Aletheia: What Makes RLVR For Code Verifiers Tick?

Anonymous authors
Paper under double-blind review

Abstract

Multi-domain thinking verifiers trained via Reinforcement Learning with Verifiable Rewards (RLVR) are a cornerstone of modern post-training. However, their adoption in code generation has lagged behind that of execution feedback due to the prohibitive costs of the full RLVR pipeline. In this work, we ablate three primary choices along the performance–cost trade-off in RLVR: intermediate thinking traces, learning from negative samples, and on-policy training. We introduce **Aletheia**, a controlled, execution-grounded testbed to facilitate a contamination-free analysis of code verifier training recipes across disparate model sizes and covariate shifts across two common verifier application scenarios. Our analysis reveals that the optimal training recipe is scale-dependent: on-policy learning is the primary performance driver for small verifiers, whereas the thinking budget becomes the most vital factor at larger scales. While leveraging negative samples has a consistent impact on top-1 selection accuracy across sizes, their contribution to ranking reconstruction increases monotonically with scale and plays a key role in stabilizing training at large sizes. Our Pareto optimality analysis demonstrates that eliminating on-policy training at larger model scales yields a verifier that performs comparably to the full RLVR recipe. Furthermore, we find that eschewing thinking traces serves as a compute-efficient strategy at lower budgets, offering a strong trade-off between training cost and verifier accuracy. Ultimately, our work provides the empirical foundation necessary to efficiently deploy robust code verifiers, thereby enabling their wider adoption in post-training pipelines for large code generation models[1].

1 Introduction

There has been a strong uptick in the adoption of coding assistants that use agentic harnesses such as Claude Code (Anthropic, 2025). Their daily use by software engineers is driven by recent increases in capability via improved post-training (Yang et al., 2025; Research et al., 2026). Most post-training recipes require that LLM-generated code be verified against runtime signals (Le et al., 2022; Shojaee et al., 2023; Gehring et al., 2025; Liu et al., 2023a). However, self-contained executable codes with accompanying test-cases are a scarce resource, even for curated competitive programming datasets, and manual creation does not scale Wang et al. (2025d). While automatic test-case generation is a common solution (Li et al., 2022; Liu et al., 2023b; Li et al., 2023a), it struggles with test coverage and the inherent difficulty of specifying assertions for open-ended tasks. Alternatives like self-contained environment creation (Jain et al., 2025; Xie et al., 2024) and world modelling (FAIR et al., 2025) can be challenging for compiled languages without mature package managers.

In this work, we revisit surrogate code-execution verifiers (Ni et al., 2023; Li et al., 2025b; Zeng et al., 2025; Shi et al., 2022; Zhang et al., 2023b): models trained to score code snippets based on execution outcomes without actually executing them. In addition to removing the dependence on high-quality test cases, these verifiers obviate code execution and environment-setup overheads in downstream applications like Reinforcement Learning with Verifiable Rewards (RLVR; Zhu et al., 2026) and Best-of-N (BoN) inference (Zeng et al., 2025). Such verifiers can additionally leverage the code understanding and generalization capabilities of LLMs to provide granular feedback signals for long-horizon tasks where sparse episodic rewards may fail to adequately guide convergence (Cui et al., 2025).

[1]We open-source our code at https://anonymous.4open.science/r/aletheia-2026/

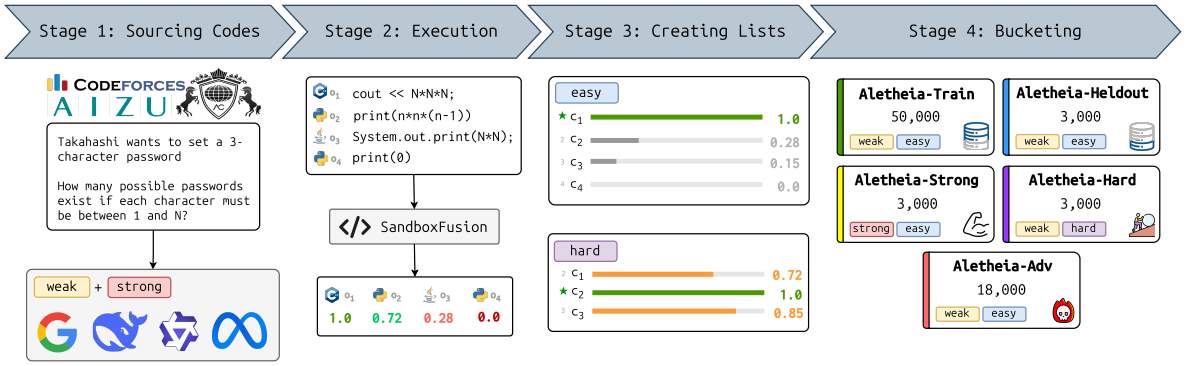

Figure 1: **`Aletheia` testbed curation pipeline.** We follow a four-stage procedure: (1) generating solutions for competition-level programming questions from `CodeContests`$^+$ (Wang et al., 2025d) from a pool of **`Weak`** and **`Strong`** open-source LLMs; (2) obtaining ground-truth pass rates (`PR`s) for the obtained codes through execution using SandboxFusion; (3) constructing lists of `2-5` candidates where exactly one is fully correct. These lists are either **`Easy`** (second best `PR` $<$ `0.5`) or **`Hard`** (second best `PR` $\in$ `[0.7, 0.9]`); and (4) partitioning the resulting data into completely disjoint training and evaluation sets across three covariate shifts: strong generators, hard comparisons, and adversarial prompts. Refer to Section 2.1 for a detailed description.

The lack of RLVR adoption for code verifiers. Recently, generative verifiers have proliferated in post-training pipelines for reasoning-heavy domains such as math and science (Liu et al., 2025b; Ma et al., 2025; Cen et al., 2025). Often, these verifiers are trained to reason before answering using RLVR (Chen et al., 2025d; Huang et al., 2025), which boosts interpretability (Gunjal et al., 2025) and mitigates reward hacking (Chen et al., 2025b). However, these advances have made fewer inroads into code verifiers, which are predominantly encoder-only regression models (Zhu et al., 2026; Zhang et al., 2023b; Shi et al., 2022). Although such models can approximate code execution outcomes (Akhauri et al., 2025), they remain brittle against adversarial and semantic-preserving transformations even at scale (Haroon et al., 2025; Lyu et al., 2025), pointing to the need for richer, generative approaches. Yet the lack of adoption is unsurprising: the full RLVR recipe is expensive and demands intricate orchestration of rollout, behavior, and reference policies. Thus, we study verifier training along three axes where RLVR differs from cheaper post-training recipes, namely: generating long intermediate reasoning traces (**Thinking**), learning from both positive and negative samples (**Negatives**), and learning from data generated by an updated policy (**Online**).

Although these RLVR components have been studied independently for generator training (Tajwar et al., 2024; Lanchantin et al., 2025; Zhu et al., 2025), no equivalent analysis exists for verifiers. This gap is significant due to the well-documented inconsistencies between the generative and verification abilities of LLMs (Li et al., 2023b; Rodriguez et al., 2025; Song et al., 2025). Unlike generators, evaluating verifiers requires going beyond downstream task accuracy. Razin et al. (2026) show that even a perfectly accurate verifier can induce a flat loss landscape and stall learning. Moreover, LLM verifiers are known to break down under distribution shifts (Eisenstein et al., 2024) and adversarial prompting (Moon et al., 2025; LAM et al., 2025), which can corrupt the generator being supervised. The optimal code verifier training recipe also likely varies with scale (Kaplan et al., 2020; Hoffmann et al., 2022) and depends on the application, such as BoN selection or use as an RL reward model (Wen et al., 2025; Kim et al., 2025).

The need for an evaluation testbed for verifiers. Although integrating verifiers into their downstream use cases seems like a natural way to evaluate verifiers, it is noisy and the resulting policy improvement is a poor proxy for reward quality. This is especially true for verifiers used as RL reward models. Such methods are known to succeed only when the base model learns certain task-specific primitives during earlier training phases (Setlur et al., 2026; Zhang et al., 2025a; Wang et al., 2025c), making downstream performance an amalgam of prior capability, reward signal, and the optimization process (Wen et al., 2025; Kim et al., 2025). These confounds are so stark that certain models improve even under random rewards (Shao et al., 2025).

Additionally, the pipeline is expensive and slow to iterate on, which significantly hurts reproducibility and slows progress (Frick et al., 2025). Evidence suggests that the early steps in RLVR merely improve sampling efficiency (Yue et al., 2025; Wu et al., 2025) and true expansion of reasoning capabilities requires prolonged

training (Liu et al., 2025a; Yao et al., 2025b), which puts a principled evaluation out of reach for most reasonable budgets. Even recent RL scaling studies require discarding the first ∼1.5k GPU-hours of every run (Devvrit et al., 2026), much like pre-training scaling studies (Li et al., 2025a; Porian et al., 2024).

To sidestep these limitations, we create **Aletheia**, an execution-grounded testbed for evaluating verifier training recipes. We mirror downstream evaluation scenarios through a contamination-free testbed that enforces a strict training−evaluation partition, isolating algorithmic robustness to out-of-distribution (OOD) scenarios from simple data exposure. We ablate the three RLVR components mentioned earlier across three covariate shifts commonly encountered in downstream evaluations: stronger generators than the ones seen during training (Aletheia-Strong), incorrect solutions that are semantically very close to the correct one (Aletheia-Hard), and adversarially modified code snippets (Aletheia-Adv). We validate our findings across three model sizes (1.5/7/14B), uncovering scale-dependent training dynamics. Following prior work on reward model evaluations (Feng et al., 2025; Wen et al., 2025; Kim et al., 2025), we evaluate BoN performance as the verifier's ability to select the best candidate from a list (ListAcc), and RL performance as its ability to reconstruct the full ordered ranking of a list of candidates ($K\tau$).

Our analysis reveals that while RLVR is the best-performing method to train verifiers in most evaluation settings, the contribution of each ablated component varies with scale. Across both downstream application scenarios, we find that on-policy learning is critical for small verifiers, but its contribution diminishes as model size increases. Conversely, thinking traces offer limited benefits at smaller scales but become essential for 14B models. Meanwhile, negative samples provide a near-consistent boost to all sizes for BoN selection but a monotonic boost for RL, and play a critical role in stabilizing training at larger scales. Utilizing additional compute for verifiers at inference time using self-consistency yields modest gains and cannot compensate for any of the core components we analyze.

Additionally, we study the Pareto optimality of each ablated recipe with respect to cost and performance. DPO-Think-14B is competitive with the full GRPO-Think recipe at 5.2× lower cost, and is the only 14B method on the Pareto frontier across all evaluations simultaneously. The full GRPO recipe is warranted only when peak performance across generator shifts and adversarial perturbations is required. For Aletheia-Hard, practitioners should favor DPO-Think-14B for BoN or RAFT-14B for RL, as on-policy training provides no cost-adjusted benefit. GRPO-Instruct-7B is a good low-budget option on the Pareto frontier for all evaluations, but has a weaker absolute performance than DPO-Think-14B. We summarize our contributions as follows.

- We introduce **Aletheia**, an execution-grounded testbed that enables contamination-free evaluation across three covariate shifts: stronger generators, harder comparisons, and adversarial responses (Section 2).
- We isolate the contributions of three core RLVR components to code verifier performance across different model sizes (1.5/7/14B): thinking traces, on-policy learning, and negative samples (Section 3).
- We conduct a cost-benefit analysis of the RLVR training recipe for robust code verifiers (Section 4).

2 Experimental Setup

2.1 Testbed Creation

A code verifier's value is ultimately determined by its downstream use, either as a re-ranker in BoN inference or as a reward model inside an RLVR pipeline. Evaluating verifiers against either of these uses, however, is far from straightforward. The de-facto offline metric, paired accuracy on RewardBench-style benchmarks (Lambert et al., 2025; Liu et al., 2024b; Tan et al., 2024), is cheap but only weakly predictive of downstream performance: it is insensitive to score separation across the full candidate list (Razin et al., 2026), to discrimination under near-correct distractors (Kim et al., 2025; Feng et al., 2025), and to the distributional coverage required as the policy drifts from the verifier's training distribution (Eisenstein et al., 2024; Wen et al., 2025).

To reliably evaluate verifier training recipes, we construct **Aletheia**: a controlled, contamination-free, execution-grounded testbed. Each prompt is a listwise selection problem, which disincentivizes models from gaming rewards via random guessing and correlates better with downstream performance than paired accuracy (Wen et al., 2025). Following prior work on reliable verifier evaluation (Kim et al., 2025), we source codes from a wide range of model families and sizes (Table 2), but maintain a single generator within a single prompt to mimic downstream evaluation (Feng et al., 2025). Concretely, given a coding problem $\mathbf{x}$ and N distinct candidate solutions $\mathbf{C} = \{\mathbf{c}_n\}_{n=1}^{N}$ with execution-derived pass rates $\mathbf{P} = \{\mathrm{p}_n\}_{n=1}^{N}$, the verifier

Table 1: **Descriptive statistics for our datasets.** Code lengths are measured in tokens and averaged over the lists within a single instance. Average similarities are computed as the cosine similarities between the embeddings of the codes in each list.

Name	#Rows	#Qtns	Code Len.	Code Sim.
Aletheia-Train	50,000	1247	208	0.896
Aletheia-Heldout	3,000	456	236	0.893
Aletheia-Strong	3,000	151	287	0.898
Aletheia-Hard	3,000	137	186	0.931
Aletheia-Adv	18,000	456	246	0.882

Table 2: **Generators used for our datasets.** ΔScore is the difference in BigCodeBench-Instruct scores (Zhuo et al., 2025) between the **Strong** and **Weak** models, representing the gap in their abilities. We use the -Instruct variants for all the generators.

Model Family	Weak	Strong	ΔScore
deepseek-ai/deepseek-coder	6.7B	33B	6.5
google/gemma2	9B	27B	5.1
meta/llama-3.1	8B	70B	13.3
qwen/qwen2.5-coder	7B	32B	8.6

generates $\mathbf{y} = (\mathbf{z}, \mathbf{o})$: a reasoning trace $\mathbf{z}$ followed by a predicted index $\mathbf{o} \in \{1, \ldots, N\}$ identifying the best candidate. By construction, exactly one candidate is fully correct ($\max_n \mathrm{p}_n = 1$), so the ground truth is unambiguous and the reward $\mathbf{1}[\mathbf{o} = \arg\max_n \mathrm{p}_n]$ is trivially computable.

We source competitive programming problems from CodeContests$^+$ (Wang et al., 2025d), each with $\approx$25 synthetic test cases. We only include high-quality problems with true positive rates and true negative rates > 0.9 against a pre-evaluated user-submitted solution pool, and discard samples with < 5 test cases or a time limit > 3 seconds. This gives us 4903 programming questions. We generate 50 completions for each problem in Python, C++, and Java at a sampling temperature of 1.0 using a mix of **Weak** and **Strong** LLMs (Table 2). The resulting code snippets are executed using SandboxFusion[2] to calculate their pass rates (PRs); the percentage of test cases passed for each code. We construct lists of 2 - 5 solution codes with distinct PRs. We divide these lists into **Easy** and **Hard** buckets based on their PRs; **Easy** contains lists where $\mathrm{PR}_{\mathrm{incorrect}} \in$ [0, 0.5] and **Hard** has $\mathrm{PR}_{\mathrm{incorrect}} \in$ [0.7, 0.9], making them harder to distinguish from the correct code.

We subsample the **Weak**-**Easy** bucket to 50,000 instances, dubbed **Aletheia-Train**. Additionally, we create four evaluation datasets, evenly distributed by list length and programming language:

- **Aletheia-Heldout.** An in-distribution evaluation set containing **Easy** comparisons by **Weak** models.
- **Aletheia-Strong.** **Easy** comparisons by **Strong** models, which tests the verifier's robustness to a shift in the generator's capability, without altering the quality of the codes being compared (Zhou et al., 2025).
- **Aletheia-Hard.** **Hard** comparisons generated by **Weak** models. Verifier performance on this dataset indicates their ability to generalize from easy to hard (Hase et al., 2024; Sun et al., 2024).
- **Aletheia-Adv.** It evaluates the adversarial robustness of our verifiers. We apply three positive and negative modifications to the incorrect and correct codes in Aletheia-Heldout respectively (see Section C), based on prior work on biases in LLM judges (LAM et al., 2025; Hwang et al., 2025; Moon et al., 2025).

All OOD evaluations introduce a unidirectional shift from the training data. We further prevent contamination by ensuring no overlap between training and evaluation coding problems. The prompts used in this work are listed in Section F. We summarize dataset statistics in Table 1. We calculate the average code similarities in our datasets by encoding them using Qwen3-Embedding-8B (Zhang et al., 2025b), which achieves stellar performance on text embedding benchmarks like MTEB (Muennighoff et al., 2023; Enevoldsen et al., 2025). The codes within a single prompt are quite similar by virtue of being generated from the same model, with all similarity scores being > 0.88. The codes in Aletheia-Hard particularly stand out, with an average similarity of 0.93 and a smaller average length of 186 tokens. Since all codes in this dataset pass at least 70% of the test cases, they are semantically very close to the correct solution and thus harder to distinguish.

2.2 Evaluation Metrics

The Aletheia testbed is carefully designed to evaluate reward models in their two most common downstream usecases via complementary metrics. Crucially, our task formulation matches BoN inference almost exactly: selecting the PR = 1.0 winner from a generator-matched pool across a variable number of candidates. Moreover, recent work suggests that accuracy evaluated over multiple comparisons and across a range of response quality predicts BoN performance better than paired accuracy (Wen et al., 2025). Thus, we report the average top-1

[2] bytedance/SandboxFusion

selection accuracy (`ListAcc`) across all evaluation datasets as a predictor of downstream BoN performance. During evaluation, we generate responses using a temperature of `0.6` and `top-p = 0.95` nucleus sampling.

Despite having high correlation with BoN, accuracy alone is insufficient to predict the utility of a verifier as an RL reward-model (Kim et al., 2025). Even a perfectly accurate verifier can induce a flat loss landscape if it cannot sufficiently differentiate the relative quality of the incorrect codes (Razin et al., 2026). As mentioned in Section 1, direct RL integration is not indicative of reward quality and is prohibitively expensive. Thus, we evaluate our verifiers' ability to reconstruct the full ranking of N candidates rather than just top-1 selection accuracy, which is proven to better predict downstream RL performance (Kim et al., 2025; Wen et al., 2025; Feng et al., 2025). We adopt Kendall's τ-b ($\mathsf{K}\tau$) as our reranking metric. For each list of N candidates, we issue all $\binom{N}{2}$ pairwise comparisons as independent verifier generations and parse each verdict to identify the winning candidate. We score every candidate $\mathbf{c}_n$ by its number of pairwise wins and compute

$$\mathsf{K}\tau = \frac{n_C - n_D}{\sqrt{(n_C + n_D + T_w)(n_C + n_D + T_p)}},$$

where n_C and n_D are the numbers of candidate pairs ranked concordantly and discordantly by the predicted win counts against the execution-derived pass rates $\mathbf{P}$, and T_w, T_p count pairs tied in the predicted win counts and in $\mathbf{P}$ respectively ($T_p = 0$ in our case). We use the tie-corrected τ-b rather than τ-a because candidates routinely tie in predicted win counts. We report $\mathsf{K}\tau$ averaged over lists in each evaluation set. A $\mathsf{K}\tau$=`+1` implies a perfectly ordered list, `0` an uninformative ordering, and `-1` indicates a fully reversed one. Unlike top-1 selection accuracy, $\mathsf{K}\tau$ uses every pair in the candidate list and remains discriminative even when correct and incorrect codes have near-identical pass rates, which captures the score-separation property required by downstream policy-gradient training (Razin et al., 2026).

2.3 Training Details

We validate our findings across a wide range of model sizes and training parameter counts, including `1.5B`, `7B`, and `14B` for each method. Unless explicitly mentioned, we initialize each method from the `DeepSeek-R1-Distill-Qwen2.5` models (DeepSeek-AI, 2025) because they have been warm-started to generate reasoning traces before answering. To ensure a fair comparison, all methods are trained for an identical number of gradient updates. For on-policy methods, we generate `16` responses at a high sampling temperature of `1.0` and award a `+1` to generations that identify the correct candidate, and `0` otherwise. We also apply a `-1` penalty to generations that violate the format. We provide a detailed description of our training setup in Section A and experiment with alternate reward formulations in Section B.

3 Research Questions and Results

Our analysis employs a series of controlled experiments to isolate the contributions of Thinking, Negatives, and Online components. We use `GRPO` (Shao et al., 2024) as a baseline that includes all three components and compare it with algorithms that lack the single component under study. This design choice eliminates confounding factors, ensuring observed performance reflects each component's individual contribution. We study the effect of removing two components in Section E.1.

3.1 `RQ1`: Do Code Verifiers Need to Generate Long Reasoning Traces?

Summary Of Findings For Best-of-N 1

- The contribution of thinking traces to verifier performance increases monotonically with model scale.
- Training reasoning budget follows a similar pattern: 1.5B saturates beyond 8k, but 7-14B continue improving up to 16k.
- Thinking traces are crucial for **Easy**-to-**Hard** generalization.
- Reasoning-style traces enable models to utilize increased inference compute.

Summary Of Findings For RL 1

- The contribution of thinking traces to RL reward model performance similarly increases with model size.
- Training reasoning budget yields continued gains in $\mathsf{K}\tau$ up to 16k for all verifier sizes.
- $\mathsf{K}\tau$ and ListAcc agree on general trends, but the magnitude of each effect varies.

Background. Thinking traces significantly boost LLM performance (Wei et al., 2022; Kojima et al., 2022), but the source of these gains is ambiguous: several works find no causal relation between the model's CoT

Table 3: **Results for ablating thinking trace generation ( ).** We report list accuracy and Kendall τ scores for BoN and RL, respectively. For both metrics, thinking-style traces offer little benefit to small models but are essential for larger models. Increasing the reasoning budget to 16k is almost always useful, but the style of traces alone is most impactful for larger models. Thinking is especially critical for `Aletheia-Hard`.

Thinking	Negatives	Online	Method	Size	Cost ($)	Aletheia-Heldout		Aletheia-Strong		Aletheia-Hard		Aletheia-Adv		Average	
						BoN	RL	BoN	RL	BoN	RL	BoN	RL	BoN	RL
-			Random		-	32.08	0.00	32.08	0.00	32.08	0.00	32.08	0.00	32.08	0.00
			GRPO-Instruct	1.5B	0.587	38.78	11.90	40.51	13.09	31.22	4.11	30.99	2.16	35.41	7.82
			GRPO-Think-4k		1.208	42.73	18.48	40.70	13.08	36.32	5.36	31.15	2.32	37.76	9.81
			GRPO-Think-8k		2.491	46.82	22.90	43.65	17.56	41.61	10.41	38.09	9.50	42.55	15.09
			GRPO-Think-16k		7.806	49.58	27.49	46.09	23.57	40.74	8.92	40.97	16.45	44.38	19.11
			GRPO-Instruct	7B	2.069	57.74	35.42	51.80	29.08	38.59	9.88	52.20	29.20	50.07	25.90
			GRPO-Think-4k		3.561	59.54	37.40	55.00	32.03	46.73	14.92	44.04	21.84	51.32	26.55
			GRPO-Think-8k		7.179	65.03	35.22	56.96	32.40	53.16	17.81	52.03	28.15	56.76	28.40
			GRPO-Think-16k		15.101	74.81	51.52	67.28	46.30	53.11	20.31	65.04	46.16	65.05	41.07
			GRPO-Instruct	14B	9.463	63.45	41.70	55.11	31.01	44.15	16.97	54.24	27.51	54.26	29.30
			GRPO-Think-4k		8.558	73.23	46.14	64.95	39.74	54.56	21.34	58.09	31.46	62.69	34.67
			GRPO-Think-8k		14.900	78.37	50.23	69.87	41.73	61.74	27.45	65.71	39.39	68.91	39.70
			GRPO-Think-16k		36.992	88.02	60.10	83.65	60.21	66.84	30.95	83.67	56.84	80.54	52.03

and final answer (Turpin et al., 2023; Aljohani et al., 2025), casting doubt on the notion that the generated tokens allow the model to *think* before answering. This behavior is less common, but still prominent in Large Reasoning Models (LRMs) (Chua & Evans, 2025). Thus, long intermediate chains may not directly influence response quality (Stechly et al., 2025; Kambhampati et al., 2025), sparking interest in generating shorter intermediate tokens (Arora & Zanette, 2025; Sui et al., 2025). We quantify the impact of deeper thinking on verifier quality through a controlled ablation study.

Setup. We evaluate the impact of generating thinking traces on verifier quality by comparing short chain-of-thought (CoT) with longer reasoning-style traces. We also study the impact of varying $\texttt{B}_{\texttt{tr}} = \text{len}(\mathbf{z})$: the maximum permitted length of reasoning traces during training. We train four models: `GRPO-Instruct` with $\texttt{B}_{\texttt{tr}} = 4096$, and three `GRPO-Think` variants with $\texttt{B}_{\texttt{tr}} \in \{4096,\ 8192,\ 16384\}$. `GRPO-Instruct` is initialized from the `Qwen2.5-Instruct`, which does not generate thinking traces by default. We report BoN and RL performance trends using the metrics discussed in Section 2.2, along with the average costs per step, assuming `$10.6` per H200-hour[3]. We also study the effects of self-consistency (Wang et al., 2023) on BoN performance.

Findings as a Best-of-N selector. `GRPO-Think-16k` outperforms all other verifiers in Table 3, but the gap to `GRPO-Instruct` varies with scale. `GRPO-Instruct` and `GRPO-Think-4k` differ by $\leq$ `2.4` BoN points on average for `1.5-7B` verifiers, indicating the *style* of intermediate trace makes little difference at smaller scale. At `14B`, the same comparison goes to `8.4` points. Expanding the training reasoning budget follows the same pattern: for `1.5B`, the `4k`$\rightarrow$`8k` doubling adds `4.8` BoN points, but `8k`$\rightarrow$`16k` adds only `1.8`, indicating diminishing returns. In contrast, `7-14B` models keep climbing up to `16k`, with gains of `8.3` and `11.6` BoN points respectively at the `8k`$\rightarrow$`16k` step. These trends are likely driven by larger models being generally better at utilizing thinking primitives (Gandhi et al., 2025) and longer contexts (Hsieh et al., 2024; Liu et al., 2024a).

Across sizes, the verifiers stay robust to shifts in generator capabilities on `Aletheia-Strong`, losing only $\approx$`5.2` BoN points across all models. This reproduces the scalable-supervision pattern of Burns et al. (2024) and contrasts with earlier reports on verification tasks (Zhou et al., 2025). Similar to the average trend, the style of intermediate traces has minimal influence on `1.5-7B` models, and increasing $\texttt{B}_{\texttt{tr}}$ to `8k` and `16k` tracks the same trend for shifts in generator capability.

Overall, the `1.5B` models are sensitive to adversarial perturbations, performing worse than the random baseline for $\texttt{B}_{\texttt{tr}}$=`4k`. This drop in performance is mitigated by scaling the reasoning budget to `16k`, increasing the model size, or both. Crucially, simply switching the *style* of thinking traces has only a small impact on

[3] `jarvislabs.ai/h200-price`

Figure 2: **Inference-time scaling for the thinking ablation.** CoT-trained models don't benefit from additional compute, while thinking models see a small upward trend across all sizes.

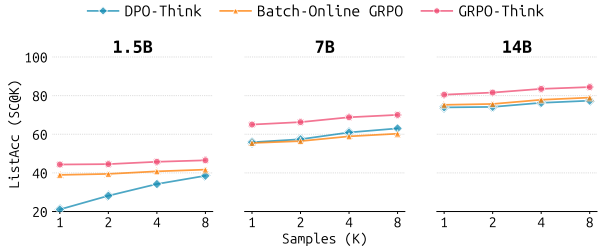

Figure 3: **Inference-time scaling for the online-offline ablation.** The offline – online gap narrows with increasing K, but does not close. DPO-Think can surpass BO-GRPO at larger inference budgets.

performance, and can even hurt the 7B model. We observe robustness to adversarial perturbations only at the larger reasoning budgets and model sizes, contradicting the general notion that reasoning traces make models less vulnerable to judging biases (Wang et al., 2025a). The training – evaluation partition in Aletheia allows us to attribute these trends directly to the training recipe, without contamination confounds.

We observe consistently large performance drops on Aletheia-Hard, suggesting that easy-to-hard generalization performance is challenging for all model sizes. Contrary to the general trend, simply switching from CoT to reasoning traces improves BoN performance across the board, with additional scaling bringing linear gains. Since the codes in Aletheia-Hard are highly similar (c.f. Table 1), verifiers benefit from utilizing reasoning primitives like backtracking and subgoal-setting (Gandhi et al., 2025).

Thinking is also vital for utilizing additional compute at test-time with self-consistency, with CoT traces in GRPO-Instruct yielding a flat SC@K curve (Figure 2). All GRPO-Think variants see a slight upward trend, with the effect being most pronounced at B_{tr}=16k. The curves are strictly ordered across all sizes, suggesting that while self-consistency provides modest gains at inference-time, it cannot replace B_{tr} scaling.

Findings as an RL reward. The comparison under $K\tau$ paints a very similar picture to the one under ListAcc, with GRPO-Think-16k emerging as the best verifier for RL training with $K\tau$=52.03 at the 14B scale. The training budget sweep also tells the same story under $K\tau$ with the algorithm that produces the best selector also generally producing the best ranker, albeit with different magnitudes. Similarly, the trends established for the OOD evaluation datasets on BoN also generally hold for the RL setting. However, there are some notable exceptions to this trend. Although at 1.5B, increasing B_{tr} from 8k→16k yields diminishing returns for BoN as established earlier, it yields continued gains for RL. On Aletheia-Adv, this scaling yields a modest 2.88 BoN points, but a more substantial 6.95 $K\tau$ points. Surprisingly, while increasing B_{tr} from 4k→8k for the 7B model brings a 5.49 BoN point gain on Aletheia-Heldout, it decreases $K\tau$ by 2.18.

3.2 RQ2: Is On-policy Learning Essential for Verifier Training?

Summary Of Findings For Best-of-N 2

- Off-policy training collapses below random at 1.5B, but is competitive with semi-online methods at larger sizes.
- The online-offline gap between DPO-GRPO narrows with scale but never closes: from 23.27 BoN points at 1.5B to 6.65 at 14B.
- BO-GRPO recovers neither GRPO-Think's BoN performance nor offers a meaningful cost saving.
- Inference-time scaling narrows the offline-online BoN gap, most visibly at 1.5B, but cannot close it.

Summary Of Findings For RL 2

- DPO-Think-1.5B's high $K\tau$ scores are an artifact of noisy evaluation due to low response parseability.
- The necessity of on-policy learning for RL training similarly diminishes with scale.
- On-policy training has little effect on easy-to-hard generalization.
- BO-GRPO offers no consistent $K\tau$ gain over offline DPO at 7 – 14B despite higher compute cost.

Background. On-policy learning is perhaps the most widely studied and the most expensive aspect of RLVR training. Despite its effectiveness, on-policy training is very inefficient and often impractical. Thus, practitioners usually resort to introducing some amount of off-policyness to increase training efficiency (Noukhovitch

Table 4: **Results for ablating on-policy learning ( ).** We report list accuracy and Kendall τ scores for BoN and RL, respectively. Online learning is important for small verifiers, but its importance diminishes as scale increases. Batch-online methods are similarly useful at small scales but don't help larger models. On-policy learning is especially irrelevant for **Easy**-to-**Hard** generalization.

Thinking Negatives Online Batch-online

	Method	Size	Cost ($)	Aletheia-Heldout		Aletheia-Strong		Aletheia-Hard		Aletheia-Adv		Average	
				BoN	RL	BoN	RL	BoN	RL	BoN	RL	BoN	RL
-	Random		-	32.08	0.00	32.08	0.00	32.08	0.00	32.08	0.00	32.08	0.00
	DPO-Think	1.5B	5.951*	21.70	9.46	19.90	16.00	23.41	12.08	19.37	16.39	21.11	13.48
	BO-GRPO		4.322	43.13	17.66	39.89	12.53	39.26	7.19	33.83	6.48	38.99	10.97
	GRPO-Think		7.806	49.58	27.49	46.09	23.57	40.74	8.92	40.97	16.45	44.38	19.11
	DPO-Think	7B	6.403*	63.75	37.58	55.54	31.89	51.20	16.56	52.94	28.81	55.88	28.71
	BO-GRPO		9.588	64.71	41.27	56.18	30.72	52.08	15.68	49.35	26.05	55.46	28.43
	GRPO-Think		15.101	74.81	51.52	67.28	46.30	53.11	20.31	65.04	46.16	65.05	41.07
	DPO-Think	14B	7.087*	82.56	51.12	74.39	43.46	67.58	29.90	71.06	44.26	73.89	42.19
	BO-GRPO		31.144	83.82	49.42	76.33	41.31	67.34	32.21	73.45	42.01	75.29	41.24
	GRPO-Think		36.992	88.02	60.10	83.65	60.21	66.84	30.95	83.67	56.84	80.54	52.03

et al., 2025; Piché et al., 2025). There is no consensus on its necessity: some works find it vital to success in RL algorithms (Noukhovitch et al., 2025; Tang et al., 2024; Yu et al., 2025a), while others claim that introducing a certain amount of off-policyness can match or even outperform fully on-policy methods on mathematical reasoning tasks (Lanchantin et al., 2025; Chen et al., 2025a; Song et al., 2024).

Setup. We study the impact of this decision through three representative algorithms. `DPO-Think` serves as our purely offline algorithm, and `Batch-online (BO-) GRPO` represents the middle ground between online and offline methods, sampling a batch of responses and performing multiple gradient updates on mini-batches of generated data (Zheng et al., 2025). We present the results in Table 4. For `DPO-Think`, we include the costs of creating an offline preference dataset as detailed in Section A.

Findings as a Best-of-N selector. `GRPO-Think` continues to dominate the ablated variants at all model sizes. The offline-online gap narrows with scale, with the `DPO-GRPO` gap shrinking from `23.27` BoN points at `1.5B` to just `6.65` at `14B`. `DPO-Think` collapses at smaller sizes, performing worse than the random baseline due to degeneration and unparseable verdicts. Specifically, `DPO-Think-1.5B` produces a parseable final answer in only `43%` of cases on average (see Table 12 for more details). `BO-GRPO` is a good tradeoff at smaller scales, with lower costs than a fully offline approach, and performance within `6` points of the full `GRPO-Think` recipe.

However, the story flips at `7-14B` sizes. `DPO-Think` closes the gap with `BO-GRPO` on BoN while also decreasing costs, and is `4.4`× cheaper at `14B`. `BO-GRPO` fails to recover `GRPO-Think`'s BoN, and its lower cost is not worth the performance drop, since cheaper alternatives exist. Our observation contradicts that of Lanchantin et al. (2025), who report batch-online methods that match or outperform fully online training on math tasks. We attribute the discrepancy to their use of `Llama-3.1-8B-Instruct` (no thinking traces) and the original GRPO recipe (Shao et al., 2024), while we incorporate improvements detailed in Section A.

The `Aletheia-Strong` and `Aletheia-Adv` evaluations track the general trend almost exactly, except for `DPO-Think-7B` being slightly more robust to adversarial prompts than `BO-GRPO`. On-policy training has no perceptible impact on easy-to-hard generalization at `7-14B`, suggesting that a well-curated offline preference dataset may be sufficient for such generalization. Although `DPO-Think-1.5B` gains `1.71` BoN points on `Aletheia-Hard`, it is still worse than random chance and we thus do not consider it a meaningful gain.

Scaling inference-time compute benefits all methods and model sizes (Figure 3). Such scaling particularly benefits `DPO-Think-1.5B`, which almost matches `BO-GRPO`'s BoN at `K=8`. This behavior is largely driven by a higher parse rate due to increased sampling at inference time. At `7B`, additional inference-time compute even allows `DPO-Think` to surpass `BO-GRPO`'s BoN scores at `K=8`, but still trails `GRPO-Think` even at `K=1`. We thus conclude that, similar to Thinking, inference-time compute scaling cannot fully compensate for the lack of on-policy training, but can significantly narrow the gap, especially for semi-online methods.

*DPO includes the cost for creating the offline dataset.

Table 5: **Results for ablating learning from negatives ( ).** We report list accuracy and Kendall τ scores for BoN and RL respectively. Negatives are consistently beneficial across all model sizes for BoN, but more important for large models during RL. **Easy**-to-**Hard** generalization is largely unaffected by negatives.

Thinking	Negatives	Online	Method	Size	Cost ($)	Aletheia-Heldout		Aletheia-Strong		Aletheia-Hard		Aletheia-Adv		Average	
						BoN	RL	BoN	RL	BoN	RL	BoN	RL	BoN	RL
-			Random		-	32.08	0.00	32.08	0.00	32.08	0.00	32.08	0.00	32.08	0.00
		✓	RAFT	1.5B	4.167	34.76	14.20	31.88	12.31	33.67	9.06	29.12	12.25	32.30	11.96
✓	✓	✓	GRPO-Think		7.806	49.58	27.49	46.09	23.57	40.74	8.92	40.97	16.45	44.38	19.11
		✓	RAFT	7B	6.948	60.86	38.82	52.00	31.89	48.84	17.33	49.24	29.36	52.72	29.35
✓	✓	✓	GRPO-Think		15.101	74.81	51.52	67.28	46.30	53.11	20.31	65.04	46.16	65.05	41.07
		✓	RAFT	14B	12.906	75.55	49.42	66.02	41.31	65.23	32.21	62.03	42.01	67.20	41.24
✓	✓	✓	GRPO-Think		36.992	88.02	60.10	83.65	60.21	66.84	30.95	83.67	56.84	80.54	52.03

Findings as an RL reward model. Similar to BoN, on-policy training makes `GRPO-Think` the best reranker for all model sizes. Despite sub-random BoN performance, `DPO-Think-1.5B` has competitive Kτ values at its scale, even scoring highest on `Aletheia-Hard` with Kτ=`12.08`. However, this number does not indicate that `DPO-Think-1.5B` is a capable reranker. Rather, it is a result of the two metrics handling unparseable verdicts differently. Under list accuracy for BoN, unparseable instances are considered incorrect, causing the method to perform worse than random chance. However, Kτ discards such instances, effectively treating them as ties and penalizing both candidates equally in a pairwise tournament. Consequently, Kτ is calculated over a smaller set of comparisons, and the resulting value is noisy. We report the parse rates for all algorithms in Table 12, which clearly illustrates this behavior. This degeneration is specific to the `1.5B` verifier, possibly due to the model's limited capacity. The parse rates for `DPO-Think` recover at larger scales, suggesting that their competitive Kτ relative to `BO-GRPO` indicates genuine reranking competence.

`BO-GRPO` continues to be a poor substitute for `GRPO-Think` even as an RL reward function. Although `DPO-Think` slightly underperforms `BO-GRPO`'s BoN on average at `14B`, this trend flips on Kτ, further underscoring the shortcomings of semi-online training as a substitute for on-policy training. Unlike BoN, where on-policy learning has little impact on easy-to-hard generalization, the same cannot be said for the RL setting where fully offline methods lag behind the online and semi-online methods by up to `3.75%`. The behaviors of the evaluated recipes on `Aletheia-Strong` and `Aletheia-Adv` largely track the same trends as BoN on Kτ.

3.3 RQ3: Do Negatives Benefit Code Verifiers?

Summary Of Findings For Best-of-N 3

- GRPO-Think outperforms RAFT at every scale with a near-constant gap of ≈12.6 BoN points.
- RAFT-1.5B barely exceeds the random baseline (32.30 vs. 32.08) and falls below it on Aletheia-Strong and Aletheia-Adv.
- Inference-time scaling cannot substitute for negatives: RAFT at K=8 fails to match GRPO-Think even at K=1.

Summary Of Findings For RL 3

- Unlike BoN, the Kτ gap grows from 7.2 at 1.5B to 10.8 at 14B, indicating a growing importance of negatives for RL performance.
- RAFT becomes increasingly unstable at larger sizes, with performance even degrading over training.
- Negatives provide no discriminative benefit on Aletheia-Hard, with RAFT performing comparably to GRPO-Think.

Background. Learning from negative samples is a characteristic of RL algorithms, as well as of contrastive methods like DPO (Rafailov et al., 2023), which optimize the RL objective directly. DPO suffers from reward over-optimization (Gao et al., 2023), and Xu et al. (2024) find that even iterative DPO fails to beat the SFT baseline. The literature on negatives is mixed: Arnal et al. (2025) find that successes are more important than failures in an offline setup, whereas Zhu et al. (2025) find negative reinforcement much more important, even outperforming full training in some scenarios. Xiong et al. (2025) find that learning only from positives comes with a minor performance drop, and certain negative signals can be detrimental.

Setup. We compare `GRPO` to a variant of `RAFT` (Dong et al., 2023), modified to use verifiable rewards. `RAFT` samples and scores N generations online, training on only the correct responses via next-token prediction.

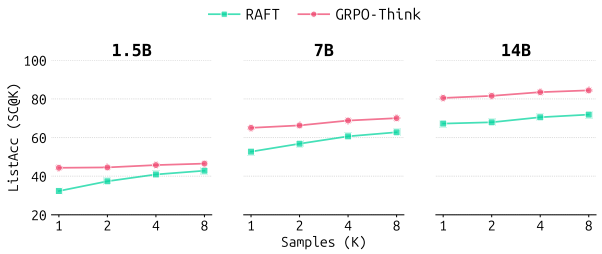

Figure 4: **Inference-time scaling plots for ablating negative samples.** The RAFT-GRPO gap narrows with compute, but is persistent across scales.

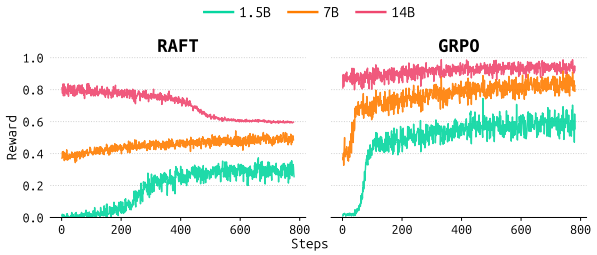

Figure 5: **Reward curves over training for RAFT and GRPO, respectively.** Training without negative samples is increasingly unstable for larger models.

Findings as a Best-of-N selector. GRPO-Think outperforms RAFT across all model sizes on BoN with a near constant gap of $\approx$12.6 points (Table 5). This contrasts with the Online and Thinking components, whose importance diminishes and increases with scale, respectively. However, the gap to GRPO-Think is more consequential at smaller scales, with RAFT-1.5B scoring just above the random baseline of 32.08 points on average and dipping below it on Aletheia-Strong and Aletheia-Adv.

The trends on individual evaluations track the average trend, with GRPO-Think consistently outperforming RAFT on BoN. Also, increasing inference compute cannot replace negative samples during training (Figure 4). While it narrows the gap to GRPO-Think, especially at 1.5B, RAFT at K=8 is worse than GRPO-Think at K=1.

Findings as an RL reward model. While GRPO-Think remains the best reranker across model sizes, the Kτ evaluation tells a more nuanced story than BoN. The RAFT-GRPO gap grows from 7.2 Kτ points at 1.5B to 10.8 at 14B, suggesting a monotonic growth in the role of negative samples. The reason for this behavior is revealed by the training reward curves (Figure 5). Despite a steady growth at 1.5B, the reward curves of RAFT flatline and even degrade at larger sizes, while GRPO-Think continues to improve. We further validate this behavior across other variants of RAFT in Section D.

GRPO-Think is more robust to shifts in generator capability and adversarial perturbations across all sizes. However, similar to our on-policy ablation, both RAFT and GRPO-Think rank near-correct distractors poorly and have largely comparable Kτ on Aletheia-Hard.

4 Optimality Analysis

In the previous section, we studied the individual roles of three components of RLVR training: Thinking, Negatives, and Online. Through our ablations, we established that all three components make nontrivial contributions to the overall success of GRPO-Think, which is the best-performing verifier on average in both BoN and RL applications. However, performance is not the only factor to consider while training verifiers, as each component has a disproportionate cost as seen in Tables 3 to 5.

In this section, we study the trade-off between performance and cost for each of these axes. Specifically, we plot the full GRPO-Think recipe alongside the three ablated variants: GRPO-Instruct (No Thinking), RAFT (No Negatives), and DPO-Think (No Online) against training cost per step for the BoN and RL usecases on each Aletheia evaluation. The dashed line in Figure 6 marks the empirical Pareto frontier per panel.

Offline training achieves near-peak performance at one-fifth the cost of fully online training. DPO-Think-14B occupies a unique position across all eight panels: at $7.09 per step, it achieves 14B-scale performance at a 5.2$\times$ lower cost than GRPO-Think-14B, placing it on the Pareto frontier across all evaluations. DPO-Think-1.5B is dominated by other models due to the fixed cost of creating the offline preference dataset and poor performance stemming from a low parse rate (Table 12).

The GRPO-Instruct verifiers anchor the low-budget end, while RAFT is dominated on all BoN settings. The 1.5B verifier is the cheapest method we study, but it is dominated by a random baseline in some BoN settings. GRPO-Instruct-7B has a good cost – performance tradeoff, and is on the Pareto frontier

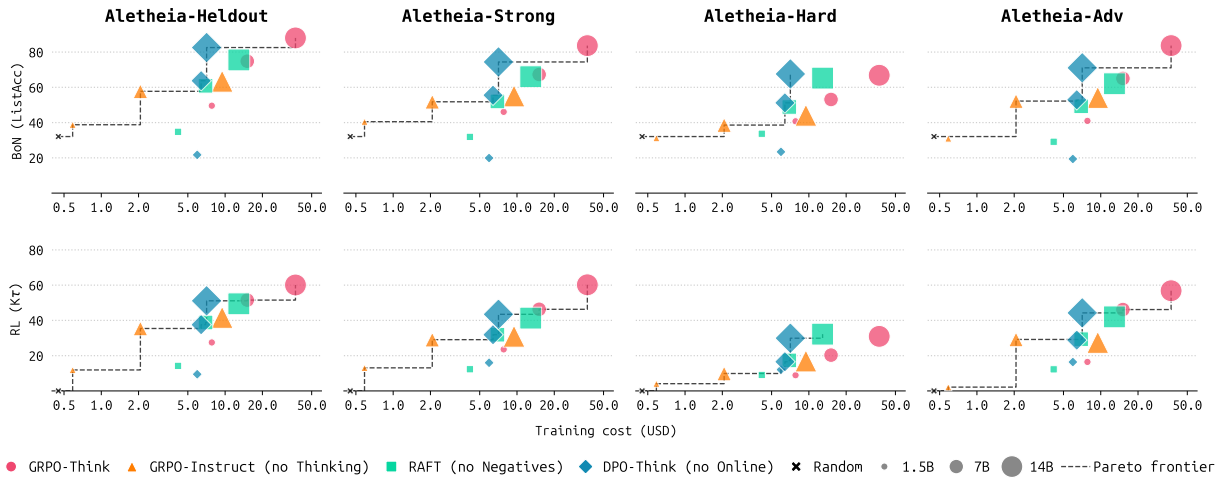

Figure 6: **Cost-performance Pareto curves for `BoN` and `RL` applications.** Top row: BoN (`ListAcc`); bottom row: RL (Kτ). Each column represents a different evaluation dataset. `DPO-Think-14B` is on the frontier in every panel, while `GRPO-Think-16k` is dominated on `Aletheia-Hard` for all model sizes.

in all panels. In contrast, removing negatives yields neither the cost savings of `GRPO-Instruct` nor the performance of `DPO-Think`, making it fall below the frontier.

`Easy`-to-`Hard` generalization favors cheaper alternatives over full GRPO-Think. `GRPO-Think` earns its high cost by extending the BoN and RL frontiers in-distribution and in evaluations with stronger generators and adversarial perturbations. However, `Aletheia-Hard` is a notable exception, where `GRPO-Think` is dominated in both metrics. We conclude that thinking traces are the most vital for `Easy`-to-`Hard` generalization, as evidenced by `DPO-Think` and `RAFT` achieving (near-)Pareto-optimal performance on both metrics.

BoN and RL frontiers largely agree on structure but have some important differences. Although the two metrics agree on the coarse frontier topology, they differ in three key ways. First, `GRPO-Think` enters the Pareto frontier on `Aletheia-Heldout` and `Aletheia-Adv` for RL by narrowly outperforming `DPO-Think-14B`, but is dominated on BoN. This suggests that on-policy training provides a larger benefit for pairwise *ranking* than for argmax *selection*. Second, the magnitude of the `GRPO-Think-14B` premium differs by metric: on `Aletheia-Heldout`, it extends the frontier by `5.5` `ListAcc` points but by `9.0` Kτ points beyond `DPO-Think-14B`, suggesting pairwise ranking quality benefits disproportionately from on-policy training compared to argmax selection. Lastly, `RAFT-14B` is clearly dominated in all BoN evaluations but is near-optimal on RL, even entering the Pareto frontier on `Aletheia-Hard`, which suggests that negative samples are disproportionately effective for improving ranking performance over argmax selection.

5 Related Work

We briefly elaborate on the three most relevant lines of existing work: (1) RLVR for verifier models, (2) surrogate code execution verifiers, and (3) prior analyses of RL components in LLMs.

RLVR for LLM verifiers. Recent literature has substantially expanded verifier training by framing reward modeling as a verifiable re-ranking reasoning task (Whitehouse et al., 2025; Chen et al., 2025c; Huang et al., 2025). Such models have demonstrated state-of-the-art performance on popular reward model benchmarks and have been integrated into the production post-training pipelines of several modern LLMs (Chen et al., 2025b; Du et al., 2025; NVIDIA et al., 2025). Despite empirical gains, the optimal configuration for training such models remains under-explored. In this work, we uncover compute-optimal strategies for verifier training across three disparate model sizes by ablating three core components of the RLVR recipe.

Surrogate code execution. LLMs as surrogate code executors have taken several forms, including regression-based scoring models (Inala et al., 2022; Zhang et al., 2023b; Shi et al., 2022), natural language

self-critique (Zhang et al., 2023a), and reasoning about compiler feedback (Chen et al., 2024). Alternatively, prior work has sought to train LLMs with execution semantics to directly (Zhu et al., 2026; Ni et al., 2024) or indirectly (FAIR et al., 2025; Ruan et al., 2025) improve their ability to abstractly reason about code execution. Beyond the file level, prior work has sought to reason about repository-level test-suite execution outcomes for software engineering tasks (Shum et al., 2025; Pan et al., 2025a). In this work, we show that RLVR enables training robust code verifiers that can scalably supervise much larger policy models, and that using a subset of its core components can even be optimal, yielding competitive results.

Prior analyses of RLVR in LLM training. Modern RL training is notoriously compute-intensive and inefficient (Noukhovitch et al., 2025; Piché et al., 2025), prompting a surge of work simplifying the RL recipe by omitting core components like long thinking traces (Arora & Zanette, 2025; Sui et al., 2025), negative samples (Dong et al., 2023; Gulcehre et al., 2023; Tan et al., 2025), and on-policy learning (Rafailov et al., 2023; Wang et al., 2025b). However, the contribution of these components to RLVR's success is unclear, as described in Sections 3.1 to 3.3. Moreover, they are often treated as isolated rather than synergistic contributors to RLVR's success. A notable exception is Tajwar et al. (2024), who found that on-policy learning and negative samples are complementary and especially important when high-reward responses are less likely under the policy distribution. Additionally, most RLVR studies focus on mathematical reasoning. It is unclear whether these findings transfer to more brittle domains like code verification, where even frontier models frequently fail (Haroon et al., 2025; Lyu et al., 2025). In this work, we deconstruct the training dynamics of code verifiers to reveal that the importance of specific components varies with scale. We provide a roadmap for efficiently training optimal code verifiers to supervise future generations of code models.

6 Conclusion

In this work, we presented a systematic analysis of the three primary drivers of performance and cost in the RLVR pipeline for code verifiers: generating intermediate thinking traces, learning from positive and negative samples, and on-policy training. To facilitate this study, we introduced **`Aletheia`**, a controlled, execution-grounded testbed designed to draw contamination-free conclusions about the training dynamics of code verifiers across different model sizes and covariate shifts for two common verifier application scenarios: Best-of-N inference and RL reward modeling. Our analysis reveals that these components are synergistic, but the degree of their impact is scale-dependent: for both BoN and RL, on-policy learning is the primary performance driver for small verifiers, while thinking becomes the most vital factor as model size increases. Negatives consistently boost performance on top-1 selection, are monotonic contributors to RL performance, and prevent reward curves from degrading at larger sizes. We find that scaling inference-time compute with self-consistency yields only a minor performance boost in most cases and cannot compensate for the absence of any component. Finally, we conduct a Pareto optimality analysis of our verifiers and find that `DPO-Think-14B` is an optimal choice for training verifiers for all scenarios. Although the full RLVR recipe is more performant, its cost is justified only when shifts in generator capability and adversarial perturbations are expected. At low budgets, `GRPO-Instruct-7B` is a strong baseline and is optimal across all evaluations, similar to `DPO-Think-14B`. Therefore, our work establishes a compute-optimal roadmap for practitioners by providing strategies to simplify verifier training across several scales and analyzing the consequences of these simplifications across multiple covariate shifts. More broadly, our work lays the foundation for generative code verifiers to become a more prominent fixture in the post-training pipelines of code generator LLMs by providing recipes for scalable and efficient verifier training.

Ethical Considerations

Our work focuses on training recipes for code verifiers that are used to select the best code from a list of LLM-generated snippets. These verifiers can potentially be exploited by an adversary to generate incorrect or unsafe code. We take steps to mitigate these risks by analyzing the OOD robustness of our verifiers under various shifts in generator capability and in adversarial settings, and report our findings on these evaluations separately. To further encourage research in this area, we thoroughly document our workflow and open-source our datasets, code, and models under the `CC BY-NC-SA 4.0` License.

References

Yash Akhauri, Xingyou Song, Arissa Wongpanich, Bryan Lewandowski, and Mohamed S. Abdelfattah. Regression language models for code. *CoRR*, abs/2509.26476, 2025. doi: 10.48550/ARXIV.2509.26476. URL https://doi.org/10.48550/arXiv.2509.26476.

Manar Aljohani, Jun Hou, Sindhura Kommu, and Xuan Wang. A comprehensive survey on the trustworthiness of large language models in healthcare. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 6720–6748, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.356. URL https://aclanthology.org/2025.findings-emnlp.356/.

Anthropic. Claude code. Blog, 2025. URL https://www.anthropic.com/product/claude-code.

Charles Arnal, Gaëtan Narozniak, Vivien Cabannes, Yunhao Tang, Julia Kempe, and Remi Munos. Asymmetric REINFORCE for off-policy reinforcement learning: Balancing positive and negative rewards. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Ql3sENn0mi.

Daman Arora and Andrea Zanette. Training language models to reason efficiently. *CoRR*, abs/2502.04463, 2025. doi: 10.48550/ARXIV.2502.04463. URL https://doi.org/10.48550/arXiv.2502.04463.

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=ghNRg2mEgN.

Zhepeng Cen, Haolin Chen, Shiyu Wang, Zuxin Liu, Zhiwei Liu, Ding Zhao, Silvio Savarese, Caiming Xiong, Huan Wang, and Weiran Yao. Webscale-rl: Automated data pipeline for scaling RL data to pretraining levels. *CoRR*, abs/2510.06499, 2025. doi: 10.48550/ARXIV.2510.06499. URL https://doi.org/10.48550/arXiv.2510.06499.

Howard Chen, Noam Razin, Karthik Narasimhan, and Danqi Chen. Retaining by doing: The role of on-policy data in mitigating forgetting. *CoRR*, abs/2510.18874, 2025a. doi: 10.48550/ARXIV.2510.18874. URL https://doi.org/10.48550/arXiv.2510.18874.

Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, Yufeng Yuan, et al. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning. *CoRR*, abs/2504.13914, 2025b. doi: 10.48550/ARXIV.2504.13914. URL https://doi.org/10.48550/arXiv.2504.13914.

Nuo Chen, Zhiyuan Hu, Qingyun Zou, Jiaying Wu, Qian Wang, Bryan Hooi, and Bingsheng He. Judgelrm: Large reasoning models as a judge. *CoRR*, abs/2504.00050, 2025c. doi: 10.48550/ARXIV.2504.00050. URL https://doi.org/10.48550/arXiv.2504.00050.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=KuPixIqPiq.

Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang, Denghui Zhang, Tong Zhang, et al. RM-R1: reward modeling as reasoning. *CoRR*, abs/2505.02387, 2025d. doi: 10.48550/ARXIV.2505.02387. URL https://doi.org/10.48550/arXiv.2505.02387.

James Chua and Owain Evans. Are deepseek r1 and other reasoning models more faithful? *CoRR*, abs/2501.08156, 2025. URL https://arxiv.org/abs/2501.08156.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. Process reinforcement through implicit rewards. *CoRR*, abs/2502.01456, 2025. URL https://arxiv.org/abs/2502.01456.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948, 2025. doi: 10.48550/ARXIV.2501.12948. URL https://doi.org/10.48550/arXiv.2501.12948.

Fnu Devvrit, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Sai Surya Duvvuri, Manzil Zaheer, Inderjit S Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FMjeC9Msws.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, KaShun SHUM, and Tong Zhang. RAFT: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=m7p5O7zblY.

Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *CoRR*, abs/2501.12599, 2025. URL https://arxiv.org/abs/2501.12599.

Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alexander Nicholas D'Amour, Krishnamurthy Dj Dvijotham, Adam Fisch, Katherine A Heller, Stephen Robert Pfohl, Deepak Ramachandran, et al. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=5u1GpUkKtG.

Kenneth C. Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzeminski, Genta Indra Winata, et al. MMTEB: massive multilingual text embedding benchmark. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=zl3pfz4VCV.

FAIR, Jade Copet, Quentin Carbonneaux, Gal Cohen, Jonas Gehring, Jacob Kahn, Jannik Kossen, Felix Kreuk, Emily McMilin, Michel Meyer, et al. CWM: an open-weights LLM for research on code generation with world models. *CoRR*, abs/2510.02387, 2025. doi: 10.48550/ARXIV.2510.02387. URL https://doi.org/10.48550/arXiv.2510.02387.

Yuanning Feng, Sinan Wang, Zhengxiang Cheng, Yao Wan, and Dongping Chen. Are we on the right way to assessing LLM-as-a-Judge? *CoRR*, abs/2512.16041, 2025. URL https://arxiv.org/abs/2512.16041.

Evan Frick, Tianle Li, Connor Chen, Wei-Lin Chiang, Anastasios Nikolas Angelopoulos, Jiantao Jiao, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. How to evaluate reward models for RLHF. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=cbttLtO94Q.

Kanishk Gandhi, Ayush K Chakravarthy, Anikait Singh, Nathan Lile, and Noah Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STaRs. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=QGJ9ttXLTy.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10835–10866. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23h.html.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: grounding code llms in execution feedback with reinforcement learning. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=PzSG5nKe1q.

Alexey Gorbatovski, Boris Shaposhnikov, Alexey Malakhov, Nikita Surnachev, Yaroslav Aksenov, Ian Maksimov, Nikita Balagansky, and Daniil Gavrilov. Learn your reference model for real good alignment. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=H0qIWXXLUR.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced Self-Training (ReST) for Language Modeling. *CoRR*, abs/2308.08998, 2023. URL https://arxiv.org/abs/2308.08998.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *CoRR*, abs/2507.17746, 2025. URL https://arxiv.org/abs/2507.17746.

Sabaat Haroon, Ahmad Faraz Khan, Ahmad Humayun, Waris Gill, Abdul Haddi Amjad, Ali R. Butt, Mohammad Taha Khan, and Muhammad Ali Gulzar. How accurately do large language models understand code? *CoRR*, abs/2504.04372, 2025. URL https://arxiv.org/abs/2504.04372.

Peter Hase, Mohit Bansal, Peter Clark, and Sarah Wiegreffe. The unreasonable effectiveness of easy training data for hard tasks. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 7002–7024. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.378. URL https://doi.org/10.18653/v1/2024.acl-long.378.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Hui Huang, Yancheng He, Hongli Zhou, Rui Zhang, Wei Liu, Weixun Wang, Wenbo Su, Bo Zheng, and Jiaheng Liu. Think-j: Learning to think for generative llm-as-a-judge. *CoRR*, abs/2505.14268, 2025. URL https://arxiv.org/abs/2505.14268.

Yerin Hwang, Dongryeol Lee, Taegwan Kang, Yongil Kim, and Kyomin Jung. Can you trick the grader? adversarial persuasion of llm judges. *CoRR*, abs/2508.07805, 2025. URL https://arxiv.org/abs/2508.07805.

Jeevana Priya Inala, Chenglong Wang, Mei Yang, Andrés Codas, Mark Encarnación, Shuvendu K. Lahiri, Madanlal Musuvathi, and Jianfeng Gao. Fault-aware neural code rankers. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/5762c579d09811b7639be2389b3d07be-Abstract-Conference.html.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=7evvwwdo3z.

Subbarao Kambhampati, Kaya Stechly, Karthik Valmeekam, Lucas Paul Saldyt, Siddhant Bhambri, Vardhan Palod, Atharva Gundawar, Soumya Rani Samineni, Durgesh Kalwar, and Upasana Biswas. Stop anthropomorphizing intermediate tokens as reasoning/thinking traces! In *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning*, 2025. URL https://openreview.net/forum?id=g9TJqYCcHA.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Sunghwan Kim, Dongjin Kang, Taeyoon Kwon, Hyungjoo Chae, Dongha Lee, and Jinyoung Yeo. Rethinking reward model evaluation through the lens of reward overoptimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025. URL https://aclanthology.org/2025.acl-long.649/.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html.

Man Ho LAM, Chaozheng Wang, Jen tse Huang, and Michael Lyu. Codecrash: Exposing LLM fragility to misleading natural language in code reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=CAB0EjD9EK.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing Frontiers in Open Language Model Post-Training. *CoRR*, abs/2411.15124, 2024. URL https://arxiv.org/abs/2411.15124.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. RewardBench: Evaluating reward models for language modeling. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1755–1797, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.96. URL https://aclanthology.org/2025.findings-naacl.96/.

Jack Lanchantin, Angelica Chen, Janice Lan, Xian Li, Swarnadeep Saha, Tianlu Wang, Jing Xu, Ping Yu, Weizhe Yuan, Jason E. Weston, et al. Bridging Offline and Online Reinforcement Learning for LLMs. *CoRR*, abs/2506.21495, 2025. URL https://arxiv.org/abs/2506.21495.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu-Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/8636419dea1aa9fbd25fc4248e702da4-Abstract-Conference.html.

Margaret Li, Sneha Kudugunta, and Luke Zettlemoyer. (mis)fitting scaling laws: A survey of scaling law fitting techniques in deep learning. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=xI71dsS3o4.

Qingyao Li, Xinyi Dai, Xiangyang Li, Weinan Zhang, Yasheng Wang, Ruiming Tang, and Yong Yu. Codeprm: Execution feedback-enhanced process reward model for code generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 8169–8182. Association for Computational Linguistics, 2025b. URL https://aclanthology.org/2025.findings-acl.428/.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *CoRR*, abs/2312.14852, 2023a. URL https://arxiv.org/abs/2312.14852.

Xiang Lisa Li, Vaishnavi Shrivastava, Siyan Li, Tatsunori Hashimoto, and Percy Liang. Benchmarking and improving generator-validator consistency of language models. *CoRR*, abs/2310.01846, 2023b. doi: 10.48550/ARXIV.2310.01846. URL https://doi.org/10.48550/arXiv.2310.01846.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. Rltf: Reinforcement learning from unit test feedback. *Transactions on Machine Learning Research*, 2023, 2023a. URL https://openreview.net/forum?id=hjYmsV6nXZ.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023b. URL http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. ProRL: Prolonged reinforcement learning expands reasoning boundaries in large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=YPsJha5HXQ.

Shudong Liu, Hongwei Liu, Junnan Liu, Linchen Xiao, Songyang Gao, Chengqi Lyu, Yuzhe Gu, Wenwei Zhang, Derek F. Wong, Songyang Zhang, et al. CompassVerifier: A unified and robust verifier for LLMs evaluation and outcome reward. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33466–33494, Suzhou, China, 2025b. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1698. URL https://aclanthology.org/2025.emnlp-main.1698/.

Xiang Liu, Peijie Dong, Xuming Hu, and Xiaowen Chu. Longgenbench: Long-context generation benchmark. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 865–883, Miami, Florida, USA, November 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.48. URL https://aclanthology.org/2024.findings-emnlp.48/.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. RM-Bench: Benchmarking Reward Models of Language Models with Subtlety and Style. *CoRR*, abs/2410.16184, 2024b. URL https://arxiv.org/abs/2410.16184.

Zihe Liu, Jiashun Liu, Yancheng He, Weixun Wang, Jiaheng Liu, Ling Pan, Xinyu Hu, Shaopan Xiong, Ju Huang, Jian Hu, et al. Part I: Tricks or Traps? A Deep Dive into RL for LLM Reasoning. *CoRR*, abs/2508.08221, 2025c. URL https://arxiv.org/abs/2508.08221.

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling. *CoRR*, abs/2504.02495, 2025d. URL https://arxiv.org/abs/2504.02495.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Bohan Lyu, Siqiao Huang, Zichen Liang, Qian Sun, and Jiaming Zhang. Surge: On the potential of large language models as general-purpose surrogate code executors. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3268–3308, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.162. URL https://aclanthology.org/2025.emnlp-main.162/.

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun MA, and Wenhu Chen. General-reasoner: Advancing LLM reasoning across all domains. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=pBFVoll8Xa.

Jiwon Moon, Yerin Hwang, Dongryeol Lee, Taegwan Kang, Yongil Kim, and Kyomin Jung. Don't Judge Code by Its Cover: Exploring Biases in LLM Judges for Code Evaluation. *CoRR*, abs/2505.16222, 2025. URL https://arxiv.org/abs/2505.16222.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. MTEB: Massive text embedding benchmark. In Andreas Vlachos and Isabelle Augenstein (eds.), *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 2014–2037, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.eacl-main.148. URL https://aclanthology.org/2023.eacl-main.148.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida I. Wang, and Xi Victoria Lin. LEVER: learning to verify language-to-code generation with execution. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 26106–26128. PMLR, 2023. URL https://proceedings.mlr.press/v202/ni23b.html.

Ansong Ni, Miltiadis Allamanis, Arman Cohan, Yinlin Deng, Kensen Shi, Charles Sutton, and Pengcheng Yin. Next: Teaching large language models to reason about code execution. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=B1W712hMBi.

Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron C. Courville. Asynchronous RLHF: faster and more efficient off-policy RL for language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=FhTAG591Ve.

NVIDIA, :, Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, et al. Nemotron 3 nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *CoRR*, abs/2512.20848, 2025. URL https://arxiv.org/abs/2512.20848.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=Cq1BNvHx74.

Yu Pan, Zhongze Cai, Huaiyang Zhong, Guanting Chen, and Chonghuan Wang. What matters in data for DPO? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=GrDEV4InKZ.

Indraneil Paul, Haoyi Yang, Goran Glavaš, Kristian Kersting, and Iryna Gurevych. Obscuracoder: Powering efficient code LM pre-training via obfuscation grounding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=VYvxrD7aS0.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. Pipelinerl: Faster on-policy reinforcement learning for long sequence generation. *CoRR*, abs/2509.19128, 2025. URL https://arxiv.org/abs/2509.19128.

Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=4fSSqpk1sM.

Julian Aron Prenner and Romain Robbes. Runbugrun – an executable dataset for automated program repair. *CoRR*, abs/2304.01102, 2023. URL https://arxiv.org/abs/2304.01102.

Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir R. Choudhury, Lindsey Decker, et al. Codenet: A large-scale AI for code dataset for learning a diversity of coding tasks. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/a5bfc9e07964f8dddeb95fc584cd965d-Abstract-round2.html.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (eds.), *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pp. 3505–3506. ACM, 2020. URL https://dl.acm.org/doi/10.1145/3394486.3406703.

Noam Razin, Zixuan Wang, Hubert Strauss, Stanley Wei, Jason D. Lee, and Sanjeev Arora. What makes a reward model a good teacher? an optimization perspective. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=7pufO0SJAC.

Cursor Research, :, Aaron Chan, Ahmed Shalaby, Alexander Wettig, Aman Sanger, Andrew Zhai, Anurag Ajay, Ashvin Nair, Charlie Snell, et al. Composer 2 technical report, 2026. URL https://arxiv.org/abs/2603.24477.

Juan Diego Rodriguez, Wenxuan Ding, Katrin Erk, and Greg Durrett. Rankalign: A ranking view of the generator-validator gap in large language models. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=rJOkPauru9.

Chi Ruan, Dongfu Jiang, Yubo Wang, and Wenhu Chen. Critique-coder: Enhancing coder models by critique reinforcement learning. *CoRR*, abs/2509.22824, 2025. URL https://arxiv.org/abs/2509.22824.

Amrith Setlur, Matthew Y. R. Yang, Charlie Victor Snell, Jeremiah Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e3: Learning to explore enables extrapolation of test-time compute for LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=aID0dZmMmM.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/7ede97c3e082c6df10a8d6103a2eebd2-Abstract-Conference.html.

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, et al. Spurious rewards: Rethinking training signals in rlvr. *CoRR*, abs/2506.10947, 2025. URL https://arxiv.org/abs/2506.10947.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *CoRR*, abs/2402.03300, 2024. URL https://arxiv.org/abs/2402.03300.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. Natural language to code translation with execution. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3533–3546, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.231. URL https://aclanthology.org/2022.emnlp-main.231.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K. Reddy. Execution-based code generation using deep reinforcement learning. *Transactions on Machine Learning Research*, 2023, 2023. URL https://openreview.net/forum?id=0XBuaxqEcG.

KaShun Shum, Binyuan Hui, Jiawei Chen, Lei Zhang, W. X., Jiaxin Yang, Yuzhen Huang, Junyang Lin, and Junxian He. Swe-rm: Execution-free feedback for software engineering agents. *CoRR*, abs/2512.21919, 2025. URL https://www.arxiv.org/abs/2512.21919.

Yuda Song, Gokul Swamy, Aarti Singh, J. Andrew Bagnell, and Wen Sun. The importance of online data: Understanding preference fine-tuning via coverage. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/16c628ab12dc4caca8e7712affa6c767-Abstract-Conference.html.

Yuda Song, Hanlin Zhang, Carson Eisenach, Sham Kakade, Dean Foster, and Udaya Ghai. Mind the gap: Examining the self-improvement capabilities of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 39894–39931, 2025.

Kaya Stechly, Karthik Valmeekam, Vardhan Palod, Atharva Gundawar, and Subbarao Kambhampati. Beyond semantics: The unreasonable effectiveness of reasonless intermediate tokens. In *First Workshop on Foundations of Reasoning in Language Models*, 2025. URL https://openreview.net/forum?id=lphc8Ssful.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, et al. Stop overthinking: A survey on efficient reasoning for large language models. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=HvoG8SxggZ.

Zhiqing Sun, Longhui Yu, Yikang Shen, Weiyang Liu, Yiming Yang, Sean Welleck, and Chuang Gan. Easy-to-hard generalization: Scalable alignment beyond human supervision. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5b6346a05a537d4cdb2f50323452a9fe-Abstract-Conference.html.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. Preference fine-tuning of llms should leverage suboptimal, on-policy data. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=bWNPx6t0sF.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y. Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. JudgeBench: A Benchmark for Evaluating LLM-based Judges. *CoRR*, abs/2410.12784, 2024. URL https://arxiv.org/abs/2410.12784.

Wentao Tan, Qiong Cao, Yibing Zhan, Chao Xue, and Changxing Ding. Beyond human data: Aligning multimodal large language models by iterative self-evolution. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pp. 7202–7210. AAAI Press, 2025. doi: 10.1609/AAAI.V39I7.32774. URL https://doi.org/10.1609/aaai.v39i7.32774.

Yunhao Tang, Daniel Zhaohan Guo, Zeyu Zheng, Daniele Calandriello, Yuan Cao, Eugene Tarassov, Rémi Munos, Bernardo Ávila Pires, Michal Valko, Yong Cheng, et al. Understanding the performance gap between online and offline alignment algorithms. *CoRR*, abs/2405.08448, 2024. URL https://arxiv.org/abs/2405.08448.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/ed3fea9033a80fea1376299fa7863f4a-Abstract-Conference.html.

Qian Wang, Zhanzhi Lou, Zhenheng Tang, Nuo Chen, Xuandong Zhao, Wenxuan Zhang, Dawn Song, and Bingsheng He. Assessing judging bias in large reasoning models: An empirical study. In *Second Conference on Language Modeling*, 2025a. URL https://openreview.net/forum?id=SlRtFwBdzP.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=1PL1NIMMrw.

Yubo Wang, Xiang Yue, and Wenhu Chen. Critique Fine-Tuning: Learning to Critique is More Effective than Learning to Imitate. *CoRR*, abs/2501.17703, 2025b. URL https://arxiv.org/abs/2501.17703.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling. *CoRR*, abs/2506.20512, 2025c. doi: 10.48550/ARXIV.2506.20512. URL https://doi.org/10.48550/arXiv.2506.20512.

Zihan Wang, Siyao Liu, Yang Sun, Ming Ding, and Hongyan Li. CodeContests+: High-quality test case generation for competitive programming. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 5576–5600, Suzhou, China, November 2025d. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.299. URL https://aclanthology.org/2025.findings-emnlp.299/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Xueru Wen, Jie Lou, Yaojie Lu, Hongyu Lin, XingYu, Xinyu Lu, Ben He, Xianpei Han, Debing Zhang, and Le Sun. Rethinking reward model evaluation: Are we barking up the wrong tree? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Cnwz9jONi5.

Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason Weston, Ilia Kulikov, and Swarnadeep Saha. J1: incentivizing thinking in llm-as-a-judge via reinforcement learning. *CoRR*, abs/2505.10320, 2025. URL https://arxiv.org/abs/2505.10320.

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

Yiqing Xie, Alex Xie, Divyanshu Sheth, Pengfei Liu, Daniel Fried, and Carolyn P. Rosé. Codebenchgen: Creating scalable execution-based code generation benchmarks. *CoRR*, abs/2404.00566, 2024. URL https://arxiv.org/abs/2404.00566.

Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. Iterative preference learning from human feedback: Bridging theory and practice for RLHF under kl-constraint. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=c1AKcA6ry1.

Wei Xiong, Jiarui Yao, Yuhui Xu, Bo Pang, Lei Wang, Doyen Sahoo, Junnan Li, Nan Jiang, Tong Zhang, Caiming Xiong, et al. A Minimalist Approach to LLM Reasoning: From Rejection Sampling to Reinforce. *CoRR*, abs/2504.11343, 2025. URL https://arxiv.org/abs/2504.11343.

Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. Is DPO superior to PPO for LLM alignment? A comprehensive study. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=6XH8R7YrSk.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. URL https://arxiv.org/abs/2505.09388.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Your efficient rl framework secretly brings you off-policy rl training. *Feng Yao's Notion*, 2025a. URL https://fengyao.notion.site/off-policy-rl.

Xinhao Yao, Lu Yu, Xiaolin Hu, Fengwei Teng, Qing Cui, Jun Zhou, and Yong Liu. The debate on RLVR reasoning capability boundary: Shrinkage, expansion, or both? A two-stage dynamic view. *CoRR*, abs/2510.04028, 2025b. doi: 10.48550/ARXIV.2510.04028. URL https://doi.org/10.48550/arXiv.2510.04028.

Chengzhi Yu, Yifan Xu, Yifan Chen, and Wenyi Zhang. Optimizing lvlms with on-policy data for effective hallucination mitigation. *CoRR*, abs/2512.00706, 2025a. URL https://arxiv.org/abs/2512.00706.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=2a36EMSSTp.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? *CoRR*, abs/2504.13837, 2025. URL https://arxiv.org/abs/2504.13837.

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. ACECODER: acing coder RL via automated test-case synthesis. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 12023–12040. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.587/.

Charlie Zhang, Graham Neubig, and Xiang Yue. On the interplay of pre-training, mid-training, and RL on reasoning language models. *CoRR*, abs/2512.07783, 2025a. doi: 10.48550/ARXIV.2512.07783. URL https://doi.org/10.48550/arXiv.2512.07783.

Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. Self-edit: Fault-aware code editor for code generation. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 769–787, Toronto, Canada, 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.45. URL https://aclanthology.org/2023.acl-long.45.

Tianyi Zhang, Tao Yu, Tatsunori Hashimoto, Mike Lewis, Wen-Tau Yih, Daniel Fried, and Sida Wang. Coder reviewer reranking for code generation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 41832–41846. PMLR, 2023b. URL https://proceedings.mlr.press/v202/zhang23av.html.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b.

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. *CoRR*, abs/2507.18071, 2025. URL https://arxiv.org/abs/2507.18071.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Yefan Zhou, Austin Xu, Yilun Zhou, Janvijay Singh, Jiang Gui, and Shafiq Joty. Variation in verification: Understanding verification dynamics in large language models. *CoRR*, abs/2509.17995, 2025. URL https://arxiv.org/abs/2509.17995.

Xiao Zhu, Xinyu Zhou, Boyu Zhu, Hanxu Hu, Mingzhe Du, Haotian Zhang, Huiming Wang, and Zhijiang Guo. Codescaler: Scaling code llm training and test-time inference via execution-free reward models. *CoRR*, abs/2602.17684, 2026. URL https://arxiv.org/abs/2602.17684.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The surprising effectiveness of negative reinforcement in LLM reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=ftVlLG9cks.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

Contents

1 Introduction **1**

2 Experimental Setup **3**

2.1 Testbed Creation 3

2.2 Evaluation Metrics 4

2.3 Training Details 5

3 Research Questions and Results **5**

3.1 RQ1: Do Code Verifiers Need to Generate Long Reasoning Traces? 5

3.2 RQ2: Is On-policy Learning Essential for Verifier Training? 7

3.3 RQ3: Do Negatives Benefit Code Verifiers? 9

4 Optimality Analysis **10**

5 Related Work **11**

6 Conclusion **12**

A Additional Experiment Details **24**

B Alternate Reward Formulations **25**

C Modifications for Aletheia-Adv **26**

D Alternative Approaches to Implement RAFT **26**

E Supporting Results **27**

E.1 Supervised Fine-tuning 27

E.2 Training on a Mixed Dataset 27

E.3 Inference-time scaling on RunBugRun 28

E.4 Verifier Response Parseability 29

F Prompt Templates **30**

A Additional Experiment Details

We use the implementation provided by Xiong et al. (2025) for RAFT and the trl library[2] for GRPO and DPO-Think. All training runs are conducted on a cluster of 8 NVIDIA H200 GPUs. To optimize memory usage, we employ Deepspeed ZeRO Stage-2 (Rasley et al., 2020) to shard activations and optimizer states across devices, and Flash-Attention 3 (Shah et al., 2024) to accelerate training. In all our training runs, we use the AdamW optimizer (Loshchilov & Hutter, 2019) with default parameters and a constant learning rate

[2] huggingface/trl

scheduler with 5% warmup steps, and train with an effective batch size of 64 for exactly 781 gradient steps to ensure a fair comparison.

Our GRPO implementation deviates from the original (DeepSeek-AI, 2025) to incorporate future recipe refinements. We use the DAPO loss (Yu et al., 2025b) and Truncated Importance Sampling (Yao et al., 2025a) with the truncation threshold set to 2.0. Although recent works have chosen to eliminate the KL coefficient, we set it to $\beta = 1e\text{-}3$ because our base models are already fine-tuned to generate long reasoning traces. We synchronize the reference model every 100 steps (Gorbatovski et al., 2025; Liu et al., 2025a). We use a learning rate of 1e-6 and normalize by the standard deviation within each group. We note that while Liu et al. (2025c) suggest batch-level normalization for base models, their results indicate poor performance for aligned models, such as those used in this study. We generate a batch of 64 prompts and perform a single gradient update per batch, with $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$. To encourage the model to stay within budget, we use a soft overlong punishment reward (Yu et al., 2025b).

For BO-GRPO, we use a generation batch of 256 prompts, performing 4 gradient updates per batch with $\epsilon_{\text{low}} = 3e\text{-}4$, $\epsilon_{\text{high}} = 4e\text{-}4$ and sequence-level importance sampling (Zheng et al., 2025). All other details are the same as the online GRPO variant.

Following Lambert et al. (2024), we train DPO with a learning rate of 5e-7, KL penalty $\beta = 0.1$, and an effective training batch size of 64. To reduce memory overhead, we precompute log-probabilities, eliminating the need to load the reference model during training. To train a DPO model, we also need an offline dataset of preferred and dispreferred generations. To this end, we create Aletheia-DPO by sampling 100 outputs for each prompt in Aletheia-Train using DeepSeek-R1-Distill-Qwen-[1.5-14]B, and score them with our verifiable reward function.

While prior work finds the quality of chosen responses to be more important (Pan et al., 2025b), we hypothesize that the reverse is true in a verifiable setting, where the quality of the "chosen" sample is fixed (correct), but the rejected quality can vary. Moreover, DPO is known to be sensitive to OOD shifts (Xu et al., 2024). Thus, we distribute the incorrect responses evenly between those generated by the 1.5-14B models. This also ensures that the negative samples for DPO are drawn from generations similar to those from on-policy sampling. Our hypothesis is validated by the strong performance of our DPO models, even rivaling the fully online GRPO at larger sizes.

RAFT is trained with a learning rate of 2e-6 and an effective batch size of 64. Consistent with Dong et al. (2023); Xiong et al. (2025), no KL penalty is applied. In preliminary runs, we found that fine-tuning on the entire batch of correct responses leads to overfitting, especially in large models that generate a high proportion of correct responses. We mitigate this effect by fine-tuning on a maximum of 5 correct responses per group (See Section D for more details).

B Alternate Reward Formulations

Shaping the reward during RL training is a crucial decision, and numerous proposals for optimal reward functions have been made in prior work. We experiment with four reward formulations at 7B model scale and pick the best-performing one for our final training runs. The rewards used are as follows:

- **Pairwise Exact Match (PairEM)**. The simplest formulation. Given two candidates, we prompt the verifier to indicate its preference with a single token (A or B) within boxed{}.
- **Pairwise Scores (PairSc)**. This reward is taken from the JudgeLRM paper (Gandhi et al., 2025). The verifier outputs scores on a scale of 0-10 for both candidate codes, and the reward is shaped based on accuracy, confidence, and format.
- **Listwise Exact Match (ListEM)**. A modified version of PairEM with between two and five candidates
- **Listwise Scores (ListSc)**. The verifier outputs scores on a 10-point scale for each candidate, similar to PairSc. If the correct code is assigned the highest score, we assign a reward of +1 and a bonus of +1 if this score is 10.

Both listwise rewards are loosely based on DeepSeek-GRM (Liu et al., 2025d), adapted to our setting. For PairSc and ListSc, we use the pass rate of both codes as an indication of their quality. Since one of the codes is always correct, one of the scores outputted by the model should always be 10. We train these models using GRPO as described in the main paper and present the results in Table 6. We find that relatively simple

Table 6: **Average List Accuracy results for alternate reward formulations we studied.** All results are from training the 7B model for an equal number of gradient updates using GRPO. For a fair comparison, we evaluate on code pairs, which explains the higher absolute values compared to Section 3.

Reward	ListAcc@1	ListAcc@2	ListAcc@4	ListAcc@8
PairSC	78.24 ± 0.93	78.13 ± 0.93	80.16 ± 1.0	82.22 ± 1.06
PairEM	77.19 ± 0.90	77.50 ± 0.90	78.93 ± 0.95	80.12 ± 1.00
ListSC	77.36 ± 0.89	77.32 ± 0.89	79.40 ± 0.94	80.82 ± 0.98
ListEM	**80.02 ± 0.92**	**80.14 ± 0.92**	**81.50 ± 0.94**	**83.02 ± 0.98**

ListEM works best, followed by PairSc. Moreover, we find that inference-time scaling trends hold even for these alternative reward formulations, providing limited benefits. This is also consistent with prior work that finds RLVR tends to sharpen the output distribution of models (Yue et al., 2025).

C Modifications for Aletheia-Adv

Table 7: **Modifications considered to construct Aletheia-Adv.** We report the Bias Influence Ratio (BIR) for the 7 − 32B models, along with the average. Positive modifications are applied to the incorrect code, whereas negative ones are applied to the correct one. The top six modifications are highlighted.

Name	Description	7B	14B	32B	Avg.
	Positive Biases				
Authority Bias	Claims the incorrect code was written by an experienced developer.	**0.56**	**0.67**	**0.67**	**0.64**
Egocentric bias	Indicates an incorrect code was written by the evaluator	0.52	0.49	0.54	0.52
External Reference	Claims to be the reference solution on the competition's website	**0.58**	**0.78**	**0.85**	**0.73**
Bandwagon Effect	Indicates that a majority of developers prefer the incorrect code.	0.51	0.55	0.55	0.54
Illusory Complexity	Garbage/unreachable code to elicit length bias (Zheng et al., 2023).	0.40	0.44	0.49	0.44
Self-declared correctness	States that the code is correct	**0.64**	**0.77**	**0.75**	**0.72**
	Negative Biases				
Minification	Code compressed using a rule-based minifier for C++ and Java, and python-minifier[3] for Python.	0.50	0.52	0.50	0.51
Misleading Comments	Comments indicate the correct code makes an error	**0.53**	**0.76**	**0.82**	**0.71**
Renaming Identifiers	Variable, class, and function names are obfuscated (Paul et al., 2025)	0.54	0.60	0.54	0.56
Reverse Authority Bias	Claims the incorrect code was written by a junior developer	**0.53**	**0.71**	**0.65**	**0.63**
Reverse Bandwagon Effect	Indicates that a minority of developers prefer the correct code.	0.44	0.60	0.56	0.53
Self-declared incorrectness	States that the code is incorrect	**0.60**	**0.81**	**0.86**	**0.76**

We experiment with several biasing factors for the creation of Aletheia-Adv (Table 7). To analyze the vulnerability of the base models to these factors, we prompt Deepseek-R1-Distill-Qwen2.5 from 7B to 32B parameters on the original and perturbed versions of the same prompt, and measure how often the evaluator switches its answer. Positive modifications are applied to all incorrect codes, while negative modifications are applied only to the correct code.

We conduct evaluations of perturbation effectiveness in a pairwise setting. We report the Bias Influence Ratio (BIR) as the ratio of the number of times the LLM switches to the incorrect answer to the total number of switches. A higher BIR indicates a stronger bias, while a BIR near 0.5 indicates random answer switching, which could be due to position biases in the model (Zheng et al., 2023). Overall, we verify that LRMs are more robust to common biases that are prevalent in LLMs, as observed in prior work (Wang et al., 2025a). However, they are not completely unbiased, and we select the top six most misleading modifications for analysis of adversarial robustness in the main text.

D Alternative Approaches to Implement RAFT

While designing our Negatives ablation in Section 3.3, we experimented with several variants of the RAFT algorithm. A crucial requirement was that our RAFT algorithm be fully on-policy to isolate the effect of negative samples. The original RAFT (Dong et al., 2023), akin to common rejection-sampling algorithms, is batch-online: sampling N responses each for a batch of prompts from the current model, scoring them using a reward model, and fine-tuning the current model on the K highest scoring prompt-response pairs using

a negative log-likelihood loss. Moreover, this algorithm was designed for RLHF-style reward models that output a continuous scalar score, unlike the binary reward signals prevalent in GRPO.

Our implementation closely follows Xiong et al. (2024), who adapt RAFT to the RLVR setting, with some modifications. Most importantly, we collect new data after a single gradient update (which slows training but ensures it remains fully on-policy). An unclear implementation detail in their work is whether they train on *all* the correct responses for a group or only a subset. We experiment with both variants, and the RAFT++ algorithm proposed by Xiong et al. (2024), on the 14B model as shown in Table 8.

Table 8: **Experimental results for RAFT variants.** We present the average ListAcc@1 scores across the four Aletheia- evaluation datasets, along with their 95% confidence interval.

Algorithm	ListAcc@1	Description
RAFT	51.20 ± 0.53	RAFT adapted to verifiable rewards, trained on all positives
RAFT-max5	**67.20 ± 0.52**	RAFT trained on a maximum of 5 correct responses per group
RAFT++	60.41 ± 0.47	Variant adding importance sampling and clipping to RAFT

Clearly, fine-tuning on all correct responses for each group leads to overfitting on easy examples, yielding a higher number of correct responses for training than for harder prompts. Surprisingly, adding PPO-style importance sampling and clipping techniques does not stabilize training either, further emphasizing the role of negative samples. We use the best performing max-5 variant of RAFT for all results presented in the main text.

E Supporting Results

E.1 Supervised Fine-tuning

We train models using the Supervised Fine-Tuning (SFT) objective on the positively scored responses from Aletheia-DPO (as described in Section A) across the 1.5-14B scale to further validate our analyses of the Negatives and Online components. Notably, we omit this algorithm from the main text because it differs from GRPO along two axes: Negatives and Online. However, in this section, we compare it to the other "incomplete" algorithms: DPO-Think, BO-GRPO, and RAFT. We report results in Table 9.

Removing two of the core components from GRPO significantly hampers performance across all model sizes. Our observations about the importance of on-policy learning decreasing with scale remain valid, as evidenced by the SFT–RAFT gap decreasing with scale (from 13.6% to 8.6%). Crucially, the utility of negative samples for stabilizing training is more pronounced in these ablations, as shown by comparing the SFT–RAFT gap to the SFT–(BO-GRPO) gap. Despite BO-GRPO being only partially on-policy, it outperforms the fully on-policy RAFT across all model scales due to the presence of negative samples. At small scales, using BO-GRPO: a mixture of on-policy training and negatives, is the best alternative to the full GRPO algorithm. However, at medium–large scales, using a completely offline algorithm (DPO-Think) can already yield good results at much lower cost, as mentioned in Section 4.

E.2 Training on a Mixed Dataset

A key advantage of the Aletheia testbed is the complete separation of training and evaluation data distributions, enabling an accurate estimation of the OOD robustness of our trained verifiers. However, this setting differs from the practical approach: training verifiers on a mix of all anticipated scenarios to achieve the best downstream performance. For completeness, we present the results from training a 1.5B model on a mixture of all our evaluation scenarios. Concretely, we utilized unused instances from the **Weak-Hard** and **Strong-Easy** buckets (i.e., instances that do not appear in the corresponding evaluation sets), and created a new adversarial dataset by perturbing Aletheia-Train using the six best modifications detailed in Section C. Crucially, we now apply the modifications at random rather than targeting either the correct or incorrect code, thereby avoiding information leakage about the ground truth and training the verifier to be robust to such perturbations. The resulting Aletheia-Mixed contains all four data distributions in equal proportions. We create a corresponding DPO dataset following the same procedure as in Section A, and summarize our results in Table 10.

Table 9: **Additional results on an SFT baseline.** We report `ListAcc` scores at `K=1`. SFT performs the worst overall due to the absence of two RLVR components. At larger scales, on-policy learning is the least critical factor, and negatives play a critical role in training stability for all sizes.

Thinking	Negatives	Online	Batch-online	Method	Size	Aletheia-Heldout	Aletheia-Strong	Aletheia-Hard	Aletheia-Adv	Average
-				Random		32.08	32.08	32.08	32.08	32.08
				SFT-Think	1.5B	20.27	18.23	18.30	18.00	18.70
	✓			DPO-Think	1.5B	21.70	19.90	23.41	19.37	21.11
	✓		✓	BO-GRPO	1.5B	43.13	39.89	39.26	33.83	38.99
		✓		RAFT	1.5B	34.76	31.88	33.67	29.12	32.30
	✓	✓		GRPO-Think	1.5B	49.58	46.09	40.74	40.97	44.38
				SFT-Think	7B	46.70	40.30	31.40	41.11	39.87
	✓			DPO-Think	7B	63.75	55.54	51.20	52.94	55.88
	✓		✓	BO-GRPO	7B	64.71	56.18	52.08	49.35	55.46
		✓		RAFT	7B	60.86	52.00	48.84	49.24	52.72
	✓	✓		GRPO-Think	7B	74.81	67.28	53.11	65.04	65.05
				SFT-Think	14B	66.53	60.30	48.53	59.04	58.60
	✓			DPO-Think	14B	82.56	74.39	67.58	71.06	73.89
	✓		✓	BO-GRPO	14B	83.82	76.33	67.34	73.45	75.29
		✓		RAFT	14B	75.55	66.02	65.23	62.03	67.20
	✓	✓		GRPO-Think	14B	88.02	83.65	66.84	83.67	80.54

Table 10: **Results from training `1.5B` verifiers on `Aletheia-Mixed`.** We report `ListAcc` scores at `K=1`. Training on a mixed dataset yields minor performance boosts over the `Aletheia` testbed in most cases, but sacrifices conclusions about OOD robustness.

Thinking	Negatives	Online	Batch-online	Algorithm	Aletheia-Strong	Aletheia-Hard	Aletheia-Adv
	✓	✓		GRPO-Instruct	37.11	29.68	31.61
	✓			DPO-Think	20.37	24.24	20.08
	✓		✓	BO-GRPO	40.11	39.19	33.42
		✓		RAFT	32.99	34.06	29.67
	✓	✓		GRPO-Think	41.46	39.05	34.09

Training on a mixture of all tasks yields small performance gains for most algorithms, as compared to the results in Tables 3 to 5. More importantly, our conclusions from the main text at the `1.5B` scale still hold. `DPO` performs the worst at this scale, and introducing even a semi-on-policy update can close most of the offline-online performance gap, making on-policy learning the most crucial component at this scale.

Crucially, training on a mixed dataset precludes analysis of the OOD robustness of the studied algorithms. Since `Aletheia-Mixed` contains a mixture of all anticipated evaluation scenarios, all three evaluation axes are in-distribution, providing no signal on the robustness of these verifiers in a downstream RLVR pipeline. Thus, by completely separating the training and evaluation data distributions, the `Aletheia` testbed provides a foundation for our controlled analysis, enabling us to stress-test verifiers in a proxy-evaluation setting without incurring prohibitive costs.

E.3 Inference-time scaling on `RunBugRun`

To ensure the validity of our inference-time scaling observations across reward formulations, we further test our hypothesis on the `RunBugRun` dataset (Prenner & Robbes, 2023), which contains correct-bugged code pairs from CodeNet (Puri et al., 2021). We report our results in Table 11.

The modest gains from inference-time scaling persist across an external dataset, demonstrating that our observation is a consistent pattern rather than an artifact of our specific setup. Moreover, it is supported by prior literature on entropy collapse in LLMs, which finds that RLVR merely improves the sampling efficiency of language models and can shrink the space of accessible reasoning paths (Yue et al., 2025; Wu et al., 2025).

Table 11: **Inference scaling trends on the RunBugRun dataset.** We report the ListAcc and its 95% confidence interval. GRPO-trained models see limited gains from inference compute scaling.

Reward	RBR PairAcc@1	RBR PairAcc@2	RBR PairAcc@4	RBR PairAcc@8
PairSC	67.43 ± 1.12	67.23 ± 1.12	69.48 ± 1.21	71.08 ± 1.30
PairEM	70.84 ± 1.14	70.83 ± 1.13	72.99 ± 1.22	73.61 ± 1.30
ListSC	71.01 ± 1.21	71.26 ± 1.21	73.12 ± 1.29	74.17 ± 1.37
ListEM	**73.24 ± 1.14**	**73.63 ± 1.13**	**74.99 ± 1.22**	**75.61 ± 1.30**

E.4 Verifier Response Parseability

The parse rate of a verifier is defined as the percentage of its responses that terminate with a valid and parseable verdict. In all the verifiers studied in the main text, this implies that the verifier responds with a valid option within \boxed{}. The models are trained to respond with such a format, either through demonstrations or via a format reward function. We report the parse rates for all models in Table 12. As expected, most models achieve near-perfect parse rates, since they are being explicitly trained to do so.

However, a notable exception is DPO-Think-1.5B, which has a very poor parse rate of only 43% on average. This is because the model degenerates into producing non-terminating sequences. Therefore, such a verifier has very limited utility in practice despite seemingly high Kτ scores. These high scores are simply explained by the fact that Kτ effectively treats unparseable responses as a tie between the two candidates, rather than as a failure mode of the verifier as in ListAcc (which expectedly degrades to sub-random performance). This degeneration is likely an artifact of the small model size rather than the offline dataset, since the larger DPO-Think verifiers have high parse rates. Thus, we suggest against using Kτ alone as a measure of verifier performance, and instead augment it with a parse rate check.

Table 12: **Parse rates for the algorithms studied in this work.**

Thinking Negatives Online Batch-online

Algorithm	Size	Aletheia-Heldout	Aletheia-Strong	Aletheia-Hard	Aletheia-Adv	Average
(§3.1) GRPO-Instruct		99.97	100.00	100.00	100.00	99.99
(§3.1) GRPO-Think-4k		99.78	99.72	99.14	99.30	99.49
(§3.1) GRPO-Think-8k		99.14	99.54	98.55	99.40	99.16
(§3.2) DPO-Think	1.5B	40.10	41.15	49.01	42.64	43.23
(§3.2) BO-GRPO		98.98	98.80	97.81	98.73	98.58
(§3.3) RAFT		71.50	74.58	75.48	72.03	73.40
(§3) GRPO-Think-16k		99.44	99.32	98.80	99.62	99.30
(§3.1) GRPO-Instruct		100.00	100.00	100.00	100.00	100.00
(§3.1) GRPO-Think-4k		98.64	99.17	97.13	99.05	98.50
(§3.1) GRPO-Think-8k		98.43	98.43	98.58	98.61	98.51
(§3.2) DPO-Think	7B	94.08	95.62	91.98	95.19	94.22
(§3.2) BO-GRPO		95.56	97.22	92.23	96.01	95.26
(§3.3) RAFT		83.53	84.67	85.56	85.18	84.74
(§3) GRPO-Think-16k		92.91	95.06	90.62	94.20	93.20
(§3.1) GRPO-Instruct		99.85	99.88	99.97	99.71	99.85
(§3.1) GRPO-Think-4k		95.56	97.16	96.30	95.79	96.20
(§3.1) GRPO-Think-8k		99.88	99.94	100.00	99.90	99.93
(§3.2) DPO-Think	14B	99.54	99.63	99.01	99.62	99.45
(§3.2) BO-GRPO		99.04	99.44	98.37	99.11	98.99
(§3.3) RAFT		97.16	97.62	96.88	96.74	97.10
(§3) GRPO-Think-16k		97.72	98.03	90.44	97.91	96.03

F Prompt Templates

Default training prompt

```
You are an expert judge of coding problems. Given a coding problem and multiple candidate solutions, your
task is to evaluate the correctness of each solution based on the problem description. Your evaluation
should solely be based on the functional correctness of the codes. It is guaranteed that exactly one of
the candidates is completely correct. Here is the coding question followed by the candidate solutions:
[QUESTION] {question} [/QUESTION]


[CANDIDATE_A] {code_A} [/CANDIDATE_A]

[CANDIDATE_B] ...
Indicate your choice of candidate only by responding with one of the following options: {valid_options}.
Enclose your final answer in the format \boxed{X}, where X is your chosen option among the candidates.
Do not provide any additional text. Your response should be exactly one of the options enclosed within
\boxed{}, without any extra characters or spaces. Anything else will be considered invalid.
```

GRPO-Instruct training prompt

```
You are an expert judge of coding problems. Given a coding problem and multiple candidate solutions,
your task is to evaluate the correctness of each solution based on the problem description. Your
evaluation should solely be based on the functional correctness of the codes. It is guaranteed that
exactly one of the candidates is completely correct. Indicate your choice of candidate by responding
with one of the following options: {valid_options}. Your response should be in the following format:
Analysis: <Your step-by-step reasoning here>
Final Answer: \boxed{X}, where X is your chosen option among the candidates.

Here is the coding question followed by the candidate solutions:
[QUESTION] {question} [/QUESTION]

[CANDIDATE_A] {code_A} [/CANDIDATE_A]

[CANDIDATE_B] ...
Your response should be exactly in the specified format, without any extra characters or spaces.
Anything else will be considered invalid.
```

ListSc and PairSc training prompt

```
You are an expert judge of coding problems. Given a coding problem and two candidate solutions, your
task is to evaluate the correctness of each solution based on the problem description. Your evaluation
should solely be based on the functional correctness of the codes. It is guaranteed that exactly one of
the candidates is completely correct. Here is the coding question followed by the candidate solutions:
[QUESTION] {question} [/QUESTION]

[CANDIDATE_A] {code_A} [/CANDIDATE_A]

[CANDIDATE_B] ...
Assign a score between 0 and 10 to EACH candidate, with 10 indicating a perfect solution that
passes all test cases, 5 indicating a solution that would pass some test cases but not all,
and 0 indicating a solution that fails all test cases. Output your final answer in the format
\boxed{[<score_candidate_A>,<score_candidate_B>, <score_candidate_C>, ...]} for each input candidate.
Do not provide any additional text. Your response should be a list of numbers between 0 and 10, enclosed
within \boxed{}, without any extra characters or spaces. Anything else will be considered invalid.
```

Python code generation prompt

You are an expert Python programmer. You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests. Read the inputs from STDIN solve the problem and write the answer to STDOUT (do not directly test on the sample inputs). Enclose your code within a Python markdown block. Ensure that when the python program runs, it reads the inputs, runs the algorithm and writes output to STDOUT.

C++ code generation prompt

You are an expert C++ programmer. You will be given a question (problem specification) and will generate a correct C++ program with a main function that matches the specification and passes all tests. Read the inputs from STDIN solve the problem and write the answer to STDOUT (do not directly test on the sample inputs). Enclose your code within a C++ markdown block. Ensure that when the C++ program runs, it reads the inputs, runs the algorithm and writes output to STDOUT.

Java code generation prompt

You are an expert Java programmer. You will be given a question (problem specification) and will generate a correct Java program with a public class named Main that matches the specification and passes all tests. Your class should include a public static void main(String[] args) method. Read the inputs from System.in solve the problem and write the answer to System.out (do not directly test on the sample inputs). Enclose your code within a Java markdown block. Ensure that when the Java program runs, it reads the inputs, runs the algorithm and writes output to System.out.