BEYOND SOLVING: A CLOSER LOOK AT LLMS AS SOLUTION VERIFIERS

Jack Lu* **Ryan Teehan*** **Jinran Jin** **Mengye Ren**

New York University

ABSTRACT

Large language models (LLMs) can act as both problem solvers and solution verifiers, where the latter select high-quality answers from a pool of solver-generated candidates. This raises the question of under what conditions verification pays off in solver–verifier systems. In their limited study of the factors influencing verification performance, prior work focused primarily on self-verification and tended to examine neither the relationship between solver and verifier model families nor post-training. To rectify this, we present a systematic study across 37 models spanning multiple families, sizes, and base vs. post-trained variants, evaluated on 9 benchmarks covering logical reasoning, structured puzzles, symbolic computation, mathematics, commonsense, factual recall, and domain knowledge. In order to support our analysis, we introduce and empirically validate *verifier gain*, a metric that predicts the performance improvements from *test-time verifier-based rejection sampling*. Our experiments find that 1) verification across model families is more effective than either self-verification or verification within the same family, and more generally that the benefits of verification decrease as the solver and verifier become more similar, 2) post-training weakens self-improvement abilities but strengthens cross-family improvement, and 3) some tasks are inherently more amenable to improvement through verification, particularly mathematical and logical tasks.

1 INTRODUCTION

Problem-solving with LLMs has progressed beyond querying a standalone model for a solution to a system where generated solutions are verified by other models. Test-time solution verification is integral to a variety of concrete approaches, including simple strategies, such as generating multiple candidate solutions and using a verifier as a filter (Zhao et al., 2025), as well as more complex strategies, such as iterative refinement (Madaan et al., 2023). With test-time solution verification, LLMs can solve more complex problems than when used alone (Cobbe et al., 2021; Lightman et al., 2024).

Despite the ubiquity of solution verification, studies of solver–verifier interactions have remained limited in scope. Prior work has largely examined how a single model verifies its own solutions (self-verification) and improves its own solutions (self-improvement) (Song et al., 2025). However, self-verification is not guaranteed to be effective. Models may be biased toward their own reasoning patterns, and their training may reinforce these tendencies. Additionally, this focus on self-verification offers little insight into verification performance when the solver and verifier differ. We therefore broaden our analysis to additionally include both intra-family and cross-family verification and ask the following central question:

> *When does verification actually pay off, and how do factors such as model family, model size, post-training, solver–verifier similarity, and task type influence its effectiveness?*

To accomplish this, we evaluate verifiers across a diverse suite of tasks, including synthetic tasks used to test precise logical reasoning or symbolic computation (3SAT, Sudoku, and matrix multiplication), mathematical reasoning tasks (AIME (Mathematical Association of America,

*Equal contribution.

2025), GSM8K (Cobbe et al., 2021)), commonsense and factual reasoning (CSQA (Talmor et al., 2019), GPQA (Rein et al., 2024)), and broad domain knowledge (MMLU in STEM and in social sciences (Hendrycks et al., 2021)) using 37 models from 7 model families. Using open-source model families that have base and post-trained pairs, size variants, reproducible inference pipelines, and explicit reasoning traces, we can study verification systematically.

We find that self-verification does not always "pay off." Models often favor solutions resembling their own reasoning (Section 5.2, 5.3) and post-training can sharpen this bias (Section 5.4), harming performance during self-verification or intra-family verification. Additionally, we find that some tasks inherently benefit less from verification than others (Section 5.5). Therefore, we present the following contributions, which offer actionable and empirically supported guidance for how to use verifiers effectively.

- **New Metric: Verifier Gain.** Verifier accuracy alone provides an incomplete picture of verifier usefulness at test time. To address this, we derive *verifier gain*, a metric that simulates the improvement obtained from a verifier during test-time rejection sampling. We empirically study rejection sampling with verifiers and show that this theoretical formulation closely reflects empirical performance trends.
- **Self-Improvement, Intra-Family Improvement, and Cross-Family Improvement.** We find that cross-family verification is often more beneficial than intra-family verification or self-verification, noting that it compares particularly favorably to the latter. Looking deeper, we show that the verifier gain decreases as the solution distributions of the solver and verifier become more similar. Furthermore, our results suggest that as models become stronger, whether through increased scale, post-training, or simply higher solver accuracy, they become less effective as self-verifiers and more effective as cross-family verifiers.
- **Dataset Verifiability.** We study whether tasks that are easy to solve are also easy to verify and whether some tasks are inherently more verifiable than others. We find that verification accuracy generally correlates with solver accuracy, though self-verification yields little verifier gain across all tasks. We also observe that a clear subset of tasks involving mathematical or logical reasoning consistently produces higher verifier gains.

2 Related Work

Verifiers. Broadly, verifiers can operate on the *outcome* level, by judging only the correctness of the final answer, or the *process* level, by judging the correctness of the intermediate reasoning steps. Weng et al. (2023), Wu et al. (2024), and Jiang et al. (2024) develop methods for outcome-level self-verification by predicting parts of the question conditioned on the solution. In order to reduce hallucinations, Dhuliawala et al. (2024) have language models fact-check their own generations by generating fact-check questions. Researchers have also trained general-purpose outcome-level verifiers (Hosseini et al., 2024; Zhang et al., 2025; Cobbe et al., 2021) and value models (Yu et al., 2024), either independently or jointly with the solver (Shen et al., 2021; Sareen et al., 2025). Work on process-level verification has focused on deductively verifying Chain-of-Thought (CoT) reasoning (Ling et al., 2023), verifying individual proof steps (Yang et al., 2022), and training process reward models for mathematical reasoning (Luo et al., 2025). Song et al. (2025) investigate the performance improvement caused by using an outcome-level verifier (the GV-Gap), and how this improvement changes as the solver or verifier increases in capacity. However, they primarily focus on cases where the solver and verifier are the same model. Additionally, they only study base models and do not consider post-trained models in their analysis. We finally note concurrent work by Zhou et al. (2025), which also studies the factors that influence test-time verification. In contrast to our work, they focus on the effects of problem difficulty and generator capability, and do not investigate the effects of model family or post-training.

Scaling test-time compute. Recently, prior work focused on studying scaling test-time compute with verifiers. For example, Zhao et al. (2025) study random sampling with self-verification, Chen et al. (2025) study combining parallel sampling with self-correction, and Singhi et al. (2025) compare Self-Consistency (Wang et al., 2023) to scaling with a generative verifier. Snell et al. (2025) investigate compute-optimal approaches to test-time scaling with process-level verification. Finally, verifiers also have their limitations. Imperfect verifiers can produce false positives (Stroebl et al., 2024), eliminate valid reasoning paths (Yu et al., 2025), and fail to select the right solution (Brown et al., 2024). We present additional related work on test-time verification in Appendix A.

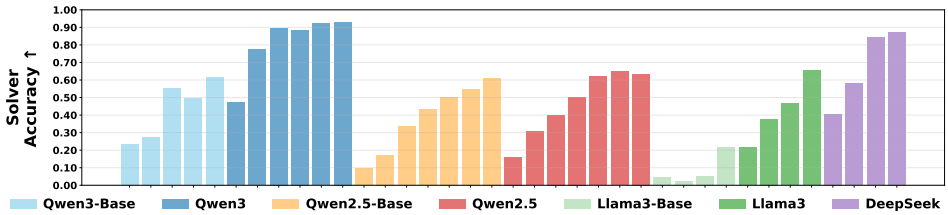

Figure 1: Average solver accuracy of each model over all datasets. Base model families are suffixed by **-Base**. Models within each family are ordered in increasing sizes. We show information for each evaluated model in Table 1.

Self-improvement. Researchers have also studied LLM self-improvement and self-evaluation, with some voicing skepticism (Huang et al., 2024b; Kamoi et al., 2024; Olausson et al., 2023; Panickssery et al., 2024). On the other hand, recent work has provided a theoretical framework for self-improvement via distribution sharpening, and empirical support alongside (Huang et al., 2024a). Zhang et al. (2024) look specifically at self-improvement for small models, arguing that they need to be paired with a stronger verifier. Some practical methods for self-improvement use natural language feedback (Madaan et al., 2023; Shinn et al., 2023; Kim et al., 2023) or train models for self-correction explicitly (Welleck et al., 2023). Other methods use tools (Gou et al., 2024), particularly code interpreters (Zhou et al., 2024), to iteratively improve solutions.

3 Preliminaries

In this section, we establish the framework used throughout this work. We define datasets, solvers, and verifiers, introduce the metrics used to evaluate solver and verifier behaviors, and specify the verification settings in our empirical analysis.

3.1 Dataset, Solvers, and Verifiers

Let $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}^\star$ be a dataset of pairs $(x, \mathcal{Y}_x)$, where $x \in \mathcal{X}$ is a problem and $\mathcal{Y}_x \subseteq \mathcal{Y}$ is a non-empty set of correct solutions. A solver $S : \mathcal{X} \to \mathcal{Y}$ is an LLM that produces a solution y for a given problem x, and a verifier $V : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ is an LLM that evaluates a problem–solution pair and returns a binary judgment. Following Song et al. (2025), who find chain-of-thought (CoT) verification more stable than multiple-choice formats, we instruct both solvers and verifiers to generate CoT reasoning before producing their final answers and judgments. We define the correctness indicator as $c(x, y) = \mathbb{1}\{y \in \mathcal{Y}_x\}$.

3.2 Evaluation Metrics

We define the accuracy of a solver S on a dataset $\mathcal{D}$ as the expected correctness of its outputs over all problems in the dataset: $\mathbb{E}_{(x,\mathcal{Y}_x)\sim\mathcal{D},\, y\sim S(x)}\big[\, c(x, y) \,\big]$. Verifier performance has several dimensions. We report common binary classification metrics, including verifier accuracy, false positive rate (FPR), false negative rate (FNR), F1-Score, and precision, with their definitions in Appendix B.

Our primary goal is to evaluate whether using a verifier V can improve a solver S at test time via rejection sampling, in which solver outputs are repeatedly sampled until the verifier accepts one. Assuming the solver has a non-zero probability of sampling a correct solution, and in the limit of infinite resampling, the expected correctness of the accepted solution converges to the verifier's precision, i.e., the proportion of accepted solutions that are actually correct. To quantify the improvement from combining a solver with a verifier, we define **verifier gain**:

$$\text{Gain}(S, V; \mathcal{D}) = \text{Precision}(S, V; \mathcal{D}) - \text{SolverAcc}(S; \mathcal{D}). \tag{1}$$

It is worth noting that verifier gain is an asymptotic metric: it reflects the limit of infinite sampling and therefore serves as a bound on the improvement attainable by verifier-based rejection sampling. Throughout this work, we use verifier gain to compare differences in verifier behavior.

3.3 Verification Settings

We group models into families (e.g., `Llama3`, `Qwen2.5`), where each family contains related models of varying sizes. Because base and post-trained models often exhibit substantially different behaviors, we treat them as distinct families. For example, the base model `meta-llama/Meta-Llama-3-70B` belongs to the `Llama3-Base` family, and the post-trained model `meta-llama/Meta-Llama-3-8B-Instruct` belongs to the `Llama3` family.

We study three different verification settings, defined by the relationship between the solver and verifier model:

- **Self-Verification.** The solver and verifier are the same model, so the model verifies its own solutions. For example, when a 70B `Llama3` model is used as both the solver and verifier, the verification metric (e.g., accuracy, FPR) is computed on this single pairing.
- **Intra-Family Verification.** The verifier evaluates solutions produced by other models within the same family. For example, a 70B `Llama3` verifier may evaluate outputs from 8B or 13B `Llama3` solvers. The reported metric is averaged over all such within-family solvers, excluding the self-verification case.
- **Cross-Family Verification.** The verifier evaluates solutions produced by models from different families. For example, a base `Llama3` verifier may evaluate outputs from `Qwen3` or from a post-trained `Llama3`. The reported metric is averaged over all such cross-family solvers.

During our evaluation, we partition our computed verifier metrics according to the verification setting and average within each partition.

4 Experimental Setup

Models. We evaluate the solver and verifier abilities of 21 post-trained models from the `Llama3` (Grattafiori et al., 2024), `Qwen2.5` (Qwen et al., 2024), `Qwen3` (Yang et al., 2025), and `DeepSeek-R1` (DeepSeek-AI et al., 2025) families. For our study of post-training effects in Section 5.4, we additionally evaluate 12 base models from the `Qwen2.5-Base` and `Qwen3-Base` families. Model sizes range from 0.5B to 72B parameters. The full model list, with sizes, families, and HuggingFace identifiers, is provided in Appendix D. The legend of Figure 1 displays the seven model families and the color scheme assigned to each.

Datasets. We compile a broad suite of real-world and synthetic tasks spanning diverse domains, including tasks requiring mathematical reasoning (GSM8K, AIME), commonsense knowledge (CSQA), and domain-specific factual knowledge of varying breadth (MMLU STEM, MMLU Social Sciences, GPQA). We also construct synthetic tasks to assess logical reasoning (3SAT), structured puzzle solving (Sudoku), and symbolic computation (Matrix Multiplication). Further dataset details, along with examples of our synthetic tasks, are provided in Appendix E.

Evaluation. Datasets such as Matrix Multiplication and the natural-language benchmarks contain a single ground-truth answer per problem. For these, we extract boxed solver outputs and evaluate them via exact matching. In contrast, datasets like Sudoku and 3SAT may admit multiple valid solutions, so we evaluate solver outputs according to the rules of the respective task. To evaluate verifiers, we prompt each model to generate CoT reasoning given the problem and solver answer before producing a boxed "correct" or "incorrect," from which we extract the final judgment.

Implementation. For both solvers and verifiers, we generate with temperature 0.7, top-p 0.9, and a maximum output length of 8192 tokens. We discard outputs that do not contain a boxed answer. All inference experiments are run using vLLM on H200 GPUs. Prompts and additional details on output filtering are provided in Appendix F.

5 Results

5.1 Does Verifier Gain Predict Improvements from Resampling?

We first empirically validate our verifier gain metric, which estimates the expected improvement in a solver's accuracy when using a verifier for rejection sampling. To assess how well this metric predicts real performance, we conduct rejection sampling experiments across all solver–verifier pairs from a 12-model subset of our post-trained models, consisting of the three smallest models from each of the four post-training families. For each problem in each dataset, the solver generates solutions

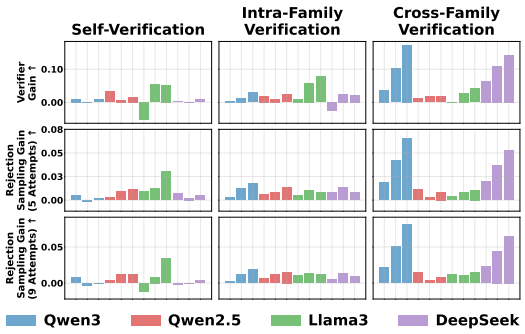

Figure 2: Comparison of theoretical and empirical verifier gains (rows) across verification settings (columns). Row 1 shows verifier gains computed from Equation 1. Rows 2 and 3 each show the gains from rejection sampling.

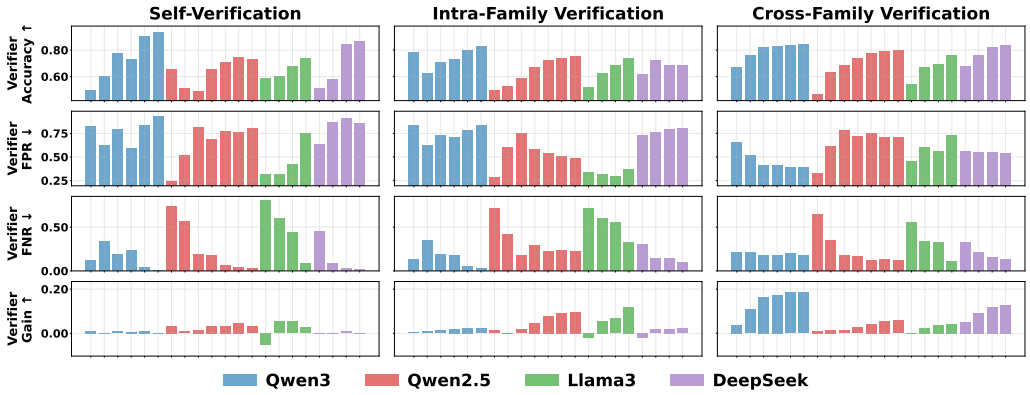

Figure 4: Here we show each verifier's metric (rows) plotted against model size for all 21 post-trained models, averaged over all datasets. In each plot, models are separated by family and ordered by increasing size. Each verifier metric is computed over our three verification settings (columns).

until the verifier labels one as correct, up to nine attempts. If no such solution is found, we retain the final attempt. The empirical results, along with the corresponding theoretical verifier gains, are shown in Figure 2. Although the measured improvements can be noisy when verifier gains are small, and the nine-attempt cap limits the rejection sampling procedure, the overall trends align closely with theoretical predictions.

Takeaway: The verifier gain given by Equation 1 is a reliable predictor of performance improvements under rejection sampling. Crucially, it can be estimated from one round of verification without requiring computationally expensive rejection sampling experiments.

5.2 Do Better Solvers Make Better Verifiers?

We next study the relationship between solver and verifier performance, and investigate how to best measure the benefits of verification.

Solver performance. We benchmark the performance of all 37 models on each of our 9 datasets, averaging performance across tasks (Figure 1) and reporting task-level results in Appendix G. Overall, solver accuracy increases with model capacity. Models in the `Qwen3` and `DeepSeek` families perform particularly well, whereas `Llama3-Base` performs poorly due to base models being unfamiliar with the question–answering instruction format. Within each family, we observe clear performance scaling for `Qwen2.5-Base` and `DeepSeek`, with the remaining families showing similar upward trends.

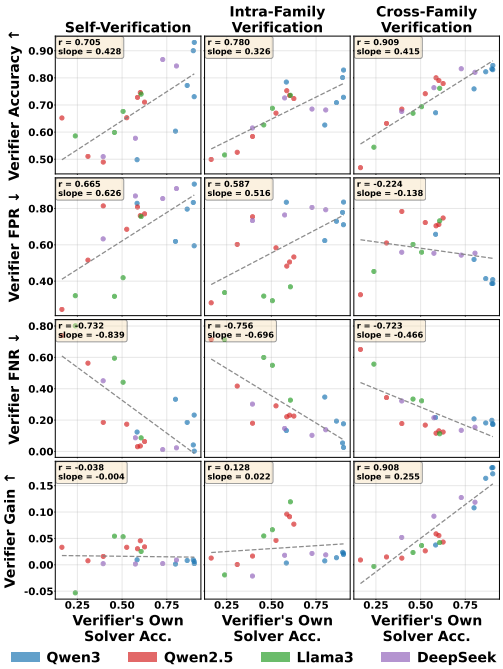

Figure 3: Correlation between each verifier's metrics (rows) and the verifier model's own solver accuracy for all 21 post-trained models, averaged over all datasets. Each verifier metric is computed over our three verification settings (columns).

Correlating verifier and solver performance. After establishing solver accuracy, we analyze whether a model's solver performance correlates with its performance as a verifier (Figure 3). For each of our 21 post-trained models and each dataset, we evaluate verification on the same set of solver models to obtain verifier accuracy, F1-score, precision, FPR, FNR, and gain for every solver–verifier pair. For each verifier, we then partition the results by our three verification settings and average within each setting over solvers and datasets.

While verifier accuracy always improves with the verifier's own solver accuracy, the relationship becomes more nuanced when examining other metrics. The FPR decreases during cross-family verification but increases during self-verification and intra-family verification, indicating that when a strong solver is used as a verifier, it falsely accepts solutions generated by itself or another model in its family. We provide additional visualizations for F1-score and precision in Appendix H.

To better understand the impact of increasing FPR, we examine verifier gain in the final row. This visualization offers a clearer view of verification quality: self-verification yields the smallest gains, and more accurate solvers do not exhibit greater self-improvement. Gains increase slightly in intra-family verification, while cross-family verification provides the greatest potential benefits.

Examining verifier performance at different model families and sizes. In Figure 4, we repeat the experiments from Figure 3 but plot each verifier metric as a function of model size within each model family. Verification accuracy and FNR consistently improve as models become larger, whereas FPR behaves more inconsistently, often increasing with model size (e.g., intra-family verification for `DeepSeek`).

For state-of-the-art post-trained models such as `Qwen3` and `DeepSeek`, we find that verifier gains are largest in the cross-family setting, smaller in the intra-family setting, and minimal during self-verification. This appears to contradict Song et al. (2025), who report that self-verification GV-Gaps increase with more pretraining FLOPs. However, their analysis focuses on older model families, and Figure 3 likewise shows larger verifier gains for older models such as `Qwen2.5` and `Llama3`.

We hypothesize that stronger post-trained models like `DeepSeek` and `Qwen3` show negligible gains in self-verification and limited gains in intra-family verification for two reasons: (a) they may already engage in *spontaneous* self-verification when used as solvers, reducing the benefit of an additional *forced* verification round, and (b) their distributions are significantly sharpened by post-training (Huang et al., 2024a), which limits the improvement obtained from rejection sampling.

Takeaways:

- Verifier models are biased toward accepting incorrect solutions when performing self-verification or intra-family verification.
- Verification accuracy alone is not a reliable predictor of how much a verifier can improve a solver at test time. Instead, computing verifier gain using solver accuracy and verifier precision provides a more reliable metric.
- While model families like `Llama3` and `Qwen2.5` show some ability to self-improve, stronger model families like `DeepSeek` and `Qwen3` do not.

5.3 Are Verifiers Biased Toward Solutions That Resemble Their Own?

Humans tend to judge solutions that resemble their own reasoning as more likely to be correct. This mirrors self-enhancement bias (Krueger, 1998), where individuals evaluate themselves more favorably than objective evidence would suggest. Our results in Figures 3 and 4 show that strong reasoning models benefit the least from self-verification and the most from cross-family verification, suggesting that an analogous effect may arise in solver–verifier interactions.

To directly investigate this behavior, we study the relationship between our verifier performance metrics and the **solver–verifier similarity score**, defined as the average cosine similarity between the two models' solution embeddings across all dataset problems. We conduct cross-verification experiments using 12 post-trained models (the three smallest models from each of the four families) and compute all verifier metrics for each pair. For intra-family verification, each solver has 2 verifiers from the same family (excluding itself),

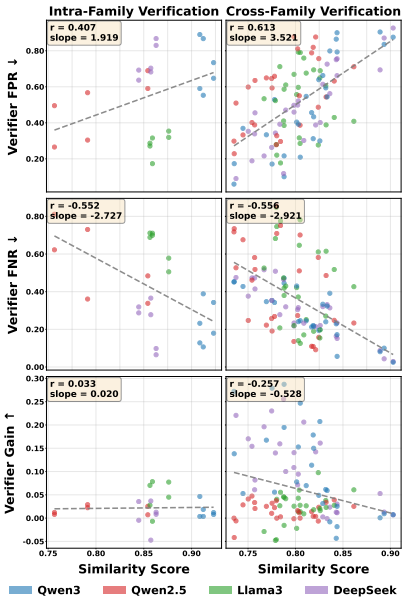

Figure 5: Correlation between verifier metrics and similarity scores.

resulting in $12 \times 2 = 24$ solver–verifier pairs. For cross-family verification, each solver has 9 verifiers from other families, giving $12 \times 9 = 108$ cross-family pairs. Solutions are embedded using `sentence-transformers/all-mpnet-base-v2`.

Figure 5 shows that, for both intra-family and cross-family settings, the more similar the solver and verifier are in their solution distributions, the more likely the verifier is to accept the solver's incorrect answers. While intra-family verifier gains are too small to yield a strong correlation, cross-family verifier gains decrease significantly as solver–verifier similarity increases. This indicates that choosing a verifier whose solution distribution differs from that of the solver leads to more reliable verification.

Takeaway: Higher similarity between solver and verifier solution distributions increases the verifier's tendency to accept incorrect solver outputs, reducing verifier gain. Using a verifier with a meaningfully different solution distribution mitigates this bias.

5.4 How Does Post-Training Affect Solver and Verifier Performance?

We examine how post-training influences verifier behavior. Our analysis focuses on the `Qwen2.5-Base`/`Qwen2.5` and `Qwen3-Base`/`Qwen3` model pairs, since other model families are either too weak (`Llama3-Base`) or do not have base models available for comparison (`DeepSeek`). Verification metrics are computed across all 37 base and post-trained models.

Post-training's effect on solver performance. We begin by evaluating how solver accuracy changes after post-training. For each model pair, we compute solver accuracy and average results across model families and datasets. As expected, post-training yields substantial improvements: `Qwen2.5` solvers improve by an average of 8.2%, while `Qwen3` solvers show a striking 35.4% gain (Appendix I).

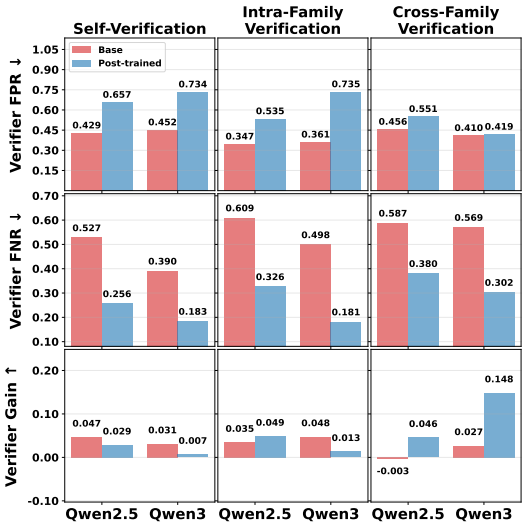

Figure 6: Changes in verifier metrics of the `Qwen2.5-Base` and `Qwen3-Base` models from post-training.

Post-training's effect on verifier performance. We next analyze how post-training affects verifier behavior (Figure 6). For each model, we compute verifier metrics against all solvers and datasets, partition results by verification setting, and average within families. For both Qwen families, post-training increases FPR and reduces verifier gain in self-verification, the latter despite improvements in FNR. Although `Qwen3` benefits more than `Qwen2.5` in solver accuracy (Figure 11), its post-trained verifiers show higher FPRs and lower gains in both self- and intra-family verification. In contrast, both families, and especially `Qwen3`, show substantial improvements in cross-family verification. Post-training tends to exacerbate the trends we discovered previously, increasing false positives and limiting the benefits of self-verification.

Takeaway: Post-training significantly enhances a base model's problem-solving ability but can reduce its self- or intra-family improvement potential. In contrast, it boosts models' performance in cross-family verification.

5.5 How Does Task Type Affect Verifiability?

Thus far, we have examined the influence of model-related factors on verifier performance. We now shift to a task-level perspective and ask: *are tasks that are easy to solve also easy to verify?* In Figure 7, we recompute the verifier metrics from Section 5.2, average them across all verifier models, and plot them against solver accuracies.

While we find a strong correlation between the solver accuracy and verifier accuracy in all verification settings, the picture is more mixed when we study the correlation with verifier gains.

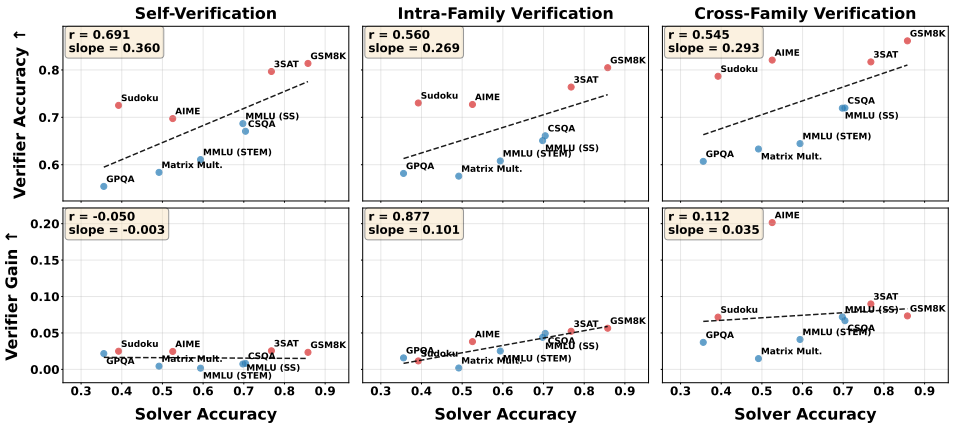

Figure 7: Correlation of verifier metrics (rows) with solver accuracies, averaged over solver-verifier pairs that belong to each verification setting (columns).

During self-verification, we find essentially no correlation between the verifier gain and the solver accuracy, but once we move to the intra-family and cross-family settings, we find a clear positive relationship. Notably, AIME appears as an outlier in the final plot, potentially because some models have encountered similar problems during post-training.

The best-fit lines for verifier accuracy further reveal two distinct clusters of tasks (colored red and blue), leading us to our next question: *are some tasks inherently easier to verify than others?* In Figure 7, we observe that AIME, GSM8K, 3SAT, and Sudoku exhibit a higher ratio of verifier accuracy to solver accuracy and deliver higher gains across all verification settings.

To explain this, we notice that among our synthetic datasets, Sudoku and 3SAT are classic examples of problems that require exponential solving time but allow polynomial-time verification. By contrast, there is no clear shortcut for verifying the product of two matrices without effectively recomputing it for Matrix Multiplication. Among the real-world datasets, GSM8K and AIME involve problems solvable with high-school-level mathematics, whereas MMLU (Social Sciences) requires domain-specific knowledge, CommonsenseQA relies on implicit world knowledge, and GPQA and MMLU (STEM) draw on specialized natural science knowledge. For these latter factual recall tasks, verifying an answer requires essentially the same knowledge as solving the problem.

Takeaways:

- Tasks that are easy to solve tend to be easy to verify.
- Tasks that are easier to solve tend to be more improvable through intra-family and cross-family verification, but not necessarily through self-verification.
- Synthetic problems with logical or structured reasoning, as well as real-world tasks which rely on mathematical reasoning, are inherently easier to verify and yield larger verifier gains than those which require factual recall.

6 Conclusion

This work presents a comprehensive study of LLM-based verification for problem solving. We show that verification accuracy alone provides an incomplete picture of the benefits of verification, motivating the introduction of *verifier gain*, which measures the expected improvement from using a verifier for test-time rejection sampling. We show lower expected improvements from self-verification and intra-family verification when compared to cross-family verification, and that these trends are exacerbated by post-training and increasing model size. Further analysis reveals that decreases in verifier gain correlate with greater similarity between the solver's and verifier's solution distributions, indicating that verifiers are biased to accept solutions that resemble their own. Examining the effects of task type on verification, we find that easier tasks are also the ones where performance can be improved with intra-family or cross-family verification. Finally, we show that some tasks are inherently easier for LLMs to verify than others: more difficult tasks generally require domain-specific or implicit world knowledge, whereas easier tasks tend to involve logical reasoning, mathematical reasoning, or structured puzzle solving.

BRIEF NOTE ON LIMITATIONS, SAFETY, AND ETHICS

Section 5.5 shows that some tasks are inherently more verifiable than others, motivating future work on developing a predictive model for the verifiability of individual tasks or questions. Section 5.3 shows that LLMs are biased toward accepting incorrect solutions that resemble their own reasoning, indicating that it will be worthwhile to examine the origins of this bias in pre-training and/or post-training.

This paper studies when test-time verification improves LLM outputs and provides evidence-driven guidance for choosing verifier configurations across models and tasks. While this may be used in beneficial or neutral settings, such as the tasks we consider in this paper, test-time verification could be used for harmful ends, such as verifying the correct procedure when designing a weapon. This danger is not inherent to our work in particular, but more broadly to research that improves the effectiveness of test-time verification.

REFERENCES

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Jiefeng Chen, Jie Ren, Xinyun Chen, Chengrun Yang, Ruoxi Sun, Jinsung Yoon, and Sercan Ö Arık. Sets: Leveraging self-verification and self-correction for improved test-time scaling. *arXiv preprint arXiv:2501.19306*, 2025.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. Universal self-consistency for large language models. In *ICML 2024 Workshop on In-Context Learning*, 2024.

Yinlam Chow, Guy Tennenholtz, Izzeddin Gur, Vincent Zhuang, Bo Dai, Aviral Kumar, Rishabh Agarwal, Sridhar Thiagarajan, Craig Boutilier, and Aleksandra Faust. Inference-aware fine-tuning for best-of-n sampling in large language models. In *ICLR*, 2025.

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

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda

Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. In *ACL Findings*, 2024.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *ICLR*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu

Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *ICLR*, 2021.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-STar: Training verifiers for self-taught reasoners. In *COLM*, 2024.

Audrey Huang, Adam Block, Dylan J Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan T Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. *arXiv preprint arXiv:2412.01951*, 2024a.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *ICLR*, 2024b.

Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James Kwok. Forward-backward reasoning in large language models for mathematical verification. In *ACL Findings*, 2024.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. 2024.

Jikun Kang, Xin Zhe Li, Xi Chen, Amirreza Kazemi, Qianyi Sun, Boxing Chen, Dong Li, Xu He, Quan He, Feng Wen, et al. Mindstar: Enhancing math reasoning in pre-trained llms at inference time. *arXiv preprint arXiv:2405.16265*, 2024.

Muhammad Khalifa, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, and Lu Wang. GRACE: Discriminator-guided chain-of-thought reasoning. In *EMNLP Findings*, 2023.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. In *NeurIPS*, 2023.

Joachim Krueger. Enhancement bias in descriptions of self and others. *Personality and Social Psychology Bulletin*, 1998.

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

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. In *NeurIPS*, 2023.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models with automated process supervision. *arXiv preprint arXiv:2406.06592*, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. 2023.

Mathematical Association of America. 1983-2025 American Invitational Mathematics Examination (AIME) I: Problems and Solutions, February 2025. URL `https://artofproblemsolving.com/wiki/index.php?title=AIME_Problems_and_Solutions`. Art of Problem Solving Wiki entry.

Eric Nichols, Leo Gao, and Randy Gomez. Collaborative storytelling with large-scale neural language models. In *ACM SIGGRAPH Conference on Motion, Interaction and Games*, 2020.

Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? *arXiv preprint arXiv:2306.09896*, 2023.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. Llm evaluators recognize and favor their own generations, 2024.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *COLM*, 2024.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *EMNLP*, 2015.

Kusha Sareen, Morgane M Moss, Alessandro Sordoni, Rishabh Agarwal, and Arian Hosseini. Putting the value back in rl: Better test-time scaling by unifying llm reasoners with verifiers. *arXiv preprint arXiv:2505.04842*, 2025.

Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu. Generate & rank: A multi-task framework for math word problems. In *EMNLP*, 2021.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Nishad Singhi, Hritik Bansal, Arian Hosseini, Aditya Grover, Kai-Wei Chang, Marcus Rohrbach, and Anna Rohrbach. When to solve, when to verify: Compute-optimal problem solving and generative verification for llm reasoning. *arXiv preprint arXiv:2504.01005*, 2025.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *ICLR*, 2025.

Yuda Song, Hanlin Zhang, Carson Eisenach, Sham M. Kakade, Dean Foster, and Udaya Ghai. Mind the gap: Examining the self-improvement capabilities of large language models. In *ICLR*, 2025.

Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. Inference scaling flaws: The limits of llm resampling with imperfect verifiers. *arXiv preprint arXiv:2411.17501*, 2024.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *NAACL 2019*, 2019.

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 *ICLR*, 2023.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. In *ICLR*, 2023.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In *EMNLP Findings*, 2023.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. Large language models can self-correct with key condition verification. In *EMNLP*, 2024.

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

Kaiyu Yang, Jia Deng, and Danqi Chen. Generating natural language proofs with verifier-guided search. In *EMNLP*, 2022.

Fei Yu, Anningzhe Gao, and Benyou Wang. OVM, outcome-supervised value models for planning in mathematical reasoning. In *NAACL Findings*, 2024.

Fei Yu, Yingru Li, and Benyou Wang. Scaling flaws of verifier-guided search in mathematical reasoning. *arXiv preprint arXiv:2502.00271*, 2025.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *ICLR*, 2025.

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. Small language models need strong verifiers to self-correct reasoning. In *ACL*, 2024.

Eric Zhao, Pranjal Awasthi, and Sreenivas Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification. In *ICML*, 2025.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. In *ICLR*, 2024.

Yefan Zhou, Austin Xu, Yilun Zhou, Janvijay Singh, Jiang Gui, and Shafiq Joty. Variation in verification: Understanding verification dynamics in large language models. *arXiv preprint arXiv:2509.17995*, 2025.

APPENDIX

A ADDITIONAL RELATED WORK

Scaling test-time compute. A simple method for scaling test-time compute involves sampling several candidates and selecting one *post hoc*, for example via Best-of-N. This can take the form of sample-and-rank approaches (Nichols et al., 2020), majority vote (Wang et al., 2023), model-based aggregation (Chen et al., 2024), or sampling then filtering (Weng et al., 2023). LLMs can also be finetuned to explicitly optimize Best-of-N performance (Chow et al., 2025). Instead of *post hoc* selection, we can guide the model towards good samples via constraints (Roy & Roth, 2015), a scoring function (Yang et al., 2022), or sequential construction (Kang et al., 2024; Khalifa et al., 2023).

B ADDITIONAL DETAILS ON VERIFIER METRICS

We show the mathematical definitions of relevant verifier metrics below. For clarity, we include dependencies (e.g., $(S, V; \mathcal{D})$) in the definitions, but sometimes omit them for brevity when the context is clear.

$$
\begin{aligned}
\text{VerifierAcc}(S, V; \mathcal{D}) &= \mathbb{E}_{(x,\mathcal{Y}_x)\sim\mathcal{D},\, y\sim S(x)}\big[\, \mathbb{1}\{\, V(x,y) = c(x,y)\,\}\,\big] \\
\text{TPR}(S, V; \mathcal{D}) &= \mathbb{E}[\, V(x,y) \mid y \in \mathcal{Y}_x \,] \\
\text{FPR}(S, V; \mathcal{D}) &= \mathbb{E}[\, V(x,y) \mid y \notin \mathcal{Y}_x \,] \\
\text{FNR}(S, V; \mathcal{D}) &= \mathbb{E}[\, 1 - V(x,y) \mid y \in \mathcal{Y}_x \,] = 1 - \text{TPR}(S, V; \mathcal{D}). \\
\text{Precision}(S, V; \mathcal{D}) &= \mathbb{E}[\, c(x,y) \mid V(x,y) = 1 \,] = \frac{\text{SolverAcc} \cdot \text{TPR}}{\text{SolverAcc} \cdot \text{TPR} + (1 - \text{SolverAcc}) \cdot \text{FPR}} \\
\text{Recall}(S, V; \mathcal{D}) &= \text{TPR} \\
\text{F1}(S, V; \mathcal{D}) &= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}
\end{aligned}
$$

C ADDITIONAL DETAILS ON VERIFICATION SETTINGS

We show the mathematical definitions of our three verification settings. Let $\mathcal{M}$ denote the space of models and $\mathcal{F}$ the space of model families. We define a function

$$\text{Family} : \mathcal{M} \to \mathcal{F},$$

that maps each model (e.g., `meta-llama/Meta-Llama-3-70B`) to its corresponding family (e.g., `Llama-3-Base`). Note that $S, V \in \mathcal{M}$. For any verifier metric $M(\cdot, \cdot; \mathcal{D})$ such as VerifierAcc, TPR, or G, we define:

$$
\begin{aligned}
\text{Self-Verif}(V; \mathcal{D}, M) &= M(V, V; \mathcal{D}), \\
\text{Intra-Verif}(V, \mathcal{S}; \mathcal{D}, M) &= \frac{\sum_{\substack{S \in \mathcal{S} \\ S \neq V,\, \text{Family}(S) = \text{Family}(V)}} M(S, V; \mathcal{D})}{|\{S \in \mathcal{S} : S \neq V,\, \text{Family}(S) = \text{Family}(V)\}|}, \\
\text{Cross-Verif}(V, \mathcal{S}; \mathcal{D}, M) &= \frac{\sum_{\substack{S \in \mathcal{S} \\ \text{Family}(S) \neq \text{Family}(V)}} M(S, V; \mathcal{D})}{|\{S \in \mathcal{S} : \text{Family}(S) \neq \text{Family}(V)\}|}.
\end{aligned}
$$

D ADDITIONAL DETAILS ON MODELS

We show the information for each of our 37 evaluated models in Table 1.

E ADDITIONAL DETAILS ON DATASETS

E.1 REAL-WORLD DATASETS

Note that for MMLU (STEM) and MMLU (Social Sciences), we concatenate questions from all subjects that belong to the STEM and Social Sciences supercategories in Hendrycks et al. (2021), respectively.

E.2 SYNTHETIC DATASETS

We generate three synthetic datasets, named 3SAT, Matrix Multiplication, and Sudoku, with 1000 samples each. We submit the data generation code in the Supplementary Materials, but briefly explain each synthetic dataset's generation parameters below.

Each 3SAT CNF contains uniformly sampled numbers of variables and clauses from 2 to 8 (inclusive). Each Sudoku puzzle is a 9x9 grid with 12 randomly missing cells. Each Matrix Multiplication problem is about multiplying 2 4x4 integer matrices with values uniformly sampled from $[-5, 5]$. All data are generated in a way that ensures the existence of a valid solution. Note that while Matrix Multiplication has a singular correct answer for each problem, Sudoku and 3SAT are allowed multiple correct answers as long as the solver's answer is correct by their rules.

The generation code files for all synthetic datasets are seeded for reproducibility.

An example of a generated 3SAT problem:

```
## Problem Definition

**SAT (Boolean Satisfiability Problem)** is a fundamental problem
in computer science where we need to determine if there exists an
assignment of Boolean values (True/False) to variables that makes
a given Boolean formula evaluate to True.
**Variables**: In this problem, variables are named as single
letters. Each variable can be assigned either True (T) or
False (F).
**Literals**: A literal is either a variable (like a) or its
negation (like ~a, meaning "not a"). If a is True, then ~a is
False, and vice versa.
**Clauses**: A clause is a disjunction (OR operation) of literals.
A clause is satisfied (True) if at least one of its literals is
True. For example, the clause (a or ~b) is True if either a is
True OR b is False (or both).
**CNF (Conjunctive Normal Form)**: The Boolean formula is given in
CNF, which is a conjunction (AND operation) of multiple clauses.
The entire formula is satisfied only if ALL clauses are satisfied
simultaneously.
**3SAT**: This is a special case of SAT where every clause contains
exactly 3 literals.

## The Problem

Find a satisfying assignment for the following CNF formula:
(~c or ~b or d) and (d or ~b or ~c) and (d or a or c) and
(~c or d or a) and (b or ~a or d) and (c or d or ~b)

## Instructions

Provide your answer as a list of variable assignments, one per line,
in the format "variable_name T" or "variable_name F". For example:
\boxed{
```

```
a T
b F
}
This means a=True, b=False.

Another example answer is
\boxed{
a F
b T
}
This means a=False, b=True.

Output and only output the T/F values for the variables that appear
in the provided CNF formula.
```

An example of a generated Sudoku problem:

```
## Sudoku Problem

**Sudoku** is a logic-based number-placement puzzle. The objective is
to fill a 9x9 grid with numbers so that each column, each row, and
each of the 3x3 sub-grids contains all of the numbers from 1 to 9.

## The Puzzle

Complete the following 9x9 Sudoku grid (empty cells are marked with '_'):

7 4 2 1 _ 5 8 9 6
1 6 9 2 4 8 3 5 7
8 5 3 _ _ 7 2 1 4
2 _ 8 9 7 1 4 6 5
5 7 6 4 8 2 9 3 _
4 9 1 3 _ 6 _ 8 _
3 1 5 8 2 4 6 7 9
6 8 _ 7 1 _ 5 2 3
_ 2 7 5 6 _ 1 4 8

## Instructions

Provide your answer as a completed 9x9 grid with all numbers filled
in, formatted exactly like the puzzle above but with numbers instead
of underscores.

For example, a completed 4x4 grid should look like:
\boxed{
1 2 3 4
3 4 1 2
2 3 4 1
4 1 2 3
}
```

An example of a generated Matrix Multiplication problem:

```
## Matrix Multiplication Problem

**Matrix Multiplication** is a fundamental operation in linear
algebra where we compute the product of two matrices. For two
square matrices A and B of size 4x4, the product C = A x B is
computed as:
```

```
C[i][j] = Sum(k=0 to 3) A[i][k] x B[k][j]

## The Problem

Compute the product of the following two 4x4 matrices:

**Matrix A:**
0 1 1 4
-1 3 4 4
-2 -5 -5 0
-4 4 5 0

**Matrix B:**
1 2 0 5
1 -2 0 0
3 -1 -3 -3
2 5 -4 2

## Instructions

Provide your answer as the resulting 4x4 matrix C = A x B,
formatted with each row on a separate line and numbers separated
by spaces.

For example, a 2x2 result matrix is formatted like:
\boxed{
1 2
3 4
}
```

F Additional Details on Experimental Setup

We use the following solver prompt for all models:

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

{question}
```

We use the following verifier prompt for all models:

```
You are a teacher that is evaluating a student's answer
to a question. Your task is to determine whether the answer
is correct or incorrect.

Question: {question}

Student's Answer: {response}

Please evaluate the student's answer carefully. Consider:
- Is the answer factually accurate?
- Is the reasoning sound and logical?
- Does it fully address the question asked?

After your evaluation, provide your judgment in the
following format:
- If the answer is correct, write: \boxed{{correct}}.
- If the answer is incorrect, write: \boxed{{incorrect}}.

First explain your analysis over the student's answer,
```

```
then provide your final judgment in the boxed format.
Make sure the final judgment is either "correct" or
"incorrect" inside the \boxed{{}}. Do not put anything
else in \boxed{{}}. Do not repeat the student's answer
in \boxed{{}}.
```

Figure 8 displays the ratio of filtered solver outputs due to not containing a box for answer extraction, averaged across all datasets.

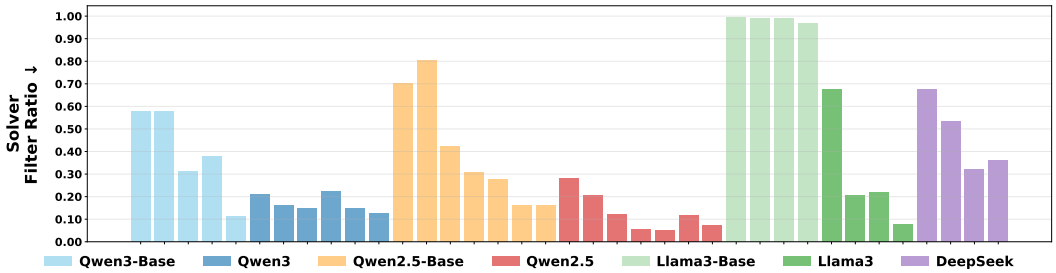

Figure 8: Average ratio of filtered solver outputs for each model over all datasets. Base model families are suffixed by **-Base**. Models within each family are ordered in increasing size.

G Solver Accuracy by Dataset

Figure 9 shows the solver accuracies of all 37 models on each of our 9 datasets.

H F1-Score and Precision Visualization

Figure 3 shows the correlation between each model's verification ability and its own solver accuracy for all 21 post-trained models. We additionally display verifier F1-Score and precision in Figure 10.

In comparison to verifier accuracy, while F1-Score also positively correlates with verifier's own solver accuracy for all verification settings, the slopes decrease from self-verification to intra-family verification, and further decrease for cross-family verification, showing that the increase in false positive rate in Figure 3 has a stronger negative impact on lowering F1-score than accuracy.

While Section 5.2 explains the low verifier gains for self- and intra-family verification through close examination of FPR, we additionally plot verifier precision in Figure 10. However, since precision is the expected performance of verifier-based rejection sampling in the limit of infinite sampling and our main metric "verifier gain" is defined in terms of it (Equation 1), precision does not help explain the differences in verifier gains across verification settings itself.

I Effect of Post-Training on Solver Performance

Figure 11 shows the average improvement in solver accuracies of `Qwen2.5-Base` and `Qwen3-Base` families of models from their respective post-training procedures.

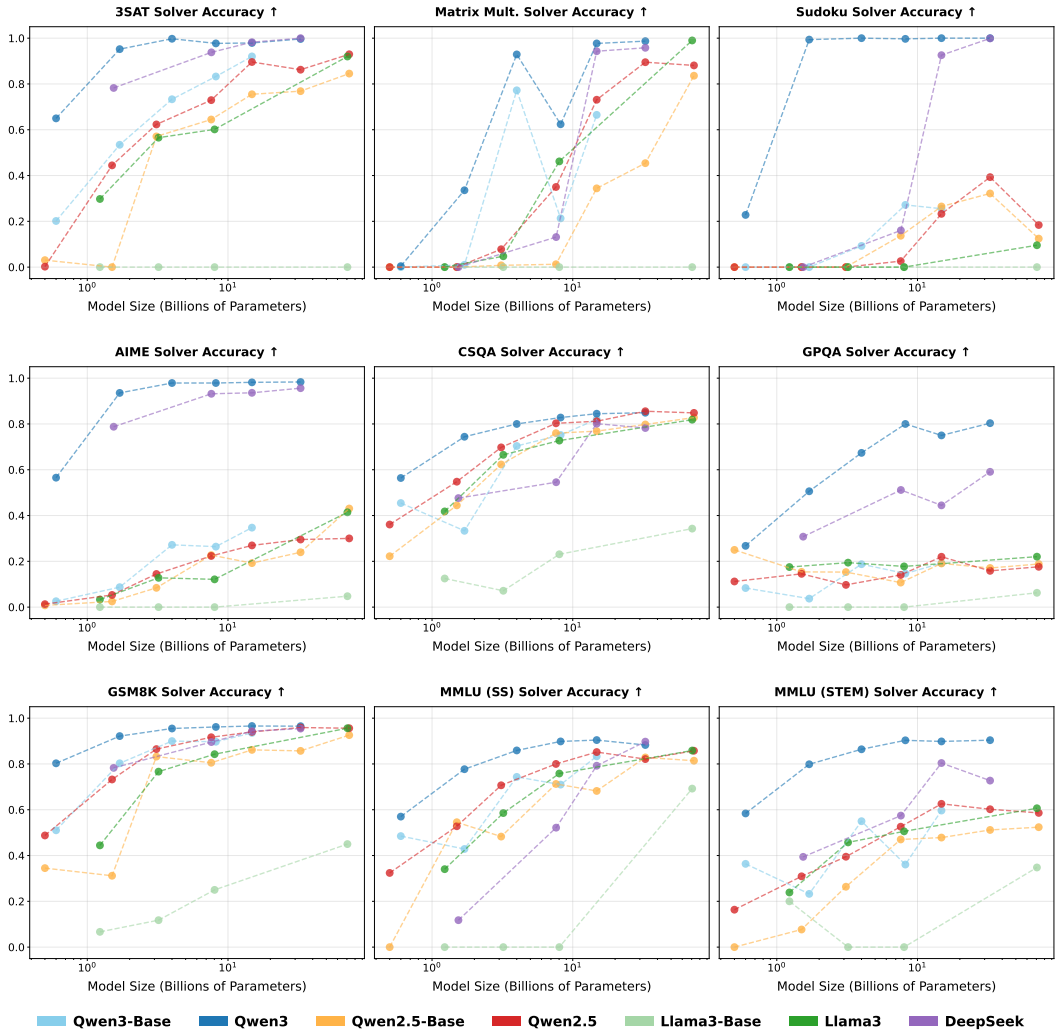

Figure 9: The solver accuracies of 37 models on each dataset.

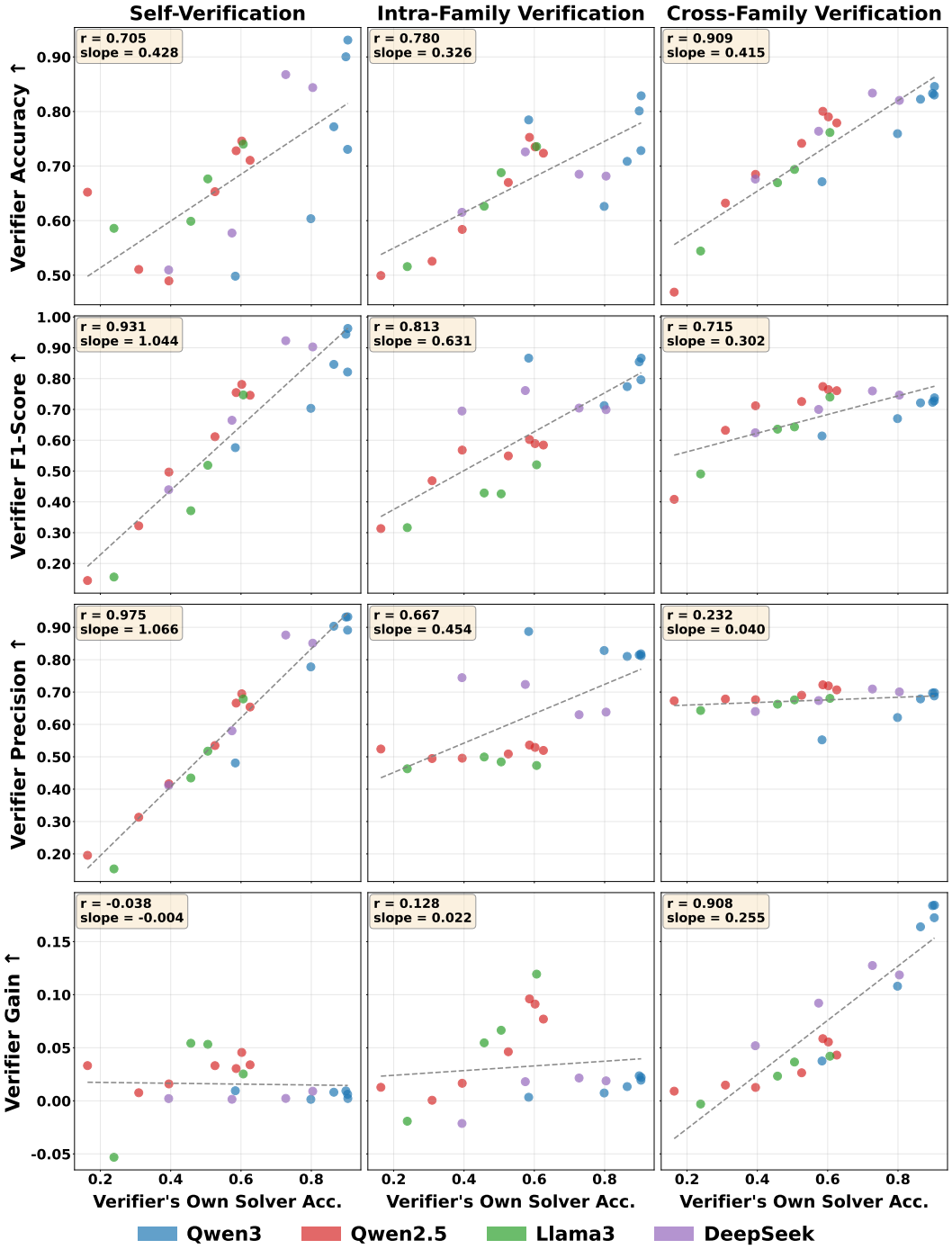

Figure 10: Correlation between each model's verifier metrics (rows) and its own solver accuracy for all 21 post-trained models, averaged over all datasets. Each verifier metric is computed over three settings (columns): self-verification, intra-family verification, and cross-family verification. We use the same set of post-trained models as the set of solver models.

Table 1: Complete list of each evaluated model's HuggingFace identifier, family, and size.

HuggingFace Identifier	Family	Size
Qwen/Qwen3-0.6B-Base	Qwen3-Base	0.6B
Qwen/Qwen3-1.7B-Base	Qwen3-Base	1.7B
Qwen/Qwen3-4B-Base	Qwen3-Base	4B
Qwen/Qwen3-8B-Base	Qwen3-Base	8B
Qwen/Qwen3-14B-Base	Qwen3-Base	14B
Qwen/Qwen3-0.6B	Qwen3	0.6B
Qwen/Qwen3-1.7B	Qwen3	1.7B
Qwen/Qwen3-4B	Qwen3	4B
Qwen/Qwen3-8B	Qwen3	8B
Qwen/Qwen3-14B	Qwen3	14B
Qwen/Qwen3-32B	Qwen3	32B
Qwen/Qwen2.5-0.5B	Qwen2.5-Base	0.5B
Qwen/Qwen2.5-1.5B	Qwen2.5-Base	1.5B
Qwen/Qwen2.5-3B	Qwen2.5-Base	3B
Qwen/Qwen2.5-7B	Qwen2.5-Base	7B
Qwen/Qwen2.5-14B	Qwen2.5-Base	14B
Qwen/Qwen2.5-32B	Qwen2.5-Base	32B
Qwen/Qwen2.5-72B	Qwen2.5-Base	72B
Qwen/Qwen2.5-0.5B-Instruct	Qwen2.5	0.5B
Qwen/Qwen2.5-1.5B-Instruct	Qwen2.5	1.5B
Qwen/Qwen2.5-3B-Instruct	Qwen2.5	3B
Qwen/Qwen2.5-7B-Instruct	Qwen2.5	7B
Qwen/Qwen2.5-14B-Instruct	Qwen2.5	14B
Qwen/Qwen2.5-32B-Instruct	Qwen2.5	32B
Qwen/Qwen2.5-72B-Instruct	Qwen2.5	72B
meta-llama/Llama-3.2-1B	Llama3-Base	1B
meta-llama/Llama-3.2-3B	Llama3-Base	3B
meta-llama/Llama-3.1-8B	Llama3-Base	8B
meta-llama/Llama-3.1-70B	Llama3-Base	70B
meta-llama/Llama-3.2-1B-Instruct	Llama3	1B
meta-llama/Llama-3.2-3B-Instruct	Llama3	3B
meta-llama/Llama-3.1-8B-Instruct	Llama3	8B
meta-llama/Llama-3.1-70B-Instruct	Llama3	70B
deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B	DeepSeek	1.5B
deepseek-ai/DeepSeek-R1-Distill-Qwen-7B	DeepSeek	7B
deepseek-ai/DeepSeek-R1-Distill-Qwen-14B	DeepSeek	14B
deepseek-ai/DeepSeek-R1-Distill-Qwen-32B	DeepSeek	32B

Table 2: HuggingFace information and sizes of real-world datasets.

Dataset Name	HuggingFace Identifier	HuggingFace Split	Size
GSM8K	openai/gsm8k	test	1319
AIME	TianHongZXY/aime-1983-2025	test	963
MMLU (STEM)	cais/mmlu	test	316
MMLU (Social Sciences)	cais/mmlu	test	308
CSQA	tau/commonsense_qa	validation	2442
GPQA	Idavidrein/gpqa	train	198

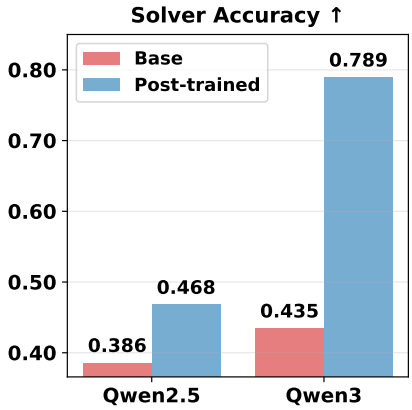

Figure 11: Improvements in solver accuracies of `Qwen2.5-Base` and `Qwen3-Base` models from post-training.