the construction of generated material from the learning procedure that uses it. Within this framework, we studied how the design of a computation module for generation affects both generation quality and downstream self-distillation. Our empirical study shows that higher answer accuracy alone is insufficient to characterize supervision quality. Motivated by this finding, we developed Computation-Augmented Self-Distillation (CASD), which separates answer selection from privileged-context selection. On mathematical reasoning benchmarks, CASD achieves performance comparable to or better than baselines trained with ground-truth labels and outperforms all evaluated label-free baselines. These results highlight the value of designing generation-time computation for effective downstream learning. Our study focuses on mathematical reasoning with on-policy self-distillation. A natural next step is to identify properties of generated reasoning that predict its usefulness for learning beyond answer correctness, and to investigate whether these properties generalize across tasks and learning procedures.

AI USE STATEMENT

In this work, we used generative AI tools to polish the language of the manuscript. We have reviewed all AI-assisted work and take full responsibility for the final content of this work.

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, 2024. URL https://arxiv.org/abs/2306.13649.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions, 2026. URL https://arxiv.org/abs/2505.23281.

Hongwei Chen, Yishu Lei, Dan Zhang, Bo Ke, Danxiang Zhu, Xuyi Chen, Yuxiang Lu, Zhengjie Huang, Shikun Feng, Jingzhou He, Yu Sun, Hua Wu, and Haifeng Wang. Matryoshkathinking: Recursive test-time scaling enables efficient reasoning, 2025. URL https://arxiv.org/abs/2510.10293.

Justin Chih-Yao Chen, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. Magdi: Structured distillation of multi-agent interaction graphs improves reasoning in smaller language models, 2024. URL https://arxiv.org/abs/2402.01620.

Zhuowei Chen and Xiang Lorraine Li. Neuron-aware data selection for annotation-free llm self-distillation, 2026. URL https://arxiv.org/abs/2607.02460.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms, 2026. URL https://arxiv.org/abs/2605.00674.

John Gkountouras, Josip Jukić, and Ivan Titov. Consensus as privileged context for label-free self-distillation, 2026. URL https://arxiv.org/abs/2607.13643.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

Jiaxin Guo, Yanwei Yue, Xuanbo Fan, Chunyu Yang, and Yan Zhang. Learning from consensus and disagreement: Unsupervised on-policy self-distillation with minority-trajectory contrast, 2026. URL https://arxiv.org/abs/2608.08764.

Yinghui He, Simran Kaur, Adithya Bhaskar, Yongjin Yang, Jiarui Liu, Narutatsu Ri, Liam Fowl, Abhishek Panigrahi, Danqi Chen, and Sanjeev Arora. Self-distillation zero: Self-revision turns binary rewards into dense supervision, 2026. URL https://arxiv.org/abs/2604.12002.

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.

Byeongho Heo, Jaehui Hwang, Sangdoo Yun, and Dongyoon Han. On-policy delta distillation, 2026. URL https://arxiv.org/abs/2607.15161.

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

Li Jiang, Haoran Xu, Yichuan Ding, and Amy Zhang. Trajectory-refined distillation, 2026. URL https://arxiv.org/abs/2606.08432.

Jeonghye Kim, Xufang Luo, Minbeom Kim, Sangmook Lee, Dohyung Kim, Jiwon Jeon, Dongsheng Li, and Yuqing Yang. Why does self-distillation (sometimes) degrade the reasoning capability of LLMs? In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=Az7guis46K.

Yijiang Li, Bingyang Wang, Yijun Liang, Yunjie Tian, Di Fu, and Nuno Vasconcelos. On-policy self-distillation without any supervision, 2026. URL https://arxiv.org/abs/2608.06296.

Jialong Liu, Yuling Shi, Ning Yang, Xiaodong Gu, and Zuchao Li. Srpo: Self-reflective policy optimization for long-horizon reasoning, 2026a. URL https://arxiv.org/abs/2608.23493.

Xiaogeng Liu, Xinyan Wang, Yingzi Ma, Yechao Zhang, and Chaowei Xiao. When are teacher tokens reliable? position-weighted on-policy self-distillation for reasoning, 2026b. URL https://arxiv.org/abs/2605.21606.

Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Guilherme Penedo, Edward Beeching, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Openr1-math-220k. https://huggingface.co/datasets/open-r1/OpenR1-Math-220k, 2025.

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, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Lovish Madaan, Aniket Didolkar, Suchin Gururangan, John Quan, Ruan Silva, Ruslan Salakhutdinov, Manzil Zaheer, Sanjeev Arora, and Anirudh Goyal. Rethinking thinking tokens: Llms as improvement operators, 2025. URL https://arxiv.org/abs/2510.01123.

Mathematical Association of America. Amc resources, 2023. URL https://www.maa.org/math-competitions/amc-resources.

Guobin Shen, Xiang Cheng, Chenxiao Zhao, Lei Huang, Jindong Li, Dongcheng Zhao, and Xing Yu. Anti-self-distillation for reasoning rl via pointwise mutual information, 2026. URL https://arxiv.org/abs/2605.11609.

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

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

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan Obando-Ceron, Yoshua Bengio, Brian R. Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, Nikolay Malkin, and Moksh Jain. Recursive self-aggregation unlocks deep thinking in large language models, 2026. URL https://arxiv.org/abs/2509.26626.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Yuanda Xu, Hejian Sang, Zhengze Zhou, Ran He, Zhipeng Wang, and Alborz Geramifard. Tip: Token importance in on-policy distillation, 2026. URL https://arxiv.org/abs/2604.14084.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, 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.

Shenzhi Yang, Guangcheng Zhu, Bowen Song, Haobo Wang, Mingxuan Xia, Xing Zheng, Yingfan Ma, Zhongqi Chen, Weiqiang Wang, Junbo Zhao, and Gang Chen. Oprd: On-policy representation distillation, 2026. URL https://arxiv.org/abs/2606.06021.

Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. Distilling system 2 into system 1, 2024. URL https://arxiv.org/abs/2407.06023.

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

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

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, 2026a. URL https://arxiv.org/abs/2601.18734.

Xuandong Zhao, Zhewei Kang, Aosong Feng, Sergey Levine, and Dawn Song. Learning to reason without external rewards, 2026b. URL https://arxiv.org/abs/2505.19590.

Zhengyang Zhao, Lu Ma, and Wentao Zhang. Training with harnesses: On-policy harness self-distillation for complex reasoning, 2026c. URL https://arxiv.org/abs/2605.08741.

Ziqi Zhao, Xinyu Ma, Liu Yang, Yujie Feng, Daiting Shi, Jingzhou He, Xin Xin, Zhaochun Ren, and Xiao-Ming Wu. Rosd: Reflective on-policy self-distillation for language model reasoning across domains, 2026d. URL https://arxiv.org/abs/2605.28014.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. Ttrl: Test-time reinforcement learning, 2025. URL https://arxiv.org/abs/2504.16084.

Appendix

CONTENTS

1 Introduction **1**

2 Related Work **3**

2.1 Test-Time Scaling 3

2.2 On-Policy Distillation and Self-Distillation 3

3 Compound Computation for Amplification **3**

4 An Empirical Study of Computation Modules for Generation **4**

4.1 Parallel Sampling Outperforms Sequential Revision in Success Coverage 4

4.2 Selecting a Computation Module for Generation through Generation Accuracy . . 5

4.3 Aggregation Improves Average Accuracy but Reduces Coverage 6

4.4 Higher Generation Accuracy Does Not Consistently Improve Self-Distillation . . . 7

5 Computation-Augmented Self-Distillation **7**

5.1 Decoupling Answer Selection from Context Selection 7

5.2 Main Evaluation 8

5.3 Ablation Study 9

6 Conclusion **9**

A Experimental Details for Empirical Studies of Computation Modules for Generation **14**

A.1 Reflection Markers 14

B Experimental Details for Training **14**

B.1 CASD Algorithm 14

B.2 Hyperparameters 15

B.3 Data Details 20

C Prompt Templates **21**

C.1 Sampling Prompts 21

C.2 DAG-Based Generation Prompts 21

C.3 Teacher and Student Prompts for OPSD 24

A Experimental Details for Empirical Studies of Computation Modules for Generation

A.1 Reflection Markers

We measure reflection using occurrences of the following 20 lexical markers:

> `wait`, `actually`, `hmm`, `let me reconsider`, `let me rethink`, `i made an error`, `i made a mistake`, `that's wrong`, `that is wrong`, `incorrect`, `reconsider`, `rethink`, `re-examine`, `let me check`, `let me verify`, `double check`, `double-check`, `going back`, `revisit`, `on second thought`.

We match these markers case-insensitively in generated completion text, using word boundaries, allowing whitespace variation within phrases, and normalizing curly apostrophes to straight apostrophes. Each matched phrase contributes one marker, irrespective of its tokenizer length. These counts are lexical proxies for reflection, rather than annotations of semantic self-correction.

Let M denote the total number of matched markers, L the total number of completion tokens recorded by the inference server, and N the number of generated responses. We report reflection markers per 10,000 completion tokens ($10{,}000M/L$) and mean response length (L/N). Prompt tokens are excluded.

B Experimental Details for Training

B.1 CASD Algorithm

Algorithm 2 Computation-Augmented Self-Distillation (CASD)

Input: Unlabeled problems $\mathcal{D}$, initial policy π_θ
Output: Trained student π_θ

$\bar{\theta} \leftarrow \theta$; $\mathcal{B} \leftarrow \varnothing$
for each $x \in \mathcal{D}$ **do**
  Sample T_0: 16 responses from $\pi_{\bar{\theta}}(\cdot \mid x)$; $d \leftarrow 0$
  while $d < 3$ and $\neg\,\mathrm{Consensus}(T_d)$ **do**
    $T_{d+1} \leftarrow \mathrm{RSA}(x, T_d; \pi_{\bar{\theta}})$; $d \leftarrow d + 1$
  end while
  $\hat{a} \leftarrow \mathrm{Vote}(T_d)$
  $C \leftarrow \{y \in T_0 : a(y) = \hat{a}\}$
  if $C = \varnothing$ **then**
    $C \leftarrow \{y \in T_d : a(y) = \hat{a}\}$
  end if
  Sample c uniformly from C
  Add $(x, T_0, \ldots, T_d, c)$ to $\mathcal{B}$
end for
for $m = 1, \ldots, 200$ **do**
  Sample a minibatch of (x, c) from $\mathcal{B}$ and student responses $y \sim \pi_\theta(\cdot \mid x)$
  Update θ using Eq. (1), keeping $\bar{\theta}$ fixed
end for
return π_θ

Here, $a(y)$ extracts the final answer from response y. $\mathrm{Consensus}$ requires all responses to have valid and identical answers; Vote selects the most frequent answer, breaking ties uniformly at random. Each RSA call generates 16 new responses, each conditioned on four sampled responses from the preceding population.

B.2 HYPERPARAMETERS

Table 5: SFT hyperparameters. Model-specific settings are explicitly marked.

Parameter	SFT
General	
Model	Qwen3-4B / Qwen3-8B
Thinking	Off
Fine-tuning	Full-parameter
LoRA	False
Data	
Training targets	Verified DeepSeek reasoning + answer
Training examples	6,215
Max. sequence length	32,768
Shuffle	True
Sequence packing	False
Batching	
Effective global batch size	64
Micro-batch size per GPU	1
Gradient accumulation steps	8
Objective	
Loss	Completion-only cross-entropy
Prompt loss	Masked
Padding loss	Masked
Reference model	None
Parallelism	
Number of GPUs	8
Number of nodes	1
Strategy	FSDP full-shard
Gradient checkpointing	True
Attention implementation	FlashAttention-2
Liger kernel	True
Training	
Optimizer	Fused AdamW
Learning rate	5×10^{-6}
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	10^{-8}
Scheduler	Cosine
Warmup ratio	0.05
Weight decay	10^{-4}
Gradient clip norm	1.0
Precision	BF16 mixed precision
TF32	True
Epochs	3
Total optimizer steps	294
Checkpoint steps	98, 196, 294
Seed	42

Table 6: RFT hyperparameters. Model-specific settings are explicitly marked.

Parameter	RFT
General	
Model	Qwen3-4B / Qwen3-8B
Thinking	Off
Fine-tuning	Full-parameter
LoRA	False
Data	
Training examples	4B: 7,242; 8B: 7,172
Max. sequence length	32,768
Separate prompt / response limits	Not set; combined sequence limit
Shuffle	True
Sequence packing	False
Batching	
Effective global batch size	32
Micro-batch size per GPU	1
Gradient accumulation steps	4
Objective	
Loss	Completion-only cross-entropy
Prompt loss	Masked
Padding loss	Masked
Reference model	None
Parallelism	
Number of GPUs	8
Number of nodes	1
Strategy	FSDP full-shard
Gradient checkpointing	True
Attention implementation	FlashAttention-2
Liger kernel	True
Training	
Optimizer	Fused AdamW
Learning rate	5×10^{-6}
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	10^{-8}
Scheduler	Cosine
Warmup ratio	0.05
Weight decay	0.0
Gradient clip norm	1.0
Precision	BF16 mixed precision
TF32	True
Training schedule	Fixed optimizer-step budget
Total optimizer steps	200
Checkpoint steps	50, 100, 150, 200
Seed	42

Table 7: GRPO hyperparameters. Model-specific settings are explicitly marked.

Parameter	GRPO
General	
Model	Qwen3-4B / Qwen3-8B
Thinking	Off
Data	
Max. prompt length	24,576
Max. response length	8,192
Train batch size	32
Shuffle	True
Batching	
Rollouts per prompt (n)	8
PPO mini-batch size	32
PPO micro-batch size per GPU	1
Logprob micro-batch size per GPU	1
Max. concurrent sequences per engine	32
Max. batched tokens per engine	8,192
Rollout / Generation	
Inference engine	vLLM
Rollout mode	Async
Temperature	1.0
Top-p	1.0
Top-k	-1
Sampling	True
Max. new tokens	8,192
Tensor parallel size	1
Enable chunked prefill	True
Actor / Policy Optimization	
Strategy	FSDP full-shard
PPO epochs	1
Clip ratio	0.2
Entropy coefficient	0
Use KL loss	False
Gradient clip norm	1.0
Dynamic batch size	False
Parallelism	
Number of GPUs	8
Number of nodes	1
Sequence parallel size	1
Actor strategy	FSDP full-shard
Reference strategy	disabled
Gradient checkpointing	True
Training	
Optimizer	AdamW
Learning rate	5×10^{-6}
Scheduler	Linear warmup, then constant
Warmup steps	20
Weight decay	0.01
Precision	BF16 mixed precision
NCCL timeout	600
Total training steps	200
Checkpoint interval	50

Table 8: OPSD hyperparameters with a KL supervision length of 128. Model-specific settings are explicitly marked.

Parameter	OPSD and CASD
General	
Model	Qwen3-4B / Qwen3-8B
Initialization	Original pretrained checkpoint
Student thinking	Off
Teacher thinking	On
Fine-tuning	Full-parameter
LoRA	False
Data	
Dataset	OpenR1-Math, shared 10k subset
Training questions	10,000
Student correctness filtering	False
Max. rollout prompt length	24,576
Max. rollout response length	8,192
Max. distillation sequence length	32,768
KL supervision length	128
Shuffle	True
Rollout prompt truncation	`left`
Batching	
Effective global batch size	32
Rollouts per prompt (n)	1
Micro-batch size per GPU	1
Gradient accumulation steps	4
Max. concurrent sequences per engine	32
Max. batched tokens per engine	16,384
Rollout / Generation	
Inference engine	vLLM
Rollout mode	Async
Temperature	1.0
Top-p	1.0
Top-k	-1
Sampling	True
Max. new tokens	8,192
Max. context length	32,768
Tensor parallel size	1
Rollout replicas	8
Enable chunked prefill	True
GPU memory utilization	0.55
Distillation Objective	
Distillation temperature	1.1
Pointwise KL-entry upper clip	0.05
KL computation chunk size	256
Token sampling ratio	1.0
Reward weighting	Disabled
Gradient clip norm	1.0
Teacher	
Initialization	Same initial checkpoint as student
Weight updates	None; frozen
Scored continuation	Student response prefix, teacher-forced
Strategy	FSDP full-shard
Sequence parallel size	1 (disabled)
Parameter CPU offload	True

Parameter	OPSD and CASD
Parallelism	
Number of GPUs	8
Number of nodes	1
Student strategy	FSDP full-shard
Sequence parallel size	1
Gradient checkpointing	True
Student parameter CPU offload	True
Optimizer CPU offload	True
Training	
Optimizer	AdamW
Learning rate	5×10^{-6}
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	10^{-8}
Scheduler	Linear warmup, then constant
Warmup steps	20
Weight decay	0.01
Precision	BF16 mixed precision
NCCL timeout	600
Total optimizer steps	200
Checkpoint interval	50

B.3 Data Details

As early stopping is applied during data generation, we report the number of examples that terminate at each stage in Table 9. Approximately two-thirds of the examples proceed through all stages, while the remaining examples terminate earlier once the stopping criterion is satisfied.

Table 9: Distribution of termination stages during data generation.

Termination Stage	Qwen3-4B	Qwen3-8B
T0	1,426 (14.26%)	1,457 (14.57%)
T1	1,089 (10.89%)	1,109 (11.09%)
T2	800 (8.00%)	719 (7.19%)
T3	6,685 (66.85%)	6,715 (67.15%)
Total	**10,000**	**10,000**

In addition, CASD uniformly selects a rollout from the initial population T_0 whose final answer matches the terminal pseudo-answer. If no matching rollout exists in T_0, we uniformly select a matching rollout from the terminal population, without searching intermediate stages. Table 10 reports the stage from which the rollout actually used for CASD training is selected. In more than 90% of the examples, a matching T_0 rollout can be recovered; only a small fraction requires falling back to a terminal-stage rollout.

Table 10: Source stage of the rollout ultimately selected for CASD training.

Selected Rollout Source	Qwen3-4B	Qwen3-8B
T0: exact-match selection	**9,371 (93.71%)**	**9,141 (91.41%)**
T1: fallback after no match	1 (0.01%)	7 (0.07%)
T2: fallback after no match	7 (0.07%)	14 (0.14%)
T3: fallback after no match	621 (6.21%)	838 (8.38%)
Total	**10,000**	**10,000**

C PROMPT TEMPLATES

Below are the prompts we used for sampling and on-policy self-distillation. Placeholders such as `{QUESTION}` are replaced with the corresponding problem or generated response. Ellipses indicate additional candidate solutions and are not included in the prompts.

C.1 SAMPLING PROMPTS

Parallel sampling. We independently sample multiple responses to each problem using the following system and user messages.

Parallel sampling: system message

```
You are an expert mathematician. Solve the problem step by step and
    put your final answer within \boxed{}.
```

Parallel sampling: user message

```
{QUESTION}
```

Sequential sampling. We start with the same messages as parallel sampling. If a response is incorrect, we append the following text and ask the model to continue revising its answer.

Sequential sampling: text appended to the response

```
Wait, this response is incorrect. Let me carefully re-evaluate the
    question and identify the errors in the previous response before
    providing a correction, and return in the asked format.
```

We retain all previous responses and correction messages in the same assistant turn. We use correctness verification to decide whether to revise or stop, without including the ground-truth answer in the prompt. Generation also stops when the round or context limit is reached.

C.2 DAG-BASED GENERATION PROMPTS

Recursive Self-Aggregation (RSA). We first generate independent candidate solutions using the following prompt.

RSA initial solutions: user message

```
{QUESTION}

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

We then ask the model to combine four candidate solutions from the preceding round.

RSA aggregation: user message

```
You are given a math problem and several candidate solutions. Some
    candidates may be incorrect or contain errors. Aggregate the
    useful ideas and produce a single, high-quality solution. Reason
    carefully; if candidates disagree, choose the correct path. If
    all are incorrect, then attempt a different strategy. End with
    the final result in \boxed{}.
```

```
Problem:

{QUESTION}

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

Candidate solutions (may contain mistakes):

---- Solution 1 ----
{PARENT_SOLUTION_1}

---- Solution 2 ----
{PARENT_SOLUTION_2}

---- Solution 3 ----
{PARENT_SOLUTION_3}

---- Solution 4 ----
{PARENT_SOLUTION_4}

Now write a single improved solution. Provide clear reasoning and end
    with the final answer in \boxed{}.
```

We use this prompt in each aggregation round, replacing the four candidate solutions with responses from the preceding round.

PDR. Since the original PDR implementation was not open-sourced, we reconstructed the following prompts based on the description in Madaan et al. (2025). We first generate candidate solutions using the following prompt.

PDR initial solutions: user message

```
Solve the following math problem efficiently and clearly. Please
    reason step by step, and put your final answer within
    $\boxed{answer}$.

Where [answer] is just the final number or expression that solves the
    problem.

Problem: {QUESTION}
```

We then ask the model to summarize the candidate solutions.

PDR summarizing candidate solutions: user message

```
You are given a math problem and several candidate solutions.
    Aggregate the useful ideas into a compact workspace for another
    solver. Capture agreements, contradictions, derived facts,
    unresolved subgoals, and next steps.

Problem:
{QUESTION}

Candidate solutions:
---- Solution 1 ----
{CANDIDATE_SOLUTION_1}

...

---- Solution m ----
```

```
{CANDIDATE_SOLUTION_m}

Write the compact workspace.
```

We use this summary to generate a new solution, asking the model to treat it as unverified information.

PDR refinement: user message

```
Solve the following math problem efficiently and clearly. Please
    reason step by step, and put your final answer within
    $\boxed{answer}$.

Where [answer] is just the final number or expression that solves the
    problem.

Problem: {QUESTION}

Here is a summary of the reasoning traces by a few other solvers:

<summary>
{WORKSPACE}
</summary>

Treat the summary as unverified; and use the summary as context to
    come up with an answer.
```

MatryoshkaThinking. Since the original MatryoshkaThinking implementation was not open-sourced, we reconstructed the following prompts based on the description in Chen et al. (2025). We generate initial solutions using the same prompt as for PDR, then ask the model to check each candidate solution.

MatryoshkaThinking verification: user message

```
Check whether the candidate solution correctly solves the math
    problem. Verify the reasoning and final answer.

Problem:
{QUESTION}

Candidate solution:
<model_response>
{CANDIDATE_SOLUTION}
</model_response>

End with one line: VERDICT: CORRECT or VERDICT: INCORRECT.
```

We retain a candidate if the model's last valid verdict is `CORRECT`; candidates without a valid verdict are discarded. We then remove duplicates and combine the retained solutions using the following prompt. These solutions are checked by the model, without ground-truth feedback.

MatryoshkaThinking synthesis: user message

```
You are given a math problem and verified candidate solutions.
    Aggregate the useful ideas and produce a single, high-quality
    solution. Provide clear reasoning and end with the final answer
    in \boxed{}.

Problem:
```

```
{QUESTION}

Verified candidate solutions:
---- Solution 1 ----
{ACCEPTED_SOLUTION_1}

...

---- Solution m ----
{ACCEPTED_SOLUTION_m}

Now write the solution.
```

If no candidate is retained, we replace the candidate list with `(none)`.

C.3 Teacher and Student Prompts for OPSD

We give the student only the problem and the teacher one additional reference solution as privileged context.

Student Prompt. We sample the student's response using the following prompt.

OPSD student: user message

```
{QUESTION}

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

Teacher prompt. We give the teacher the same probelm together with the selected reference solution.

OPSD teacher: user message

```
Problem: {QUESTION}

Here is a reference solution to this problem:
=== Reference Solution Begin ===
{REFERENCE}
=== Reference Solution End ===


After reading the reference solution above, make sure you truly
    understand the reasoning behind each step -- do not copy or
    paraphrase it. Now, using your own words and independent
    reasoning, derive the same final answer to the problem above.
    Think step by step, explore different approaches, and don't be
    afraid to backtrack or reconsider if something doesn't work out:

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

For majority-vote references, we fill `{REFERENCE}` with a complete solution whose final answer matches the vote. We keep this prompt unchanged when comparing reference-selection methods. During distillation, the frozen teacher scores the student's response, as described in Section 3. We apply the loss only to the selected prefix of response tokens, excluding prompt tokens.