Learning from Saturated Data: Signals Beyond Correctness for LLM Training

Hanno Hiss [1] **Jasper Dekoninck** [1] **Martin Vechev** [1]

Abstract

The growing capabilities of large language models (LLMs) have led to the saturation of many benchmarks and training datasets used to improve them. Motivated by this, we investigate whether questions solved with perfect empirical accuracy can nevertheless be used to improve downstream performance. To do so, we replace binary correctness with two sources of more fine-grained quality signals: (1) pairwise LLM self-judgments, in which the model evaluates the relative quality of its own solutions, and (2) token-level entropy, where token-level uncertainty is used as a proxy for solution quality. We incorporate these signals into several training algorithms and evaluate them on Qwen3-1.7B-Base. When training exclusively on a simple arithmetic task, quality-based signals improve performance by up to 18.6% over the base model, substantially outperforming SFT. On GSM8K, however, gains are more modest and depend strongly on the quality signal. For instance, self-judgments show poor agreement with a stronger external judge and can even degrade performance below the base model. Overall, our results suggest that quality-based training can extract useful signal from saturated questions for base models, but that applying such signals to more complex tasks requires careful calibration and further study.

1. Introduction

Large language models (LLMs) have advanced rapidly, with successive generations achieving substantial gains on mathematical reasoning, code synthesis, and other challenging benchmarks (Brown et al., 2020; DeepSeek-AI et al., 2025; Yang et al., 2025a). Yet the very success of these models creates a new problem: as models grow more capable, they saturate the benchmarks and training sets used to improve them. As a result, training data that the model already solves reliably are not useful in current training regimes.

Recent work has begun to acknowledge this problem. Wang et al. (2025) show that reinforcement learning (RL) with a single training example can improve reasoning performance beyond training-set saturation, though most of the reported gains are attributed to format correction rather than actual reasoning improvements. Further, Agarwal et al. (2025) demonstrate that entropy minimization alone can match GRPO-level gains without any labeled data, but they use entropy as a global training objective rather than as a quality metric to differentiate among correct completions. This leads us to ask:

Can we extract a learning signal from questions the model empirically solves with perfect accuracy?

As shown in Figure 1, we study this question by examining quality metrics that distinguish among correct solutions, since even correct model responses can differ in reasoning clarity, conciseness, and computational efficiency. We leverage these quality differences by using them in training and investigate whether the accuracy of the model on harder questions can be improved by training on quality-ranked solutions to saturated questions.

On these saturated questions, we rank correct completions by quality using two complementary mechanisms. First, we use a *self-judge* that scores completions pairwise by prompting the model to select the better of two completions. Second, we use *self-uncertainty* as a proxy for quality, ranking completions by the inverse mean token-level entropy of the model's output distribution. We then use these signals with two different training algorithms: Direct Preference Optimization (DPO) (Rafailov et al., 2024), which trains on chosen-rejected pairs constructed from the rankings, and Logistic-Weighted RRHF (σ-RRHF), a variant of RRHF (Yuan et al., 2023) that uses a logistic weighting function to assign higher importance to higher-ranked completions while still learning from all samples.

In our experiments, we train on the `chain sum` arithmetic task from ReasoningGym (Stojanovski et al., 2025) and on GSM8K (Cobbe et al., 2021), with `Qwen3-1.7B-Base` (Yang et al., 2025a) as the base model.

Code: `https://github.com/eth-sri/saturated-learning`. [1]ETH Zurich. Correspondence to: Hanno Hiss <hhiss@ethz.ch>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026. Copyright 2026 by the author(s).

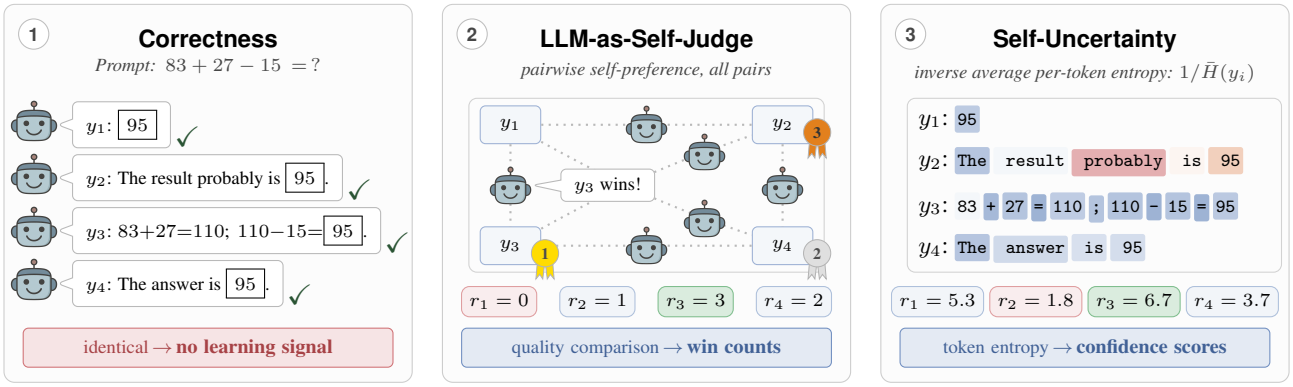


Figure 1. Quality signals on saturated questions. **(1)** *Correctness*: when all sampled completions are correct, binary reward provides no contrast. **(2)** *LLM-as-a-Judge*: a judge runs all pairwise comparisons, the win count w_i ranks completions by judged quality. **(3)** *Inverse Entropy*: mean token-level entropy along each completion gives a judge-free quality proxy.

Our main findings are:

1. **Easy-to-hard transfer on `chain sum`:** σ-RRHF with inverse-entropy-ranked completions improves pass@1 by $+18.6\%$ over the base model when trained on the saturated subset of the evaluation distribution, and still yields $+11.7\%$ over base when trained on saturated samples from a strictly easier distribution.

2. **Scorer failures on GSM8K:** In contrast, the self-judge anti-correlates with a strong external judge on GSM8K and degrades σ-RRHF below the base model. Inverse entropy helps the base model ($+3.3\%$), but fails to do so for its instruction-tuned variant.

Together, the results show that quality differences among correct completions can extend training past saturation, with scorer reliability as the central bottleneck. Our key contributions are:

- We identify saturated questions as an important training setting and identify quality-based signals as a promising way to train on these questions.

- We compare self-judging and inverse-entropy rankings across synthetic and GSM8K tasks, showing how scorer reliability determines whether easy-question training helps or hurts.

2. Related Work

Easy-to-hard generalization and data efficiency Training on easy examples can transfer surprisingly well to harder tasks. SFT on easy data can be competitive with models trained on hard data (Hase et al., 2024), and a reward model trained on easy data can supervise hard data training (Sun et al., 2024). Relatedly, labels from a weak model can elicit stronger capabilities in a strong learner (Burns et al., 2023). Finally, Wang et al. (2025) show that continued RL on a single example improves performance on MATH500, though most of the gain is attributed to format correction. We take these works one step further by focusing on fully saturated training data, rather than just easy data where the model is strong but not perfect.

Reward signals The choice of reward signal is critical in post-training, and several recent works have explored alternatives to correctness-based rewards. When verifiable rewards are unavailable, dedicated outcome reward models (ORMs) are used to judge the complete trace, process reward models provide step-level supervision (Zheng et al., 2025), and rubric-based rewards (Huang et al., 2025; Gunjal et al., 2025) provide structured criteria for subjective outputs, but can be tedious to construct. Both alternative metrics we study are outcome-level signals.

Self-improvement One of our metrics explicitly uses the model itself as a judge, leading to a process of self-improvement, which has been explored extensively in prior work, e.g., by using majority voting (Shafayat et al., 2025), multi-role self-play (Yang et al., 2025b), meta-judge training (Wu et al., 2025), or token-probability self-judging (Garg et al., 2025). However, so far, none of these works has focused on the regime of saturated training data.

Best-of-n selection Sampling multiple completions and selecting the best one is a general technique for improving model performance (Nakano et al., 2022), which has been adapted for online SFT training (Dong et al., 2023b). Several recent methods extend this to full rankings (Niu et al., 2025; Choi et al., 2026; Song et al., 2024). However, these methods have not yet been applied to the regime of saturated training data.

Entropy as signal Predictive uncertainty over generated tokens has been proposed in prior work as a proxy for output quality, including token-level entropy for translation quality estimation (Fomicheva et al., 2020), semantic entropy for free-form QA correctness (Kuhn et al., 2023), and epistemic uncertainty for hallucination detection (Xiao & Wang, 2021). Beyond its use as a passive signal, entropy minimization is an effective training objective by itself, improving reasoning performance without any labels (Agarwal et al., 2025).

3. Methods

We develop a methodology for extracting a training signal from saturated questions and investigate whether the resulting improvements transfer to hard questions the model cannot yet reliably solve. We first partition questions by difficulty to isolate saturated-to-hard transfer (Sec. 3.1), then introduce two quality signals for ranking correct completions (Sec. 3.2), and finally apply them within DPO (Sec. 3.3) and a logistic-weighted RRHF variant (Sec. 3.4).

Preliminaries An LLM M is characterized by a probability distribution π_M over sequences of tokens in a vocabulary $\mathcal{V}$, generating completions y from prompts x. A reward function $R(x, y) \in \mathbb{R}$ scores completion quality. We sample n completions per problem and write $\{y_i\}_{i=1}^n$ for this group, with $r_i = R(x, y_i)$ for their rewards. The model's uncertainty at each step is captured by the *token-level entropy*:

$$H_t(x, y_{<t}) = -\sum_{v \in \mathcal{V}} \pi_M(v \mid x, y_{<t}) \log \pi_M(v \mid x, y_{<t}). \tag{1}$$

3.1. Difficulty Partitioning

We consider two related datasets: $\mathcal{D}$, the target distribution of interest, and $\mathcal{D}'$, which is drawn from a strictly simpler distribution than $\mathcal{D}$. For instance, $\mathcal{D}$ could represent research-level mathematics problems, while $\mathcal{D}'$ could represent high-school-level ones. We estimate the model's per-question solve rate by the empirical accuracy over n samples, $\hat{p}(x) = \frac{1}{n}\sum_i r_i$, and define two saturated splits:

$$\mathcal{D}_{\text{saturated}} = \{x \in \mathcal{D} : \hat{p}(x) = 1\} \text{ and}$$
$$\mathcal{D}'_{\text{strict}} = \{x \in \mathcal{D}' : \hat{p}(x) = 1\}.$$

This distinction allows us to isolate saturated-to-hard transfer and makes it easier to analyze the relationship between training signals and accuracy improvements. The question we seek to answer is whether we can improve accuracy on $\mathcal{D} \setminus \mathcal{D}_{\text{saturated}}$, solely with access to $\mathcal{D}'_{\text{strict}}$.

3.2. Quality Scoring

Having filtered for saturated questions, training algorithms additionally require a signal to optimize. Since all completions are (by definition) correct, we need a quality signal to differentiate them and construct training pairs. We consider two complementary signals: pairwise LLM judgments from the model itself or a stronger teacher, and token-level entropy as a quality heuristic.

3.2.1. LLM-AS-A-JUDGE

We define a *judge* as a function $J(x, y_i, y_j) \in \{0, 1\}$ that returns 1 if $y_i \succ y_j$ and 0 otherwise. Each unordered pair $\{y_i, y_j\}$ is compared exactly once. The *win-rate score* of completion y_i is its fraction of won comparisons:

$$r_i^J = \frac{1}{n-1} \sum_{j \neq i} \mathbf{1}[y_i \succ y_j]. \tag{2}$$

When $J = J_M$ is implemented by the policy model π_M itself, we call this *self-judging*. The prompts used are in Sec. D.

3.2.2. SELF-UNCERTAINTY

Prior work has established predictive uncertainty over generated tokens (including token-level entropy and related semantic and epistemic measures) as a reliable proxy for output quality (Fomicheva et al., 2020; Kuhn et al., 2023; Xiao & Wang, 2021). We exploit this as a judge-free alternative to pairwise LLM evaluation. We define the inverse entropy score as:

$$r_i^H = 1/\max(\bar{H}(x, y_i), \epsilon),$$

where $\bar{H}(x, y_i) = \frac{1}{T}\sum_{t=1}^{T} H_t(x, y_{i,<t})$ is the mean token-level entropy of completion y_i (Eq. (1)), a scalar measure of how deterministic the response is, and a small ϵ guards against division by zero. Because $\bar{H}$ is averaged over length, this score does not favor shorter completions. Post-training and instruction-tuning systematically lower token-level entropy (Yu et al., 2025; Cao et al., 2026), so selecting low-entropy completions from a base model may approximate the structured distribution instruction-tuning would induce.

3.3. DPO on Saturated Questions

DPO requires a preference pair, a single chosen and a single rejected completion, per question for training. However, on $\mathcal{D}_{\text{saturated}}$, correctness can no longer distinguish the chosen completion from the rejected one. Instead, these pairs are constructed purely from quality scores, $y_w = \arg\max_{y_i} r_i$ and $y_l = \arg\min_{y_i} r_i$,[1] where r_i is the score assigned by the selected quality signal.

[1]Ties at the top or bottom are broken by random selection; if all scores are equal, the question is skipped.

3.4. Logistic-Weighted RRHF (σ-RRHF)

RRHF (Yuan et al., 2023) aligns model probabilities with reward rankings via a hinge-style loss over response pairs. In particular, given n completions $\{y_i\}_{i=1}^n$, the method decomposes the loss into two parts. First, a *rank loss* penalizes pairs where a lower-reward response receives higher model probability. Formally, the method assigns each response a length-normalized log-probability score $p_i = \frac{\log \pi_M(y_i|x)}{|y_i|}$ and defines the rank loss as:

$$\mathcal{L}_{\text{rank}} = \sum_{r_i < r_j} \max(0,\, p_i - p_j). \tag{3}$$

Second, an *SFT loss* anchors the model to the highest-reward response, $y_{i'}$ with $i' = \arg\max_i r_i$:

$$\mathcal{L}_{\text{ft}} = -\log \pi_M(y_{i'} \mid x). \tag{4}$$

The total RRHF loss is $\mathcal{L} = \mathcal{L}_{\text{rank}} + \mathcal{L}_{\text{ft}}$. To apply RRHF to our setting, we directly optimize the model to align its probabilities with the quality-based rankings induced by the selected quality signal.

Logistic weighting In the vanilla rank loss of Eq. (3), the penalty for a violated pair is determined solely by the log-probability margin $\max(0,\, p_i - p_j)$, regardless of how far apart the two responses are in quality. We add a multiplicative weight equal to the logistic sigmoid of the score gap:

$$\mathcal{L}_{\text{rank}}^{\sigma} = \sum_{r_i < r_j} \sigma(r_j - r_i)\, \max(0,\, p_i - p_j),$$

where σ is the logistic sigmoid. When two responses have nearly identical quality scores, the penalty is halved. When the gap is large, the weight approaches 1. The total loss combines the weighted rank loss with the SFT anchor on the best response:

$$\mathcal{L} = \lambda\, \mathcal{L}_{\text{rank}}^{\sigma} + \mathcal{L}_{\text{ft}},$$

where the rank loss weight λ balances the ranking signal against the SFT regularization. In standard RRHF, both terms are weighted equally ($\lambda = 1$).

4. Results

We evaluate the methods developed previously on saturated `chain sum` questions (Sec. 4.2). We further test generalization to GSM8K and find that the findings transfer only partially (Sec. 4.3). Finally, we apply the methods to an instruction-tuned model to test limitations of entropy-based scoring.

4.1. Experimental Setup

Task and data We use the `chain sum` task from ReasoningGym (Stojanovski et al., 2025), in which a sequence of additions and subtractions must be evaluated left to right. A representative instance is shown in Sec. A.3. Task difficulty is controlled via two parameters: the number of terms and the number of digits per operand.

We use `Qwen3-1.7B-Base` (Yang et al., 2025a) as our base model throughout all experiments. All methods share the same generation pool: `chain sum` problems with terms and digits ranging from 6 to 10, each with $n = 8$ sampled generations from the base model. From this we construct the saturated split $\mathcal{D}_{\text{saturated}}$ as defined in Sec. 3.1. For $\mathcal{D}'_{\text{strict}}$, we use problems with terms and digits ranging from 3 to 6, excluding 6/6. Further details on evaluation protocol and training are in Sec. B.

To probe whether quality differences matter beyond chance, we additionally include a random baseline that assigns each completion a score uniformly at random. We refer to this baseline, self-judge, strong judge, and inverse entropy as *scoring methods*, and instantiate the algorithms with each where applicable.

4.2. Learning from *Saturated* Questions

We train on questions the model already solves consistently, first on $\mathcal{D}_{\text{saturated}}$ and then on the stricter $\mathcal{D}'_{\text{strict}}$ partition. For both splits, we compare self-judge and inverse-entropy scoring, and include `Qwen3-30B` as a stronger external judge on $\mathcal{D}'_{\text{strict}}$ to contextualize the quality of these signals. The results are presented in Table 1.

Methods on saturated questions On $\mathcal{D}_{\text{saturated}}$, all training methods improve over the base model in at least one metric, but the gains depend strongly on the scoring signal. SFT already improves substantially over the base model, making it the natural reference point for methods that claim to extract additional signal from saturated data. This separates two effects: saturated examples are useful even under ordinary imitation, but quality-ranked methods must show gains beyond copying accepted correct completions. Self-judged DPO is comparatively weak: it improves over the base model, but remains below SFT. Inverse-entropy DPO mainly helps pass@8, while the clearest benefit from quality ranking appears in σ-RRHF. With self-judge scoring, σ-RRHF exceeds SFT on pass@1 and roughly matches its pass@8; with inverse entropy, it achieves the strongest result: 29.25% pass@1 and 61.75% pass@8. We ablate this gain in Sec. C.4. The strength of inverse entropy is surprising because it requires no external evaluator. One possible explanation is model compatibility: low-entropy completions may be solutions the model can already represent stably, so training on them reinforces robust computations rather

Table 1. Method comparison on saturated chain-sum problems with two training splits. **Bold** is best per column.

Model / Training	Scoring	$\mathcal{D}_{\text{saturated}}$		$\mathcal{D}'_{\text{strict}}$	
		pass@1	pass@8	pass@1	pass@8
`Qwen3-1.7B-Base`		10.63	51.50	10.63	51.50
+ SFT		19.38	55.50	14.62	51.00
+ DPO	Self-judge	13.97	52.75	13.56	53.00
+ DPO	Inv. Entropy	16.28	58.75	13.91	53.75
+ σ-RRHF	Self-judge	21.16	54.75	18.75	56.00
+ σ-RRHF	Inv. Entropy	**29.25**	**61.75**	**22.35**	**59.25**
+ σ-RRHF	`Qwen3-30B`	–	–	19.69	58.00

than pushing the model toward preferences defined outside its own distribution. This is unlikely to reflect mere distribution sharpening, which would tend to leave pass@8 flat or lower as diversity collapses; instead, inverse-entropy σ-RRHF raises pass@8 alongside pass@1.

Strictly saturated questions We extend the analysis to $\mathcal{D}'_{\text{strict}}$, where questions are intrinsically simpler by construction. This tests a stronger form of transfer: the training questions are not only saturated, but also easier than the evaluation distribution. SFT still improves pass@1 over the base model, confirming that very easy saturated examples contain reusable algorithmic signal. The smaller gains relative to $\mathcal{D}_{\text{saturated}}$ suggest that part of the benefit comes from distributional overlap with the harder evaluation problems.

Under self-judge scoring, DPO changes very little between $\mathcal{D}_{\text{saturated}}$ and $\mathcal{D}'_{\text{strict}}$ and remains below SFT on pass@1; we refer to Sec. C.5 for a possible explanation. In contrast, σ-RRHF is more sensitive to the quality signal: it improves over SFT with self-judge scoring on both metrics, and inverse entropy gives the strongest $\mathcal{D}'_{\text{strict}}$ result.

To compare against a stronger external quality signal on $\mathcal{D}'_{\text{strict}}$, we replace the self-judge with `Qwen3-30B`, a stronger instruction-tuned model. This improves over the self-judged σ-RRHF row (19.69% pass@1, 58.00% pass@8). However, the strongest $\mathcal{D}'_{\text{strict}}$ result surprisingly comes from σ-RRHF with inverse entropy (22.35% pass@1, 59.25% pass@8). This is consistent with the model-compatibility explanation above: inverse entropy can outperform the strong judge despite using only the base model's token probabilities.

4.3. GSM8K Experiments

To test whether our findings transfer to another task, we repeat key experiments on GSM8K (Cobbe et al., 2021). We construct $\mathcal{D}_{\text{saturated}}$ from the GSM8K *train* split using the same mechanism as for `chain sum` (Sec. 3.1), yielding $|\mathcal{D}_{\text{saturated}}| = 2{,}302$ questions ($\hat{p}(x) = 1$, estimated for `Qwen3-1.7B-Base`). All evaluation uses the held-out GSM8K *test* split, disjoint from $\mathcal{D}_{\text{saturated}}$.

Table 2. GSM8K accuracy for DPO and σ-RRHF variants trained on $\mathcal{D}_{\text{saturated}}$, evaluated on the test set. Inv. Entropy = inverse entropy. All σ-RRHF runs use $\lambda = 0.1$. **Bold** is best per column.

Model / Training	Scoring	pass@1	pass@8
`Qwen3-1.7B-Base`		67.83	93.25
+ SFT		68.10	93.70
+ DPO	`Qwen3-30B`	**75.52**	**94.92**
+ σ-RRHF	`Qwen3-30B`	70.29	93.48
+ DPO	Inv. Entropy	71.17	94.01
+ σ-RRHF	Inv. Entropy	70.50	93.80
+ DPO	Random	67.74	93.56
+ σ-RRHF	Random	64.57	94.09
+ DPO	Self-judge	69.34	93.71
+ σ-RRHF	Self-judge	55.43	92.57

Methods on GSM8K We compare DPO and σ-RRHF on $\mathcal{D}_{\text{saturated}}$ from GSM8K. We compare the same scoring family used above: random scoring, the self-judge, a strong judge, and inverse entropy (Table 2).

For DPO, downstream accuracy improves by $+7.7\%$ for the strong judge, $+3.3\%$ for inverse entropy, $+1.5\%$ for self-judge, and -0.1% for random scoring. σ-RRHF with inverse entropy and the strong judge both improve pass@1 while preserving pass@8, but the self-judge degrades performance well below the base model. Overall, σ-RRHF underperforms DPO across all scoring methods, plausibly because DPO's implicit KL constraint limits divergence from the reference model, whereas σ-RRHF's only regularization is its SFT anchor.

Scorer agreement on GSM8K To investigate why the performance improvements are more modest on GSM8K, we measure how closely each scoring method agrees with

Table 3. GSM8K accuracy for DPO variants starting from `Qwen3-1.7B`. + DPO pairs are constructed from quality scores on saturated questions ($\mathcal{D}_{\text{saturated}}$); + DPO (correctness) instead uses correct-vs-incorrect pairs from hard questions. Inv. Entropy = inverse entropy.

Model / Training	Scoring	pass@1	pass@8
`Qwen3-1.7B`		84.32	93.71
+ DPO	Self-judge	84.14	93.78
+ DPO	Inv. Entropy	82.13	92.57
+ DPO	`Qwen3-30B`	84.44	93.93
+ DPO (correctness)	`Qwen3-30B`	84.95	94.16

the strong judge via within-question Spearman rank correlation, aggregated over $\mathcal{D}_{\text{saturated}}$. On GSM8K, we find moderate positive correlation between inverse entropy and the strong judge ($\rho = 0.56$), supporting entropy as a selection criterion. In contrast, the self-judge and strong judge slightly disagree on GSM8K ($\rho = -0.2$), compared to a weak positive correlation on `chain sum` ($\rho = 0.19$).

This explains why the model trained with the self-judge metric underperforms even the base model: the self-judge provides actively misleading signal on GSM8K. σ-RRHF is plausibly more affected than DPO because it fits the full ranking over all n completions rather than only the best-worst pair, amplifying the systematic bias indicated by the negative self-judge/strong-judge Spearman correlation.

4.4. Instruction-Tuned Models

We test whether entropy-based DPO transfers to instruction-tuned models by repeating the GSM8K DPO experiments on the instruction-tuned variant of the same model family. As shown in Table 3, no method yields substantial improvements over the base model, and inverse entropy DPO even degrades performance. This contrasts with the base model, where inverse entropy DPO improved performance significantly. This is likely due to the low-entropy bias of instruction-tuning: further training to lower entropy is much less likely to yield quality improvements and can instead over-optimize the proxy.

5. Limitations and Future Work

We briefly discuss three limitations of our work and directions for future research.

First, our experiments use a single model family (`Qwen3-1.7B-Base`) and reveal that the relative effectiveness of training methods does not transfer straightforwardly across domains: σ-RRHF dominates on `chain sum`'s constrained output space, while DPO outperforms it on GSM8K's noisier free-form reasoning. We also evaluate only on the target task, and do not measure whether quality-based training affects other capabilities, such as general reasoning or output diversity beyond pass@8. Investigating these dynamics on larger model scales and tasks is left as future work.

Second, we focus on only two quality metrics, self-judging and inverse entropy, but there are many other potential metrics that could be explored. For instance, rubric-based rewards (Huang et al., 2025; Gunjal et al., 2025) could be adapted to this setting by using the rubric as a quality signal rather than a reward model. More generally, investigating the relationship between metric reliability and training efficacy is an important direction for future work.

Finally, we have applied quality-based scoring only to offline methods, but it could also be integrated into RL algorithms. For instance, quality-based scoring could be directly applied to GRPO (Shao et al., 2024). However, RL is notoriously sensitive to misspecification and expensive to run, so careful experimentation would be needed to determine whether the stronger signal from quality-based scoring outweighs the instability in this setting, which we leave as future work.

6. Conclusion

We study post-training on saturated questions, where correctness rewards no longer distinguish among completions. Our results show that quality differences among correct solutions can still provide a useful training signal and transfer to harder questions, but only when the quality scorer is reliable. This makes scorer quality the main bottleneck for learning from saturated data.

Impact Statement

This work studies how to improve language models using quality signals from questions they already answer correctly. Such methods could make existing datasets more useful and reduce the need for additional annotation. The main risks are that unreliable self-judgments may reinforce model biases, while entropy-based objectives may reduce solution diversity or increase overconfidence. These risks make scorer calibration and held-out validation important when applying quality-based training.

Acknowledgements

HH used compute from the Swiss AI Initiative supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID a155 on Alps.

References

Agarwal, S., Zhang, Z., Yuan, L., Han, J., and Peng, H. The Unreasonable Effectiveness of Entropy Minimization in LLM Reasoning, 2025. URL http://arxiv.org/abs/2505.15134.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners, 2020. URL http://arxiv.org/abs/2005.14165.

Burns, C., Izmailov, P., Kirchner, J. H., Baker, B., Gao, L., Aschenbrenner, L., Chen, Y., Ecoffet, A., Joglekar, M., Leike, J., Sutskever, I., and Wu, J. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision, 2023. URL https://arxiv.org/abs/2312.09390.

Cao, S., Valiant, G., and Liang, P. On the entropy calibration of language models, 2026. URL https://arxiv.org/abs/2511.11966.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating Large Language Models Trained on Code, 2021. URL http://arxiv.org/abs/2107.03374.

Choi, K., Saha, D., Kim, W., Agarwal, A., and Dwivedi, R. GOPO: Policy Optimization using Ranked Rewards, 2026. URL http://arxiv.org/abs/2602.03876.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training Verifiers to Solve Math Word Problems, 2021. URL http://arxiv.org/abs/2110.14168.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, 2025. URL http://arxiv.org/abs/2501.12948.

Dong, H., Xiong, W., Goyal, D., Zhang, Y., Chow, W., Pan, R., Diao, S., Zhang, J., Shum, K., and Zhang, T. RAFT: Reward rAnked FineTuning for Generative Foundation Model Alignment, 2023a. URL http://arxiv.org/abs/2304.06767.

Dong, H., Xiong, W., Goyal, D., Zhang, Y., Chow, W., Pan, R., Diao, S., Zhang, J., Shum, K., and Zhang, T. RAFT: Reward rAnked FineTuning for Generative Foundation Model Alignment, 2023b. URL http://arxiv.org/abs/2304.06767.

Fomicheva, M., Sun, S., Yankovskaya, L., Blain, F., Guzmán, F., Fishel, M., Aletras, N., Chaudhary, V., and Specia, L. Unsupervised Quality Estimation for Neural Machine Translation. 8:539–555, 2020. ISSN 2307-387X. doi: 10.1162/tacl_a_00330. URL https://direct.mit.edu/tacl/article/96475.

Garg, S., Singh, A., Singh, S., and Chopra, P. IPO: Your Language Model is Secretly a Preference Classifier, 2025. URL http://arxiv.org/abs/2502.16182.

Geng, S., Ivison, H., Li, C.-L., Sap, M., Li, J., Krishna, R., and Koh, P. W. The Delta Learning Hypothesis: Preference Tuning on Weak Data can Yield Strong Gains, 2025. URL http://arxiv.org/abs/2507.06187.

Gunjal, A., Wang, A., Lau, E., Nath, V., He, Y., Liu, B., and Hendryx, S. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains, 2025. URL http://arxiv.org/abs/2507.17746.

Hase, P., Bansal, M., Clark, P., and Wiegreffe, S. The Unreasonable Effectiveness of Easy Training Data for Hard Tasks, 2024. URL http://arxiv.org/abs/2401.06751.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-Rank Adaptation of Large Language Models, 2021. URL http://arxiv.org/abs/2106.09685.

Huang, Z., Zhuang, Y., Lu, G., Qin, Z., Xu, H., Zhao, T., Peng, R., Hu, J., Shen, Z., Hu, X., Gu, X., Tu, P., Liu, J., Chen, W., Fu, Y., Fan, Z., Gu, Y., Wang, Y., Yang, Z., Li, J., and Zhao, J. Reinforcement Learning with Rubric Anchors, 2025. URL http://arxiv.org/abs/2508.12790.

Ivison, H., Wang, Y., Liu, J., Wu, Z., Pyatkin, V., Lambert, N., Smith, N. A., Choi, Y., and Hajishirzi, H. Unpacking DPO and PPO: Disentangling Best Practices for Learning from Preference Feedback, 2024. URL http://arxiv.org/abs/2406.09279.

Kuhn, L., Gal, Y., and Farquhar, S. Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation, 2023. URL https://arxiv.org/abs/2302.09664v3.

Kydlíček, H. Math-Verify: Math Verification Library, 2026. URL https://github.com/huggingface/math-verify.

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, S., Gu, Y., Malik, S., Graf, V., Hwang, J. D., Yang, J., Bras, R. L., Tafjord, O., Wilhelm, C., Soldaini, L., Smith, N. A., Wang, Y., Dasigi, P., and Hajishirzi, H. Tulu 3: Pushing Frontiers in Open Language Model Post-Training, 2025. URL http://arxiv.org/abs/2411.15124.

Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., and Schulman, J. WebGPT: Browser-assisted question-answering with human feedback, 2022. URL http://arxiv.org/abs/2112.09332.

Niu, W., He, W., Xie, Z., Ou, J., Fan, H., Ge, Y., Sun, Y., Wang, Z., Sun, Y., Shi, C., Gao, J., Hao, J., and He, R. Ranking is Reward: Intra-Group Preference Ranking for Group Relative Policy Optimization. 2025. URL https://openreview.net/forum?id=9664No4ulo.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. Direct Preference Optimization: Your Language Model is Secretly a Reward Model, 2024. URL http://arxiv.org/abs/2305.18290.

Raschka, S. Practical Tips for Finetuning LLMs Using LoRA (Low-Rank Adaptation), 2025. URL https://magazine.sebastianraschka.com/p/practical-tips-for-finetuning-llms.

Shafayat, S., Tajwar, F., Salakhutdinov, R., Schneider, J., and Zanette, A. Can Large Reasoning Models Self-Train?, 2025. URL http://arxiv.org/abs/2505.21444.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL http://arxiv.org/abs/2402.03300.

Song, F., Yu, B., Li, M., Yu, H., Huang, F., Li, Y., and Wang, H. Preference Ranking Optimization for Human Alignment, 2024. URL http://arxiv.org/abs/2306.17492.

Stojanovski, Z., Stanley, O., Sharratt, J., Jones, R., Adefioye, A., Kaddour, J., and Köpf, A. REASONING GYM: Reasoning Environments for Reinforcement Learning with Verifiable Rewards, 2025. URL http://arxiv.org/abs/2505.24760.

Sun, Z., Yu, L., Shen, Y., Liu, W., Yang, Y., Welleck, S., and Gan, C. Easy-to-hard generalization: Scalable alignment beyond human supervision, 2024. URL https://arxiv.org/abs/2403.09472.

Wang, Y., Yang, Q., Zeng, Z., Ren, L., Liu, L., Peng, B., Cheng, H., He, X., Wang, K., Gao, J., Chen, W., Wang, S., Du, S. S., and Shen, Y. Reinforcement Learning for Reasoning in Large Language Models with One Training Example, 2025. URL http://arxiv.org/abs/2504.20571.

Wu, T., Yuan, W., Golovneva, O., Xu, J., Tian, Y., Jiao, J., Weston, J. E., and Sukhbaatar, S. Meta-Rewarding Language Models: Self-Improving Alignment with LLM-as-a-Meta-Judge. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 11537–11554. Association for Computational Linguistics, 2025. ISBN 979-8-89176-332-6.

doi: 10.18653/v1/2025.emnlp-main.583. URL https://aclanthology.org/2025.emnlp-main.583/.

Xiao, Y. and Wang, W. Y. On Hallucination and Predictive Uncertainty in Conditional Language Generation, 2021. URL https://arxiv.org/abs/2103.15025v1.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 Technical Report, 2025a. URL http://arxiv.org/abs/2505.09388.

Yang, Z., Shen, W., Li, C., Chen, R., Wan, F., Yan, M., Quan, X., and Huang, F. SPELL: Self-Play Reinforcement Learning for evolving Long-Context Language Models, 2025b. URL http://arxiv.org/abs/2509.23863.

Yu, Q., Zhang, Z., Zhu, R., Yuan, Y., Zuo, X., Yue, Y., Dai, W., Fan, T., Liu, G., Liu, L., Liu, X., Lin, H., Lin, Z., Ma, B., Sheng, G., Tong, Y., Zhang, C., Zhang, M., Zhang, W., Zhu, H., Zhu, J., Chen, J., Chen, J., Wang, C., Yu, H., Song, Y., Wei, X., Zhou, H., Liu, J., Ma, W.-Y., Zhang, Y.-Q., Yan, L., Qiao, M., Wu, Y., and Wang, M. DAPO: An Open-Source LLM Reinforcement Learning System at Scale, 2025. URL http://arxiv.org/abs/2503.14476.

Yuan, Z., Yuan, H., Tan, C., Wang, W., Huang, S., and Huang, F. RRHF: Rank Responses to Align Language Models with Human Feedback without tears, 2023. URL http://arxiv.org/abs/2304.05302.

Zheng, C., Zhu, J., Ou, Z., Chen, Y., Zhang, K., Shan, R., Zheng, Z., Yang, M., Lin, J., Yu, Y., and Zhang, W. A Survey of Process Reward Models: From Outcome Signals to Process Supervisions for Large Language Models, 2025. URL http://arxiv.org/abs/2510.08049.

A. Task and Data Details

This appendix collects ablation studies, hyperparameter sweeps, and auxiliary analyses that support the main results.

A.1. Difficulty Partitioning: Formal Setup

Problem setting Let questions x be drawn from some distribution over verifiable problems with binary reward. We assume access to a finite collection $\mathcal{D}$ of such questions, as well as a second collection $\mathcal{D}'$ that is strictly simpler than $\mathcal{D}$ under some complexity measure c (e.g. number of reasoning steps or operand magnitude). As intuition, consider $\mathcal{D}$ as research-level competition problems and $\mathcal{D}'$ as high-school-level ones; our concrete instantiation operates at a smaller scale, but the underlying principle is the same. We estimate the model's per-question solve rate by the empirical pass@1 over n samples:

$$\hat{p}(x) = \frac{1}{n}\sum_{i=1}^{n} r_i.$$

Motivation Standard preference-based post-training requires both correct and incorrect responses per question to form preference pairs. Hard questions offer the most room for improvement but yield few correct completions, and those that are correct may not reflect sound reasoning. Saturated questions offer the opposite trade-off: dense, high-quality correct completions but no obvious room for direct improvement. We therefore partition the training set by empirical solve rate, separating easy from hard questions so that any observed transfer from easy-question training to hard-question performance can be cleanly attributed.

Partitioning criteria We partition $\mathcal{D}$ into two subsets based on the empirical solve rate $\hat{p}(x)$, and define a third subset from $\mathcal{D}'$:

$$\begin{aligned}
\mathcal{D}_{\text{saturated}} &= \left\{x \in \mathcal{D} : \hat{p}(x) = 1\right\} \\
\mathcal{D}_{\text{hard}} &= \left\{x \in \mathcal{D} : \hat{p}(x) \leq \tfrac{1}{4}\right\} \\
\mathcal{D}'_{\text{strict}} &= \left\{x \in \mathcal{D}' : \hat{p}(x) = 1\right\}
\end{aligned}$$

Questions between the two thresholds ($\frac{1}{4} < \hat{p}(x) < 1$) are excluded from training splits so that observed performance changes are less likely to be driven by mixed difficulty levels in the training data. Comparing transfer from $\mathcal{D}'_{\text{strict}}$ versus $\mathcal{D}_{\text{saturated}}$ to hard-question performance probes the effect of the difficulty gap between training and evaluation data.

A.2. Breaking Down Task Difficulty

The empirical solve rates used to define the difficulty partitions are shown in Table 4: performance decreases monotonically as the number of terms and digits increases, giving a controlled difficulty axis for the `chain sum` task.

Table 4. Pass@1 (estimated from $k = 8$ generations) for `Qwen3-1.7B-Base` across chain-sum difficulty configurations. Cell color encodes pass@1 (darker = higher).

terms / digits	3	4	5	6
3	93%	91%	87%	82%
4	92%	87%	83%	79%
5	86%	81%	75%	68%
6	76%	70%	63%	56%

(a) $\mathcal{D}'_{\text{strict}}$ (3–6 terms & digits)

terms / digits	6	7	8	9	10
6	56%	50%	43%	32%	27%
7	47%	41%	34%	25%	18%
8	40%	33%	27%	19%	14%
9	33%	27%	20%	14%	9%
10	28%	23%	16%	11%	7%

(b) $\mathcal{D}_{\text{saturated}}$ and $\mathcal{D}_{\text{hard}}$ (6–10 terms & digits)

We further investigate whether arithmetic properties beyond the number of terms and digits explain difficulty variation within a single difficulty cell.

Within a fixed difficulty cell (same number of terms and digits), one might expect finer-grained arithmetic properties to further explain variance in pass@1. We examine three such features:

(1) *proportion of additions*: the number of addition operators out of all operators in the expression, since subtractions can drive intermediate values negative and may be harder to track;
(2) *mean operand magnitude*: the average absolute value of the operands, as larger numbers require more carry operations; and
(3) *maximum intermediate value*: the largest absolute value reached during left-to-right evaluation, which determines how large the numbers a model must reason about can grow mid-computation.
Table 5 shows that none of these features meaningfully stratifies difficulty within the 6-term/6-digit subgroup.

Table 5. pass@1 breakdown within the 6-term/6-digit subgroup ($n = 2{,}955$) by three structural features. The pass@1 range across all bins is only 0.55–0.63, indicating that difficulty is not explained by any of these arithmetic properties.

# additions (out of 5 ops)			Mean operand magnitude				Max intermediate value			
Bin	n	pass@1	Bin	n	pass@1	Mean	Bin	n	pass@1	Mean
0	104	0.617	Q1	591	0.585	401,274	Q1	591	0.561	786,505
1	473	0.557	Q2	591	0.557	493,138	Q2	591	0.575	1,135,996
2	904	0.549	Q3	591	0.555	550,571	Q3	591	0.566	1,480,625
3	892	0.556	Q4	591	0.550	608,392	Q4	591	0.559	1,898,710
4	481	0.582	Q5	591	0.568	700,397	Q5	591	0.554	2,790,242
5	101	0.627								

A.3. Task Example

Example: `chain sum` **(4 terms, 3 digits)**

Prompt: *Compute the following step by step:* $381 - 706 - 602 + 383 =$ *Show your reasoning and return the final answer in* `\boxed{}` *tags, for example* `\boxed{42}`.

Completion A: To solve this problem, we need to follow the order of operations (PEMDAS/BODMAS) which means we do the operations from left to right since there are no parentheses, exponents, multiplication/division, or addition/subtraction operations to perform.

Let's start with the first two numbers:
381 - 706 = -325

Now, we subtract the third number from the result:
-325 - 602 = -927

Finally, we add the last number:
-927 + 383 = -544

So, the final answer is:
$\boxed{-544}$

Completion B: To compute the expression 381 - 706 - 602 + 383, we can follow these steps:

1. Subtract 706 from 381:
381 - 706 = -325

2. Subtract 602 from -325:
-325 - 602 = -927

3. Add 383 to -927:
-927 + 383 = -544

So, the final answer is $\boxed{-544}$.

Both responses arrive at the correct answer but differ in verbosity and reasoning style.

Table 6. Evaluation configuration.

Parameter	Value
Evaluation dataset	`hannoh/chainsum_eval` $GSM8K$ (test split)
Number of questions	200 (`chain sum`) 1,319 (GSM8K)
Samples per question (n)	8
Temperature	0.6
Top-p	0.95
Max generation tokens	2048

Table 7. LLM-as-a-judge configuration.

Parameter	Value
Judge model	J_θ: `Qwen/Qwen3-1.7B-Base` J_ϕ: `Qwen/Qwen3-30B`
Tournament group size	2 (pairwise)
Temperature	0 (greedy)
Output constraint	Regex: ends with `Judgment: [N]`
Max generation tokens	1024

B. Hyperparameters

We list all training, optimization, and method-specific hyperparameters used throughout our experiments. Our comparison is *data-constrained*: every method trains on the same data pool, so differences in downstream performance reflect how effectively each objective extracts signal from a fixed dataset rather than differences in data quantity or coverage. We train each method with LoRA adapters using AdamW and a cosine learning-rate schedule. For each configuration, we train for 1–4 epochs and select the checkpoint with the highest pass@1 on a held-out validation set. Default hyperparameters follow established conventions for each method (Hu et al., 2021; Rafailov et al., 2024; Yuan et al., 2023); we ablate key choices (λ for RRHF, pair selection for DPO) in Sec. C.1 and C.2. Hyperparameters and method-specific additions are listed in Table 8; values shown as sets indicate the values swept, and we report results for the best-performing configuration per method.

B.1. Evaluation Configuration

Table 6 summarizes the evaluation setup used for all reported results. All reported pass@1 and pass@8 values are given in %, estimated using the unbiased estimator of Chen et al. (2021). With $n = 8$ samples per problem, pass@8 reduces to a binary metric: whether at least one sample solves the problem. For both `chain sum` and GSM8K, we check for answer equivalence using Math-Verify (Kydlíček, 2026).

B.2. Judge Configuration

Table 7 lists the configuration of the LLM-as-a-judge used to score completions during training data collection.

Few-shot prompting for judging For base models, we employ few-shot prompting with worked examples to reliably parse judge outputs. For instruction-tuned models, we use standard zero-shot prompting with evaluation criteria. Prompts are adapted per task (e.g., `chain sum` vs. GSM8K); full prompts are in Sec. D.

We use pairwise comparisons (group size 2) for the judge. Larger group sizes would allow ranking more completions in a single pass, but are likely to degrade judgment quality when using a small self-judge model (especially for base models) and additionally require proportionally larger context windows to accommodate all completions simultaneously.

The output is constrained by a regex that requires the response to end with `Judgment: [N]`. Because we use a base model as the judge rather than an instruction-tuned model, it does not reliably follow formatting instructions from the prompt alone. The regex constraint ensures that every judge response produces a parseable selection, avoiding the need to discard

Table 8. Training hyperparameters. Sets indicate the values swept; we report results for the best-performing configuration per method.

Symbol	Value	Description
	Training	
M_{base}	`Qwen3-1.7B-Base`	Base model.
E	$\{1, 2, 4\}$	Training epochs.
B	$\{2, 4, 8, 16\}$	Batch size.
L	2048	Max sequence length.
	Optimization (AdamW)	
η	$\{5 \times 10^{-6}, 10^{-5}, 5 \times 10^{-5}\}$	Learning rate.
Scheduler	Cosine	Cosine LR decay.
Warmup	0.1	Linear warmup.
w_d	0.01	Weight decay (except for DPO).
	LoRA	
r	32	LoRA rank.
α	64	LoRA alpha.
p_{drop}	0.05	LoRA dropout.
	Scoring	
ϵ	10^{-8}	Floor for $\bar{H}$ in inverse-entropy scoring (Sec. 3.2).
	DPO	
β	$\{0.1, 0.3, 0.5, 2, 5\}$	KL penalty coefficient.
w_d (DPO)	0.0	No weight decay.
Loss	Sigmoid	Loss function type.
	RRHF	
λ	$\{0, 0.01, 0.1, 0.5, 1.0\}$	Rank loss weight.
σ	Logistic weighting	Scales each hinge-loss pair by $\sigma(r_j - r_i)$, the logistic sigmoid of the score gap (Sec. 3.4).

malformed outputs.

B.3. Training Hyperparameters

Table 8 lists the hyperparameters shared across all training runs, along with method-specific additions.

We train for 1–4 epochs and select the best checkpoint by pass@1 on the held-out validation set. We set the LoRA alpha to twice the rank ($\alpha = 2r$), following the practical recommendation of Raschka (2025). For DPO we disable weight decay, following Ivison et al. (2024). The DPO β and RRHF rank loss weight λ were selected by hyperparameter sweeps described in Sec. C.2 and C.5, respectively. Effective batch sizes are achieved via gradient accumulation across per-device batches; how the batch is divided between accumulation steps and per-device batch size did not affect results, except for the ranking loss in σ-RRHF where the set of completions compared within each step depends on the per-device batch size, which we ensure is always at least n so that all completions for a question appear together in the same batch.

Table 9. **Method comparison** on $\mathcal{D}_{\text{hard}}$ chain sum problems. + DPO (correctness) uses correct-vs-incorrect pairs from $\mathcal{D}_{\text{hard}}$ (Sec. 3.3). **Bold** is best per column.

Model / Training	Scoring	pass@1	pass@8
`Qwen3-1.7B-Base`		10.63	51.50
+ SFT		30.19	73.00
+ σ-RRHF	Self-judge	**39.75**	**77.50**
+ DPO (correctness)	Random	29.56	60.50
+ DPO (correctness)	Self-judge	34.94	62.00

C. Additional Experiments

C.1. Validating Methods on Hard Questions

We first trained on $\mathcal{D}_{\text{hard}}$ (Sec. A.1), where the model fails to solve most instances and binary correctness signal is abundant. We compare Rejection Sampling, DPO, and σ-RRHF. Rejection Sampling with one iteration serves as the primary baseline, which simplifies to SFT on correct completions only; σ-RRHF uses λ=0.1.

DPO with contrastive pairs When training DPO on $\mathcal{D}_{\text{hard}}$, we filter out unsolvable training questions (pass@8 $= 0$) so that each question has at least one correct and one incorrect completion. Evaluation remains on the same held-out chain-sum evaluation set used in the saturated experiments. We consider two pair-construction strategies. **Random**: y_w is sampled uniformly from the correct completions and y_l from the incorrect ones, ignoring quality scores. **Self-judge**: $y_w = \arg\max_{i:\, r_i=1} r_i^{J_M}$ and $y_l = \arg\min_{i:\, r_i=0} r_i^{J_M}$, using the policy model's own win-rate scores (Eq. (2)). Unless otherwise noted, DPO on $\mathcal{D}_{\text{hard}}$ uses the self-judge variant.

Table 9 reveals a tension between per-sample accuracy and coverage. DPO achieves high pass@1 ($+4.8\%$ over SFT) but trails substantially on pass@8 (-11%). SFT, which trains on all correct completions equally, preserves coverage but forgoes any quality distinction among them. σ-RRHF mitigates this trade-off: its SFT anchor preserves pass@8 ($+4.5\%$ over SFT) while its quality-weighted ranking provides the largest pass@1 gain ($+9.6\%$ over SFT).

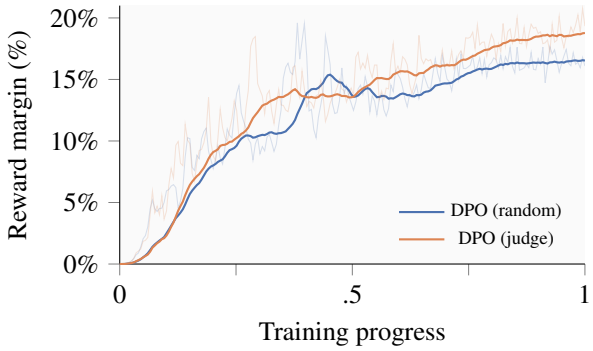
Random vs. self-judge pair selection To isolate the effect of completion ranking, we compare the two pair-construction strategies above with all other variables fixed. To understand training dynamics, we additionally track the *reward margin* (Figure 2), defined as the log-probability difference between the chosen and rejected completion under the current policy.

Self-judge pairs outperform random pairs on both metrics: pass@1 improves by $+5.4\%$ and pass@8 by $+1.5\%$ (Table 9). Reward margins grow throughout training for both strategies (Figure 2a), yet their downstream trajectories diverge: self-judge pairs continue improving pass@1 until 75% of training, whereas random pairs plateau after 50% (Figure 2b). The continued reward-margin growth for random pairs without corresponding evaluation gains suggests overfitting to the training signal. Self-judge pairs, by contrast, translate margin growth into downstream improvement for longer, suggesting that preference-signal quality strongly affects training efficacy. The pattern suggests that, on this split, the judge identifies more useful responses than random selection among completions with the same correctness level.

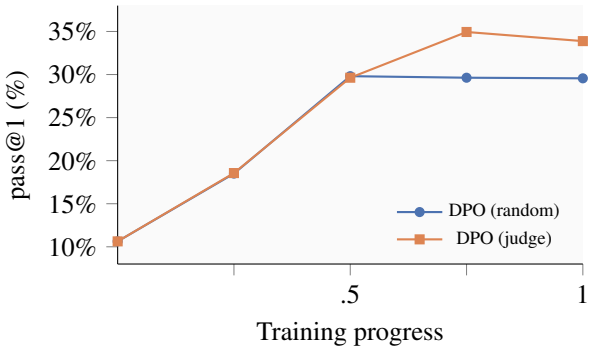
C.2. Rank Loss Weight Ablation

We ablate the rank loss weight λ in RRHF on $\mathcal{D}_{\text{hard}}$, the hardest difficulty split, where the model has the most room for improvement and the ranking signal should matter most. We sweep $\lambda \in \{0, 10^{-2}, 10^{-1}, 0.5, 1.0\}$. The baseline $\lambda = 0$ corresponds to SFT on the judge-selected argmax completion only (no rank loss), which recovers RAFT (Dong et al., 2023a). We report the mean of three runs for $\lambda = 0$.

The rank loss provides at most marginal improvement: pass@1 peaks at $\lambda = 10^{-2}$ ($+1.1\%$ over no rank loss), while pass@8 peaks at $\lambda = 10^{-1}$ ($+1.5\%$), suggesting a mild diversity benefit at moderate weights. In both cases $\lambda \geq 0.5$ actively hurts performance. This suggests that, in this setting, RRHF's performance is driven primarily by the SFT-on-argmax component (selecting and imitating the best completion via the judge) rather than the ranking objective itself. Because all completions in this ablation are ranked by the self-judge, whose agreement with the strong judge is weak (Sec. C.7), it remains an open question whether a higher-quality ranker would unlock a larger benefit from the rank loss term.

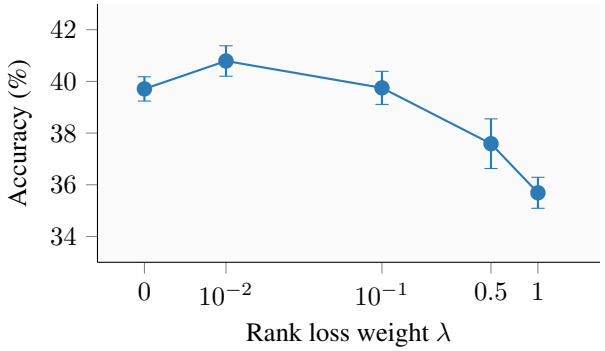


(a) Implicit reward margin $\hat{r}(x, y_w) - \hat{r}(x, y_l)$ over training.

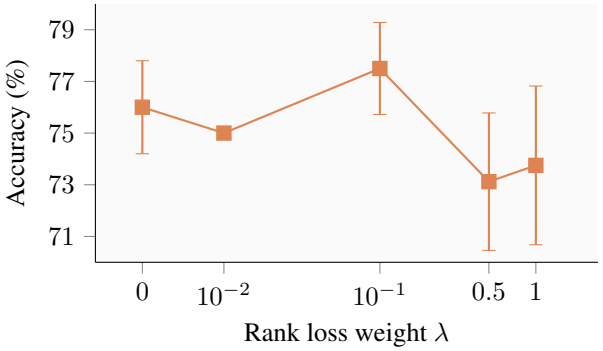


(b) pass@1 at four evaluation checkpoints.

Figure 2. DPO training dynamics on $\mathcal{D}_{\text{hard}}$ for random vs. judge-selected pair construction.



(a) pass@1



(b) pass@8

Figure 3. Rank loss weight ablation on $\mathcal{D}_{\text{hard}}$ chain sum problems. The x-axis is log-scaled; $\lambda = 0$ (no rank loss) is placed at the left as a reference point. Pass@1 peaks at $\lambda = 10^{-2}$ and degrades monotonically for larger weights. Pass@8 peaks at $\lambda = 10^{-1}$ with no clear monotonic trend, but some rank loss has an effect on generalization to OOD questions. Responses were ranked with the Self-judge.

C.3. σ-RRHF Component Ablation

To understand which components of σ-RRHF contribute to its performance, we ablate three design choices on $\mathcal{D}_{\text{hard}}$: the SFT term, the hinge in the rank loss, and the logistic pair reweighting.

No SFT loss This ablation removes the SFT component entirely, training only with the rank loss. The total loss reduces to:

$$\mathcal{L} = \mathcal{L}_{\text{rank}}.$$

Without the SFT anchor on the best completion, the model receives only relative preference signal and has no direct imitation target.

Table 10. σ-RRHF ablation on $\mathcal{D}_{\text{hard}}$ chain sum. The SFT anchoring term is essential: without it performance collapses to near-random. Hinge loss and logistic reweighting provide no consistent improvement.

Variant	pass@1	pass@8
w/o SFT term	6.69	28.00
w/o Hinge	36.56	74.00
w/o logistic weighting	39.62	77.00

Table 11. σ-RRHF λ-sweep on the saturated splits with inverse entropy, pass@1. $\lambda = 0$ omits the rank loss (SFT on the inverse-entropy-best completion).

	$\mathcal{D}_{\text{saturated}}$		$\mathcal{D}'_{\text{strict}}$	
	pass@1	pass@8	pass@1	pass@8
$\lambda = 0$	27.37	58.30	21.29	59.20
$\lambda = 0.01$	28.00	57.00	22.50	58.20
$\lambda = 0.1$	29.25	61.75	22.35	59.25
$\lambda = 0.5$	28.21	60.50	22.46	61.30
$\lambda = 1.0$	26.87	58.20	21.15	58.80

No hinge Instead of only penalizing incorrectly ranked pairs, we also reward correctly ranked ones by removing the $\max(0, \cdot)$ clamp. For every pair where $r_i < r_j$ (i.e. y_j is the better completion), the loss becomes:

$$\mathcal{L}_{\text{rank}} = \sum_{r_i < r_j} \frac{1}{1 + \exp(-(r_j - r_i))} (p_i - p_j).$$

This encourages the model to keep increasing the log-probability gap even for already correctly ranked pairs, which may lead to overconfident predictions.

No logistic weighting This ablation removes the logistic reweighting, treating all pairs equally regardless of the reward gap. For every pair where $r_i < r_j$, the loss is:

$$\mathcal{L}_{\text{rank}} = \sum_{r_i < r_j} \max(0,\, p_i - p_j).$$

Results We discuss the ablation results in Table 10. Removing the SFT term is catastrophic: pass@1 drops from 39.75% to 6.69% and pass@8 from 77.50% to 28.00%, showing that the rank loss alone is insufficient to guide learning in this setting. Removing the hinge costs roughly 3% on both metrics (36.56% pass@1, 74.00% pass@8), suggesting that continuing to push already correct rankings apart slightly hurts. Removing the logistic reweighting has negligible effect (39.62% pass@1, 77.00% pass@8), indicating that weighting pairs by reward gap adds little when the self-judge scores are noisy. Together with the rank weight ablation (Sec. C.2), these $\mathcal{D}_{\text{hard}}$ experiments suggest that RRHF's performance is driven mostly by the SFT-on-argmax component.

C.4. Ablations on the Saturated Splits

We repeat the ablations on the saturated splits $\mathcal{D}_{\text{saturated}}$ and $\mathcal{D}'_{\text{strict}}$ with inverse entropy, using the headline σ-RRHF configuration. We report pass@1; the $\lambda = 0.1$ row reproduces the inverse-entropy σ-RRHF entry of Table 1.

Rank loss weight Table 11 sweeps the rank loss weight λ, where $\lambda = 0$ omits the rank loss and trains only on the inverse-entropy-best completion. The rank loss gives a small gain at small λ (up to $+1.9$ and $+1.2$ pass@1 over $\lambda = 0$) and reverses by $\lambda = 1$.

Components Table 12 removes the rank-loss components at $\lambda = 0.1$. Dropping the logistic weighting lowers both metrics; removing the hinge makes training diverge (zero accuracy in all runs). Both differ from $\mathcal{D}_{\text{hard}}$, where the logistic weighting was negligible and the hinge cost only a few points.

C.5. DPO Hyperparameter Sensitivity

On both $\mathcal{D}_{\text{saturated}}$ and $\mathcal{D}'_{\text{strict}}$, self-judge DPO underperforms SFT on pass@1. A plausible explanation is that DPO's contrastive objective is more sensitive to scoring noise than SFT: when the self-judge cannot meaningfully separate chosen and rejected completions, training on potentially near-arbitrary preferences becomes counterproductive. We sweep the KL penalty β in DPO to find stable training configurations.

Table 12. σ-RRHF component ablation at $\lambda = 0.1$ on the saturated splits, pass@1.

	$\mathcal{D}_{\text{saturated}}$		$\mathcal{D}'_{\text{strict}}$	
	pass@1	pass@8	pass@1	pass@8
full σ-RRHF	29.25	61.75	22.35	59.25
w/o logistic weighting	27.02	59.70	21.33	57.30
w/o Hinge	diverged			

Table 13. DPO hyperparameter sweep on the $\mathcal{D}'_{\text{strict}}$ split (200 questions, 8 samples per question), using self-judge pair selection. **Bold** indicates the best configuration by pass@1.

β	η	Epochs	pass@1	pass@8
0.1	5e-6	4	5.06	26.00
0.3	5e-6	4	6.50	31.00
0.5	5e-6	4	8.75	38.00
2	5e-6	4	12.81	52.00
5	**5e-6**	**4**	**13.56**	**53.00**

We sweep $\beta \in \{0.1, 0.3, 0.5, 2, 5\}$ on the $\mathcal{D}'_{\text{strict}}$ split with self-judge pair selection. The standard value $\beta = 0.1$ suggested by Rafailov et al. (2024) yields only 5.06% pass@1, while stronger regularization substantially improves performance, with the best configuration ($\beta = 5$, $\eta = 5 \times 10^{-6}$, 4 epochs) reaching 13.56% pass@1 — still below SFT. This need for much stronger KL regularization is consistent with observations by Geng et al. (2025); Lambert et al. (2025) and likely reflects noisy self-judge preference pairs.

C.6. GSM8K DPO Training Dynamics

We show the implicit reward margin $\hat{r}(x, y_w) - \hat{r}(x, y_l)$ over training on GSM8K for the DPO variants from Sec. 4.3. Inverse entropy reaches the highest margin, the `Qwen3-30B` strong judge grows more steadily, and the Qwen3-1.7B-Base self-judge stays near zero — consistent with its near-zero Spearman correlation with the strong judge.

C.7. Scorer Agreement and Entropy Correlations

We assess how well the three scoring methods agree on completion rankings: strong judge J_ϕ (`Qwen3-30B`), self-judge J_θ (Qwen3-1.7B-Base), and mean token-level entropy $\bar{H}$. For each question we rank its n completions independently under each scorer and compute the within-question Spearman ρ between every pair. Table 14 reports the mean and median ρ across questions; Figure 5 shows the full distributions. The `chain sum` correlations are computed on $\mathcal{D}'_{\text{strict}}$, where all questions are strictly easier by construction (terms and digits ranging from 3 to 6, excluding the 6-term/6-digit cell); the GSM8K correlations use $\mathcal{D}_{\text{saturated}}$.

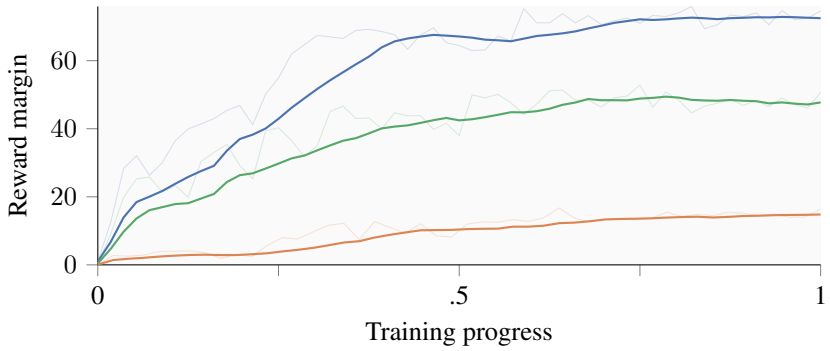


Figure 4. Implicit reward margin $\hat{r}(x, y_w) - \hat{r}(x, y_l)$ over training on GSM8K. Inverse entropy reaches the highest margin; Qwen3-30B grows more steadily; Qwen3-1.7B-Base (self-judge) stays near zero. Faint lines are raw per-step values; bold lines are rolling averages.

Table 14. Within-question Spearman rank correlations between scoring methods ($r_i^{J_\phi}$: strong judge, $r_i^{J_\theta}$: self-judge, r_i^H: inverse entropy) on $\mathcal{D}'_{\text{strict}}$ `chain sum` and $\mathcal{D}_{\text{saturated}}$ GSM8K. Each cell reports mean (median) ρ and the percentage of questions with $\rho > 0$.

	`chain sum` ($\mathcal{D}'_{\text{strict}}$)		GSM8K ($\mathcal{D}_{\text{saturated}}$)	
Pair	$\bar{\rho}$ ($\tilde{\rho}$)	% pos.	$\bar{\rho}$ ($\tilde{\rho}$)	% pos.
$r_i^{J_\phi}$ vs. $r_i^{J_\theta}$ (`Qwen3-30B` vs. `Qwen3-1.7B-Base`)	$+0.15$ ($+0.19$)	62.0	-0.20 (-0.24)	27.9
$r_i^{J_\phi}$ vs. r_i^H (`Qwen3-30B` vs. Inverse entropy)	$+0.44$ ($+0.50$)	89.4	$+0.56$ ($+0.63$)	94.0
$r_i^{J_\theta}$ vs. r_i^H (`Qwen3-1.7B-Base` vs. Inverse entropy)	$+0.16$ ($+0.19$)	68.5	-0.20 (-0.22)	30.9

Table 15. Completion length (median tokens) by inverse-entropy rank: *selected* is the highest-scoring completion (entropy-argmax, the SFT anchor), *lowest* the lowest-scoring (entropy-argmin). The last column is the within-question Spearman ρ between inverse-entropy score and token length.

Split	selected	lowest	random	ρ(score, len)
ChainSum $\mathcal{D}_{\text{saturated}}$	330	262	299	$+0.41$
ChainSum $\mathcal{D}'_{\text{strict}}$	180	129	153	$+0.39$
GSM8K $\mathcal{D}_{\text{saturated}}$	311	161	234	$+0.72$

The strong judge and inverse entropy correlate moderately on both tasks ($\bar{\rho} = +0.44$ on $\mathcal{D}'_{\text{strict}}$ `chain sum`, $+0.56$ on GSM8K), showing that lower-entropy completions tend to receive higher judge scores. The self-judge agrees only weakly with both the strong judge and inverse entropy on $\mathcal{D}'_{\text{strict}}$ `chain sum` ($\bar{\rho} \approx +0.15$ and $+0.16$), and on GSM8K the self-judge vs. inverse-entropy correlation reverses sign ($\bar{\rho} = -0.20$), consistent with the self-judge failures observed in Sec. 4.3.

C.8. Completion Length under Inverse-Entropy Selection

We test whether inverse entropy selects for shorter completions. For each question we compare the token length of the highest-scoring completion (the completion σ-RRHF anchors its SFT term on), the lowest-scoring completion, and a uniformly random one (Table 15). On all three splits the highest-scoring completion is the longest, and the within-question Spearman correlation between inverse-entropy score and token length is positive. Measured in characters the difference is small on `chain sum` and large on GSM8K. Inverse entropy does not select for shorter completions.

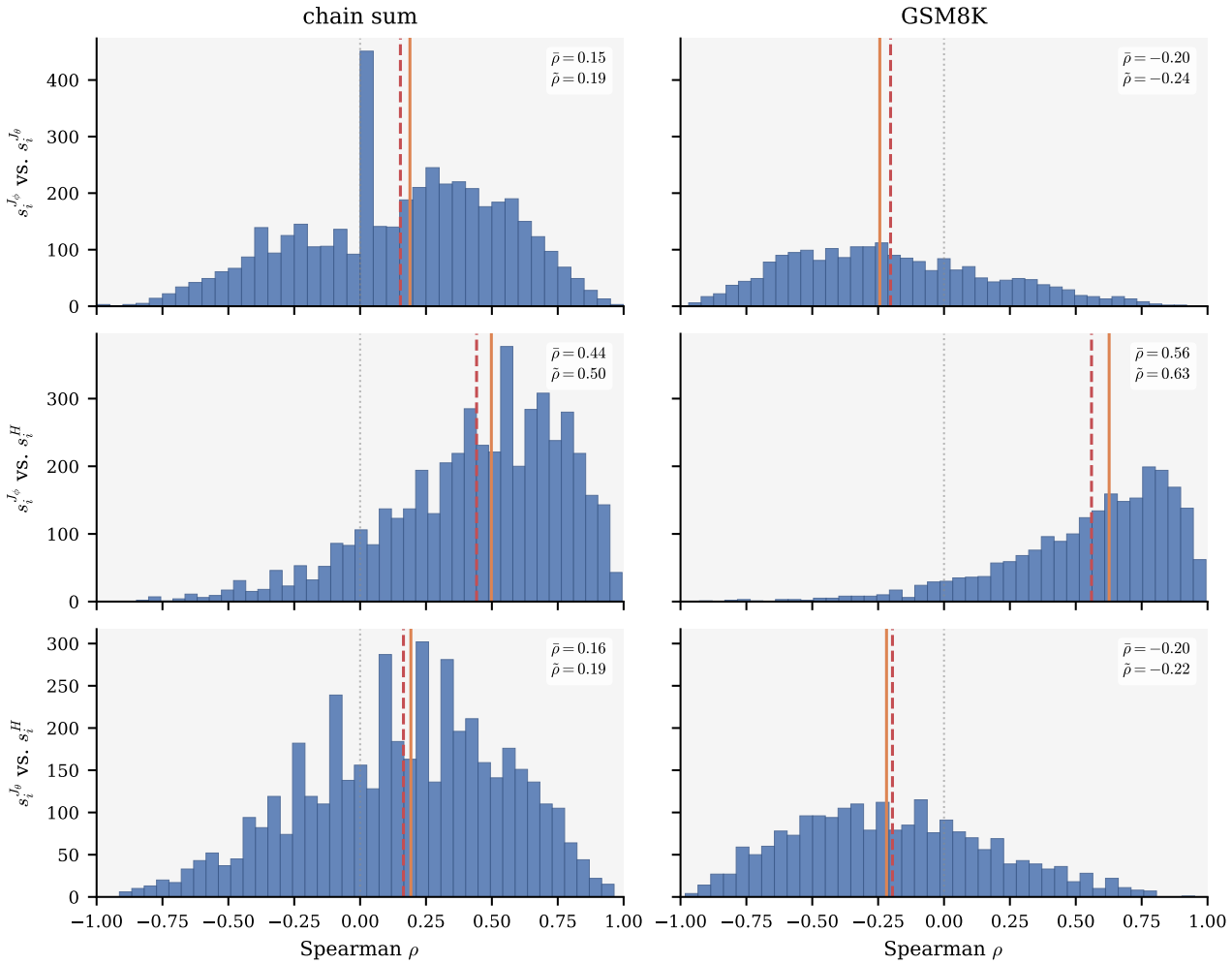


Figure 5. Distribution of within-question Spearman ρ between each pair of scorers, for $\mathcal{D}'_{\text{strict}}$ `chain sum` (left) and $\mathcal{D}_{\text{saturated}}$ GSM8K (right). Row labels indicate the scorer pair. Dashed red line: mean ($\bar{\rho}$); solid orange line: median ($\tilde{\rho}$). The strong judge and inverse entropy show consistent positive correlation (middle row), while the self-judge agrees only weakly with the other two scorers.

D. Prompts

The following prompts are used for LLM-as-a-judge completion ranking (Sec. 3.2). Each task uses two prompt formulations: a *full few-shot* version that embeds a worked example, and a *short system/user* pair without a demonstration. The judge generates free-form analysis constrained by a regex to end with `Judgment: [IDX]`.

D.1. Prompts for `chain sum`

D.1.1. Select Best Correct Completion

Identifies the highest-quality correct completion among multiple correct solutions. The judge evaluates step-by-step correctness, logical progression, notation, and efficiency.

Full few-shot prompt (used when `instruct=False`):

`chain sum` – Select Best Correct Completion (Full Few-Shot)

```
# Mathematical Solution Evaluation

There are math problems followed by a number of solutions. The task is to
systematically analyze these solutions to identify the most mathematically
sound approach.

### Evaluation Process

#### 1. Initial Screening
* Group solutions by their final answers.
* Identify and explain mathematical contradictions between different answers.
* Eliminate solutions with clear mathematical errors.

#### 2. Detailed Analysis
For remaining solutions, evaluate:
* **Mathematical Precision and Accuracy:** Correctness of calculations and
theorem applications.
* **Logical Progression:** The flow and sequence of steps.
* **Completeness:** Whether the mathematical reasoning is fully documented.
* **Notation:** Proper use of mathematical symbols and the `\boxed{}` format.
* **Special Conditions:** Handling of edge cases or specific constraints.
* **Error Correction:** For solutions containing and addressing errors,
evaluate the identification and correction methodology.

#### 3. Solution Comparison
Compare viable solutions based on:
* **Efficiency:** The elegance of the approach.
* **Clarity:** How well the reasoning is communicated.
* **Sophistication:** The depth of the method used.
* **Robustness:** Whether the solution works for all cases.

### Response Requirements
1. **Brief Analysis** of conflicting answers.
2. **Detailed Evaluation** of mathematically sound solutions.
3. **Justification** for eliminating incorrect solutions.
4. **Clear Explanation** for selecting the best approach.

An evaluation ends with:
**Judgment: [IDX]** *(Where `IDX` is the index $0 - {max_idx}$ of the best
solution.)*

---

## EVALUATION A
**Problem:** Compute the following step by step:
```

```
  593615 + 204846 - 838944 + 816336 - 913166 - 338746 =

  **Solutions (all correct, ground truth = -476059):**
0) Let's solve the problem step by step:

1. **Addition:**
\\[
  593615 + 204846 = 798461
\\]

2. **Subtraction:**
\\[
  798461 - 838944 = -40483
\\]

3. **Addition:**
\\[
  -40483 + 816336 = 775853
\\]

4. **Subtraction:**
\\[
  775853 - 913166 = -137313
\\]

5. **Subtraction:**
\\[
  -137313 - 338746 = -476059
\\]

The final answer is: \\boxed{-476059}

1) Let's solve this step by step:

1. Compute 593615 + 204846
2. Subtract 838944 from the result
3. Add 816336 to the new result
4. Subtract 913166 from the new result
5. Subtract 338746 from the new result

Now, let's compute each step:

1. 593615 + 204846 = 798461
2. 798461 - 838944 = -40483
3. -40483 + 816336 = 775853
4. 775853 - 913166 = -137313
5. -137313 - 338746 = -476059

So, the final answer is \\boxed{-476059}.

### EVALUATION
**Initial Screening:** Both solutions arrive at -476059. No answer
disagreement; both are arithmetically correct.

**Analysis:**
- Solution 0: Proceeds directly to computation. Each step is labeled by
operation type (**Addition:**, **Subtraction:**) and formatted in LaTeX
display math. The structure is clean: one numbered item per operation,
intermediate result immediately visible.
- Solution 1: Opens with a planning phase that lists all five steps as
instructions before computing anything. The plan is redundant given the
computations that follow. The inline `number = result` style lacks the
visual clarity of display math.
```

```
**Comparison:** Both are correct and complete. Solution 0 is more direct --
it computes each step immediately without a preamble. The operation-type
labels and LaTeX formatting make intermediate values easy to scan and verify.
Solution 1's two-phase structure (plan then execute) adds length without
adding information.

**Judgment: [0]**

---

## EVALUATION B
**Problem:** {problem}

**Solutions:** {solutions}

### EVALUATION
```

D.2. GSM8K Prompts

D.2.1. Select Best Correct Completion

Same role as the `chain_sum` variant but adapted for grade-school math word problems: the evaluation criteria add problem comprehension, unit-conversion accuracy, and logical flow alongside the structural quality criteria.

Full few-shot prompt (used when `instruct=False`):

GSM8K – Select Best Correct Completion (Full Few-Shot)

```
# Word Problem Solution Evaluation

There are word problems followed by a number of solutions. The task is to systematically analyze
    these solutions to identify the most mathematically sound approach.

### Evaluation Process

#### 1. Initial Screening
* Group solutions by their final answers.
* Identify and explain contradictions between different answers.
* Eliminate solutions with clear mathematical or logical errors.

#### 2. Detailed Analysis
For remaining solutions, evaluate:
* **Problem Comprehension:** Does the solution correctly identify all given quantities and what is
     being asked?
* **Step-by-step Correctness:** Are all arithmetic operations and unit conversions accurate?
* **Logical Flow:** Does each step follow naturally from the previous one?
* **Completeness:** Are all sub-problems addressed before combining into a final answer?
* **Clarity:** Is the solution easy to follow without unnecessary repetition?

#### 3. Solution Comparison
Compare viable solutions based on:
* **Directness:** Does the solution get to the point without redundant preambles?
* **Organization:** Are the steps well-structured and labeled?
* **Presentation:** Is the final answer clearly stated?

### Response Requirements
1. **Brief Analysis** of any answer disagreements.
2. **Detailed Evaluation** of mathematically sound solutions.
3. **Clear Explanation** for selecting the best approach.
```

```
An evaluation ends with:
**Judgment: [IDX]** *(Where ‘IDX‘ is the index $0 - {max_idx}$ of the best solution.)*

---

## EVALUATION A
**Problem:** For every 12 cans you recycle, you receive $0.50, and for every 5 kilograms of
    newspapers, you receive $1.50. If your family collected 144 cans and 20 kilograms of
    newspapers, how much money would you receive?

**Solutions (all correct, ground truth = 12):**
0) To solve this problem, we need to calculate the amount of money received from recycling the
    cans and the newspapers separately, and then add those amounts together.

1. **Calculate the money received from recycling cans:**

   We know that for every 12 cans, you receive $0.50. Your family collected 144 cans. To find out
       how many sets of 12 cans they have, we divide the total number of cans by 12:
   \\\\[
     \\\\frac{{144 \\\\text{{ cans}}}}{{12 \\\\text{{ cans/set}}}} = 12 \\\\text{{ sets}}
   \\\\]

   Since each set of 12 cans gives $0.50, we multiply the number of sets by $0.50:
   \\\\[
     12 \\\\text{{ sets}} \\\\times \\\\$0.50/\\\\text{{set}} = \\\\$6.00
   \\\\]

2. **Calculate the money received from recycling newspapers:**

   We know that for every 5 kilograms of newspapers, you receive $1.50. Your family collected 20
       kilograms. To find out how many sets of 5 kilograms they have:
   \\\\[
     \\\\frac{{20 \\\\text{{ kg}}}}{{5 \\\\text{{ kg/set}}}} = 4 \\\\text{{ sets}}
   \\\\]

   Since each set of 5 kilograms gives $1.50:
   \\\\[
     4 \\\\text{{ sets}} \\\\times \\\\$1.50/\\\\text{{set}} = \\\\$6.00
   \\\\]

3. **Add the amounts:**
   \\\\[
     \\\\$6.00 + \\\\$6.00 = \\\\$12.00
   \\\\]

Therefore, the total amount of money your family would receive is $\\\\boxed{{12}}$.

1) Let’s solve this step by step:

1. Calculate the money earned from recycling cans:
   - For every 12 cans, you receive $0.50.
   - Your family collected 144 cans.
   - To find out how many sets of 12 cans there are in 144 cans, divide 144 by 12: 144 / 12 = 12
       sets.
   - Multiply the number of sets by the amount earned per set: 12 sets * $0.50/set = $6.00.

2. Calculate the money earned from recycling newspapers:
   - For every 5 kilograms of newspapers, you receive $1.50.
   - Your family collected 20 kilograms of newspapers.
   - To find out how many sets of 5 kilograms there are in 20 kilograms, divide 20 by 5: 20 / 5 = 4
       sets.
   - Multiply the number of sets by the amount earned per set: 4 sets * $1.50/set = $6.00.
```

```
3. Add the money earned from both cans and newspapers:
   - Money earned from cans: $6.00
   - Money earned from newspapers: $6.00
   - Total money earned: $6.00 + $6.00 = $12.00

Final answer: You would receive $12.00.

### EVALUATION
**Initial Screening:** Both solutions arrive at $12.00. No answer disagreement; both are
    arithmetically correct.

**Analysis:**
- Solution 0: Each sub-problem is clearly labeled (**Calculate the money received from recycling
    cans/newspapers**). Uses LaTeX display math for key calculations, making intermediate values
    easy to verify at a glance. The structure is clean with numbered sections and a boxed final
    answer.
- Solution 1: Correct and complete, but uses a flat inline style throughout. The final answer
    lacks \\\\boxed formatting. A separate summary block ("Money earned from cans: $6.00...")
    restates values already computed, adding length without information.

**Comparison:** Both are correct. Solution 0 is better organized -- bold section headers, LaTeX
    for key steps, and a boxed answer make it easier to scan and verify. Solution 1 is slightly
    redundant with its summary block.

**Judgment: [0]**

---

## EVALUATION B
**Problem:** A baker makes 48 cookies. He keeps one-quarter for himself and divides the rest
    equally among 3 friends. How many cookies does each friend get?

**Solutions (all correct, ground truth = 12):**
0) The total number of cookies baked is 48.

Step 1: The baker keeps one-quarter for himself.
One-quarter of 48: 48 / 4 = 12 cookies kept.

Step 2: Find the remaining cookies.
Remaining = 48 - 12 = 36 cookies.

Step 3: Divide among 3 friends.
Each friend gets: 36 / 3 = 12 cookies.

Check: 12 + 12 + 12 + 12 = 48 (ok)

Each friend receives $\\\\boxed{{12}}$ cookies.

1) Cookies the baker keeps: 48 / 4 = 12

Remaining to share: 48 - 12 = 36

Each friend gets: 36 / 3 = 12

$\\\\boxed{{12}}$

### EVALUATION
**Initial Screening:** Both solutions reach 12. No disagreement.

**Analysis:**
- Solution 0: Correct and systematic, but adds a verification step (12 + 12 + 12 + 12 = 48) that
    restates values already established by the arithmetic. The labeled steps are helpful, but the
```

```
    check adds length without new information.
- Solution 1: Three concise lines, one arithmetic step each. Reaches the same conclusion without
    redundancy.

**Comparison:** Both are correct. Solution 1 is more direct -- the verification in Solution 0 is
    unnecessary when every step is already clearly shown. Solution 1 is preferred.

**Judgment: [1]**

---

## EVALUATION C
**Problem:** A car drives at 60 mph for 2 hours, then at 40 mph for 1.5 hours. How many miles does
     it travel in total?

**Solutions (all correct, ground truth = 180):**
0) The car drives at 60 mph for 2 hours, so that's 60 * 2 = 120 miles. Then it goes at 40 mph for
    1.5 hours, which is 40 * 1.5 = 60 miles. Total: 120 + 60 = 180 miles. The answer is $\\\\boxed
    {{180}}$.

1) Distance in first segment: 60 mph * 2 hours = 120 miles

Distance in second segment: 40 mph * 1.5 hours = 60 miles

Total distance: 120 + 60 = 180 miles

$\\\\boxed{{180}}$

### EVALUATION
**Initial Screening:** Both solutions reach 180. No disagreement.

**Analysis:**
- Solution 0: Correct, but merges all steps into a continuous sentence. While compact, it makes it
     harder to scan each individual computation.
- Solution 1: Computes each segment on a dedicated labeled line; the final sum is on its own line.
     Each step is easy to locate and verify in isolation.

**Comparison:** Solution 1's one-operation-per-line structure is easier to read and audit than
    Solution 0's run-on prose. Solution 1 is preferred for clarity.

**Judgment: [1]**

---

## EVALUATION D
**Problem:** {problem}

**Solutions (indices 0--{max_idx}):**
{solutions}

### EVALUATION
```

System prompt (used when `instruct=True`):

GSM8K – Select Best Correct Completion (System)

```
You are an expert evaluator of mathematical word problem solutions. Given a
word problem and candidate solutions, determine which demonstrates the best
reasoning quality.

Evaluate on:
```

```
- **Problem comprehension**: Does the solution correctly identify all
quantities and relationships?
- **Step-by-step correctness**: Are all calculations accurate?
- **Logical clarity**: Is the reasoning easy to follow?
- **Completeness**: Are all sub-steps documented?
- **Directness**: Is the approach efficient without unnecessary repetition?

Provide a brief comparative analysis, then end your response with exactly:
**Judgment: [IDX]**
where IDX is the 0-based index of the best solution.
```

User prompt (used when `instruct=True`):

GSM8K – Select Best Correct Completion (User)

```
**Problem:** {problem}

**Solutions (indices 0-{max_idx}):**
{solutions}
```