When Does Verification Pay Off? A Closer Look at LLMs as Solution Verifiers

Jack Lu*, Ryan Teehan*, Jinran Jin, and Mengye Ren
Agentic Learning AI Lab, New York University
{yl11330, rst306, jj3007, mengye}@nyu.edu
https://agenticlearning.ai/llm-verification
November 20, 2025

Abstract

Large language models (LLMs) can act as both problem solvers and solution verifiers, with verifiers improving solver performance by selecting high-quality answers from a pool of candidates. However, prior studies of solver–verifier interactions have been limited, focusing mainly on self-verification and rarely examining how verifiers judge outputs from models in their own or in another model family. Modern LLMs also undergo extensive post-training, but its effect on verification remains unclear. 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. We compare self-verification with verification within the same family and across different families. To support this, we introduce and empirically validate *verifier gain*, a metric that predicts the performance improvements from *test-time verifier-based rejection sampling*. We analyze how metrics like verifier gain and false positive rate scale with model size and post-training, and characterize differences in dataset verifiability. Our findings show that cross-family verification is especially effective; post-training reduces self-improvement but strengthens cross-family improvement; and mathematical and logical tasks exhibit the highest inherent verifiability.

1 Introduction

Problem-solving with LLMs has shifted from solely querying a model for a solution to a system where models both solve and verify. The paradigm of verifying solutions at test time is broad, spanning both simple strategies, such as generating multiple candidate solutions and using a verifier as a filter (Zhao et al., 2025), and more complex iterative refinement approaches (Madaan et al., 2023). With generative verification at test time, LLMs can solve more complex problems than they can when used alone (Cobbe et al., 2021; Lightman et al., 2024).

Despite the increasing dominance of this paradigm, 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 itself (self-improvement) (Song et al., 2025), yet self-verification is not guaranteed to be effective: models may be biased toward their own reasoning patterns, their training may reinforce

* Equal contribution.

these tendencies, and different tasks may vary in how much they benefit from verification. Additionally, this focus on self-verification offers little insight into how verification behaves when the solver and verifier differ. With open-source model families that have base and post-trained pairs, size variants, reproducible inference pipelines, and explicit reasoning traces, we can systematically study verification across models. We therefore broaden our analysis to include both intra-family and cross-family verification and ask the following central question:

> *When does verification actually pay off, and how does each factor, such as model family, model size, post-training, solver–verifier similarity, or task type, influence how effective verification is in improving the solver?*

To accomplish this, we evaluate verifiers across a diverse suite of tasks, including synthetic tasks used to test precise logical reasoning or symbolic computation (3-SAT, Sudoku, and matrix multiplication), mathematical reasoning tasks (AIME (Mathematical Association of America, 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 social sciences (Hendrycks et al., 2021)) using 37 models from 7 model families.

Our analysis indicates that self-verification does not always "pay off": models often favor solutions resembling their own reasoning (Section 5.1, 5.3), post-training can sharpen this bias (Section 5.4), and 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 extensively compare performance improvements from self-verification, intra-family verification, and cross-family verification, finding that cross-family verification is often the most beneficial, particularly when compared to self-verification. We link these differences to similarities in the solution distributions of the solver and verifier: verifier gain decreases as these distributions become more similar. 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

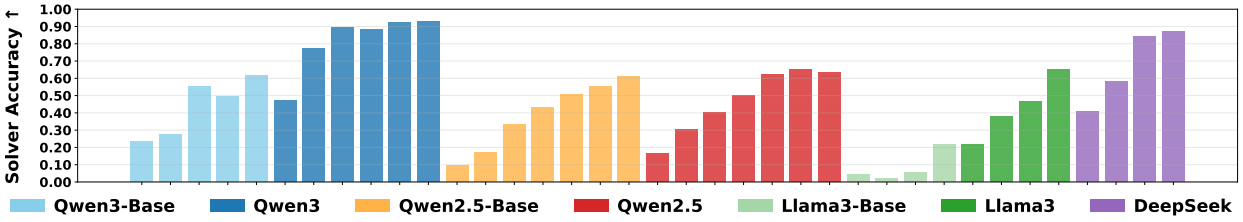

Figure 1: *Average solver accuracy of each model over all datasets. Base model families are suffixed by* ***-Base****. Models within each family are ordered in increasing sizes.*

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). Finally, 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.

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.

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 solutions 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 categorize each solver–verifier pair from our pool of models into one of three verification settings. Each solver–verifier pair is considered an instance of:

1. **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.
2. **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.
3. **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.

Using these three categorizations, we evaluate a verifier by applying it to a set of solver models, computing the corresponding verifier metrics, and then partitioning these metrics according to the three verification settings and averaging within each partition. A formal mathematical description of each verification setting is provided in Appendix C.

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 16 base models

from the `Llama3-Base`, `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. Figure 1's legend displays the seven model families and the color scheme assigned to each.

Datasets. To comprehensively evaluate each model's performance as both a solver and a verifier, we compile a broad suite of real-world and synthetic tasks spanning diverse domains. We include 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. Code for all experiments, including synthetic-data generation, is included in the Supplementary Materials and will be open-sourced upon publication.

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 from 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 Do Better Solvers Make Better Verifiers?

Solver performance. We first 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.

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 2). 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 divide the verifier metrics into three verification settings and average within each setting over solvers and datasets.

Verifier accuracy tends to improve with the verifier's own solver accuracy, but the relationship becomes more nuanced when examining other metrics. The FPR increases during self-verification and intra-family verification but decreases slightly during cross-family verification. This indicates that verifiers with stronger solver abilities are more likely to incorrectly label solutions as correct when verifying their own outputs or outputs from models within their family. We provide additional visualizations for F1-score and precision in Appendix H.

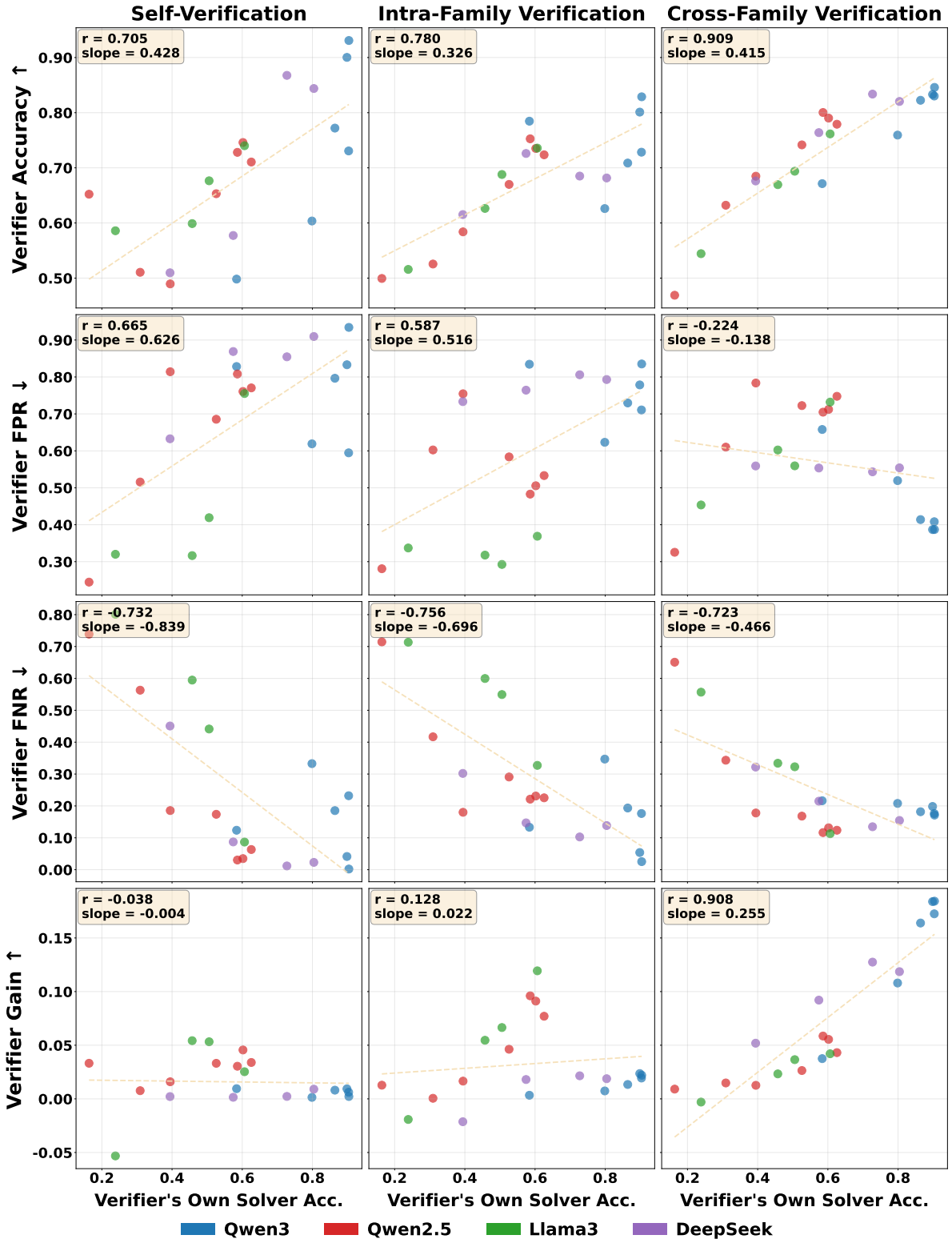

Figure 2: *Correlation between each verifier's metrics (rows) and its own solver accuracy for all 21 post-trained models, averaged over all datasets. Each verifier metric is computed over our three verification settings (columns).*

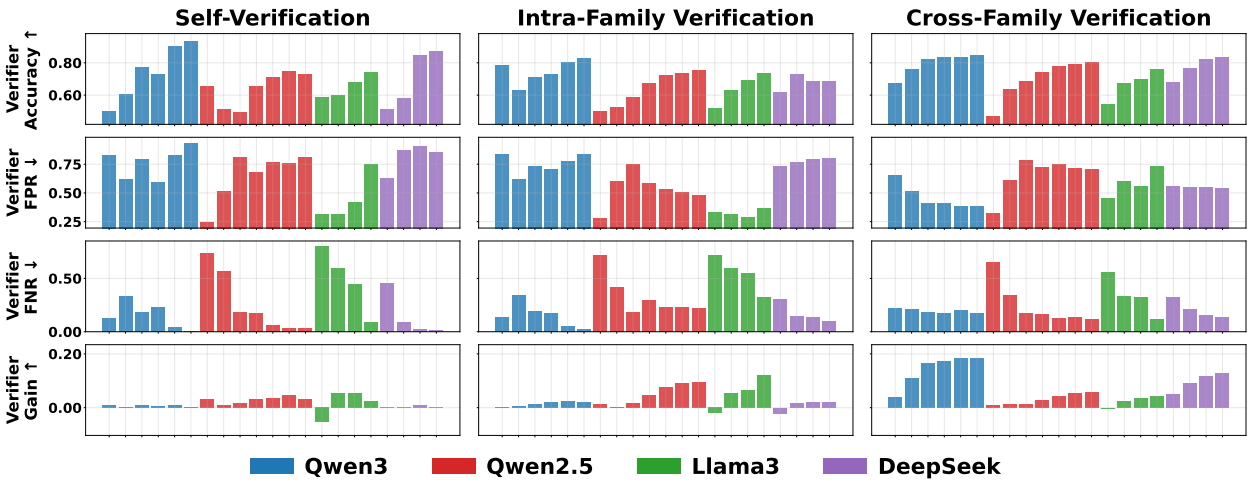

Figure 3: *Correlation between each verifier's metrics (rows) and model size for all 21 post-trained models, averaged over all datasets. In each plot, models are separated by family and ordered by increasing size. Each verifier metric is computed over our three verification settings (columns).*

To better interpret the trends suggested by the accuracy and FPR results, we examine verifier gain in the final row. Verifier gain quantifies the expected benefit of using the verifier during rejection sampling, i.e., repeatedly sampling until the verifier accepts a solution (a common solver–verifier interaction setting (Song et al., 2025)). 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 3, we repeat the experiments from Figure 2 but plot each verifier metric as a function of model size within each model family. We observe that 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. At first glance, 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 2 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 based on their verifier gains, stronger model families like `DeepSeek` and `Qwen3` do not.

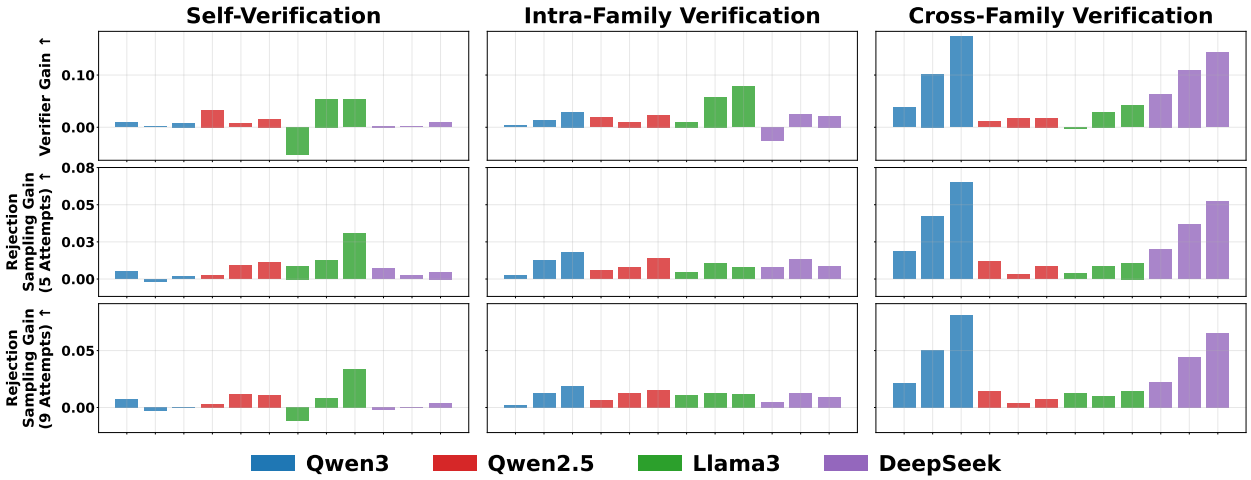

Figure 4: *Comparison between theoretical and empirical verifier gains (rows) for each verification setting (columns). Row 1 shows verifier gains computed from Equation 1. Rows 2 and 3 each show the gains from rejection sampling, computed from rejection sampling using verifiers for up to 5 and 9 solver attempts, respectively.*

5.2 Is Verifier Gain a Good Predictor for Improvements from Resampling?

Our verifier gain metric 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 until the verifier labels one as correct, for 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 4.

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. In particular, we observe smaller gains from self-verification than from both intra-family and cross-family verification, as well as stronger scaling behavior for cross-family verification relative to the other settings.

Takeaways: 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.

5.3 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 the well-documented self-enhancement bias (Krueger, 1998), in which individuals evaluate themselves more favorably than objective evidence would suggest. Our results suggest that an analogous effect may arise in solver–verifier interactions. As shown earlier in Figures 2 and 3, strong reasoning models benefit the least from self-verification and the most from cross-family verification, hinting at a similar bias.

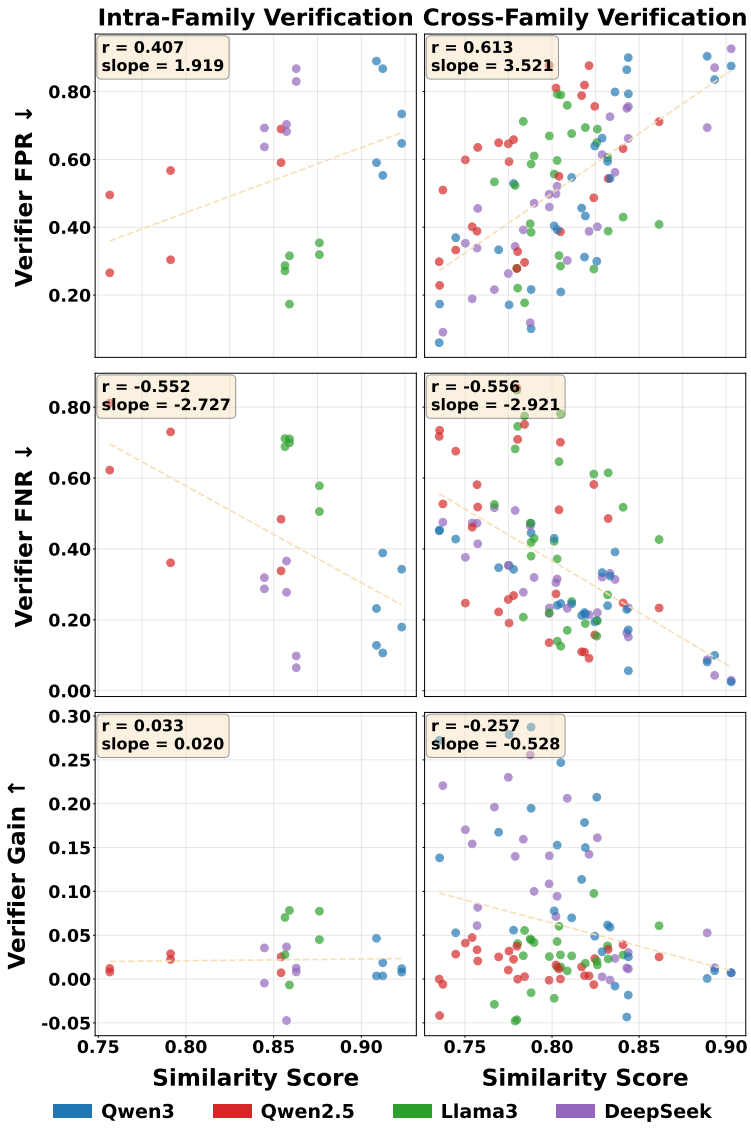

Figure 5: *Correlation between verifier metrics with similarity scores between solver-verifier pairs. Each marker is colored based on the verifier model family.*

To directly investigate this behavior, 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), 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. For each pair, we plot the verifier metric against the **solver–verifier similarity score**, defined as the average cosine similarity between the two models' solution embeddings across all dataset problems. Solutions are embedded using `sentence-transformers/all-mpnet-base-v2`.

Figure 5 shows that, for both intra-family and cross-family settings, FPR exhibits a clear positive trend: 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 with increasing similarity. This indicates that, when selecting a solver–verifier pair, choosing a verifier whose solution distribution differs from that of the solver leads to more reliable verification.

> **Takeaways:** 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. We exclude `Llama3-Base` due to its weak solver and verifier performance and omit `DeepSeek` because matching base models are unavailable. Verification metrics are computed across all 37 base and post-trained models.

Post-training 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` shows a striking 35.4% gain. Full results are provided in Appendix I.

Post-training 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 `Qwen2.5` and `Qwen3`, post-training increases FPR and reduces verifier gain in self-verification, 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.

> **Takeaways:** 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.

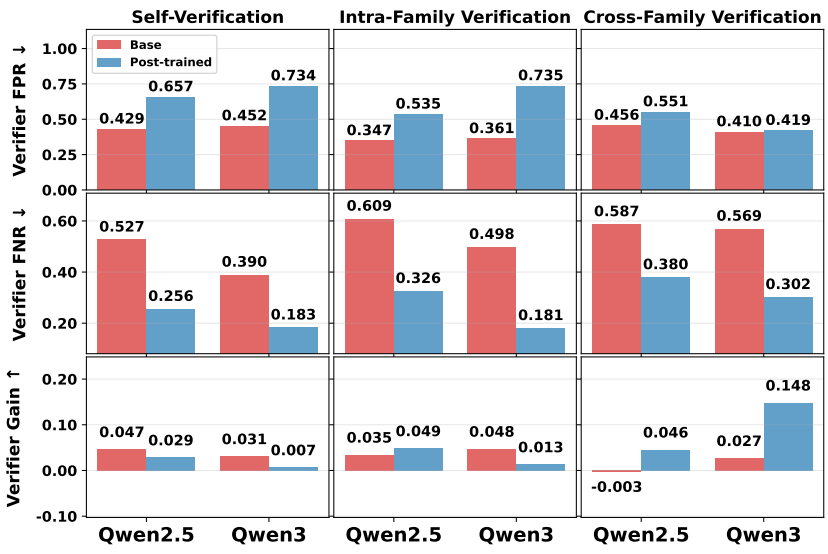

Figure 6: *Changes in verifier metrics of the* `Qwen2.5-Base` *and* `Qwen3-Base` *models from post-training.*

5.5 Which Datasets are Easy to Verify?

Thus far, we have examined verifier performance and its contribution to solver accuracy through rejection sampling. 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 5.1, average them across all verifier models, and plot them against solver accuracies.

We find that verification accuracy correlates strongly with solver accuracy. In contrast, verifier gains from self-verification do not correlate with problem difficulty, whereas both intra-family and cross-family verifier gains exhibit clear positive correlations. 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), indicating that verification difficulty cannot be explained solely by solver accuracy. This leads to our next question: *are some tasks inherently easier to verify than others?* Relatedly, Song et al. (2025) find that models cannot self-improve on tasks requiring factual recall, but can self-improve on Sudoku with sufficient pretraining scale. We observe a similar separation in Figure 7: 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

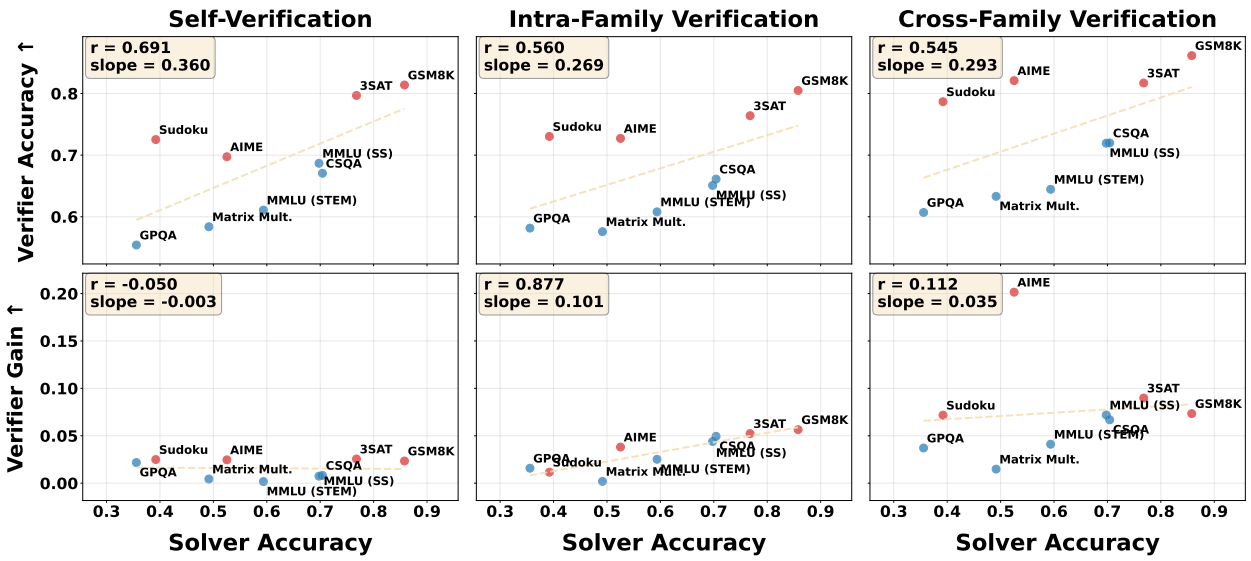

Figure 7: *Correlation of verifier metrics (rows) with solver accuracies, averaged over solver-verifier pairs that belong to each verification setting (columns).*

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 content. For these latter tasks, verifying an answer requires essentially the same knowledge as solving the problem.

> **Takeaways:** Although tasks that are easy to solve are typically easier to verify, some tasks are inherently easier to verify. These include synthetic problems with logical or structured reasoning (e.g., 3SAT, Sudoku) and real-world tasks relying primarily on mathematical reasoning rather than extensive factual recall (e.g., GSM8K, AIME). Such tasks also yield larger gains from test-time rejection sampling with verifiers.

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 expected improvement obtained by using a verifier for test-time rejection sampling, motivating the introduction of *verifier gain*, a more informative measure that captures this expected improvement. Using this metric, our analysis shows that verifier gain is often lower for self-verification and intra-family verification than for cross-family verification, particularly as model size increases or post-training is applied. Further analysis reveals that decreases in verifier gain correlate with greater similarity between the solver's and verifier's solution distributions. 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.

Limitations and future work. 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.

Acknowledgement

We thank members of the NYU Agentic Learning AI Lab for their helpful discussions. JL is supported by the NSERC PGS-D Scholarship. The work is supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) under grant RS-2024-00469482, funded by the Ministry of Science and ICT (MSIT) of the Republic of Korea in connection with the Global AI Frontier Lab International Collaborative Research. The compute is supported by the NYU High Performance Computing resources, services, and staff expertise. We also thank Modal for providing additional compute resources.

References

Brown, B., Juravsky, J., Ehrlich, R., Clark, R., Le, Q. V., Ré, C., and Mirhoseini, A. (2024). Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Chen, J., Ren, J., Chen, X., Yang, C., Sun, R., Yoon, J., and Arık, S. Ö. (2025). Sets: Leveraging self-verification and self-correction for improved test-time scaling. *arXiv preprint arXiv:2501.19306*.

Chen, X., Aksitov, R., Alon, U., Ren, J., Xiao, K., Yin, P., Prakash, S., Sutton, C., Wang, X., and Zhou, D. (2024). Universal self-consistency for large language models. In *ICML 2024 Workshop on In-Context Learning*.

Chow, Y., Tennenholtz, G., Gur, I., Zhuang, V., Dai, B., Kumar, A., Agarwal, R., Thiagarajan, S., Boutilier, C., and Faust, A. (2025). Inference-aware fine-tuning for best-of-n sampling in large language models. In *ICLR*.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv: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.

(2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dhuliawala, S., Komeili, M., Xu, J., Raileanu, R., Li, X., Celikyilmaz, A., and Weston, J. (2024). Chain-of-verification reduces hallucination in large language models. In *ACL Findings*.

Gou, Z., Shao, Z., Gong, Y., yelong shen, Yang, Y., Duan, N., and Chen, W. (2024). CRITIC: Large language models can self-correct with tool-interactive critiquing. In *ICLR*.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Guzmán, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhotia, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Zhang, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Albiero, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Wang, X., Tan, X. E., Xia, X., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Srivastava, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Teo, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Dong, A., Franco, A., Goyal, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Liu, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Gao, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Le, E.-T., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Kokkinos, F., Ozgenel, F., Caggioni, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Inan, H., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Zhan, H., Damlaj, I., Molybog, I., Tufanov, I., Leontiadis, I., Veliche,

I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Lam, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Jagadeesh, K., Huang, K., Chawla, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Liu, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Mehta, N., Laptev, N. P., Dong, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Parthasarathy, R., Li, R., Hogan, R., Battey, R., Wang, R., Howes, R., Rinott, R., Mehta, S., Siby, S., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Mahajan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Patil, S., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Deng, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Koehler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wu, X., Wang, X., Wu, X., Gao, X., Kleinman, Y., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Zhao, Y., Hao, Y., Qian, Y., Li, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., Zhao, Z., and Ma, Z. (2024). The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. (2021). Measuring massive multitask language understanding. In *ICLR*.

Hosseini, A., Yuan, X., Malkin, N., Courville, A., Sordoni, A., and Agarwal, R. (2024). V-STar: Training verifiers for self-taught reasoners. In *COLM*.

Huang, A., Block, A., Foster, D. J., Rohatgi, D., Zhang, C., Simchowitz, M., Ash, J. T., and Krishnamurthy, A. (2024a). Self-improvement in language models: The sharpening mechanism. *arXiv preprint arXiv:2412.01951*.

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., and Zhou, D. (2024b). Large language models cannot self-correct reasoning yet. In *ICLR*.

Jiang, W., Shi, H., Yu, L., Liu, Z., Zhang, Y., Li, Z., and Kwok, J. (2024). Forward-backward reasoning in large language models for mathematical verification. In *ACL Findings*.

Kamoi, R., Zhang, Y., Zhang, N., Han, J., and Zhang, R. (2024). When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs.

Kang, J., Li, X. Z., Chen, X., Kazemi, A., Sun, Q., Chen, B., Li, D., He, X., He, Q., Wen, F., et al. (2024). Mindstar: Enhancing math reasoning in pre-trained llms at inference time. *arXiv preprint arXiv:2405.16265*.

Khalifa, M., Logeswaran, L., Lee, M., Lee, H., and Wang, L. (2023). GRACE: Discriminator-guided chain-of-thought reasoning. In *EMNLP Findings*.

Kim, G., Baldi, P., and McAleer, S. (2023). Language models can solve computer tasks. In *NeurIPS*.

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

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. (2024). Let's verify step by step. In *ICLR*.

Ling, Z., Fang, Y., Li, X., Huang, Z., Lee, M., Memisevic, R., and Su, H. (2023). Deductive verification of chain-of-thought reasoning. In *NeurIPS*.

Luo, L., Liu, Y., Liu, R., Phatale, S., Guo, M., Lara, H., Li, Y., Shu, L., Meng, L., Sun, J., and Rastogi, A. (2025). Improve mathematical reasoning in language models with automated process supervision. *arXiv preprint arXiv:2406.06592*.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. (2023). Self-refine: Iterative refinement with self-feedback.

Mathematical Association of America (2025). 1983-2025 American Invitational Mathematics Examination (AIME) I: Problems and Solutions. Art of Problem Solving Wiki entry.

Nichols, E., Gao, L., and Gomez, R. (2020). Collaborative storytelling with large-scale neural language models. In *ACM SIGGRAPH Conference on Motion, Interaction and Games*.

Olausson, T. X., Inala, J. P., Wang, C., Gao, J., and Solar-Lezama, A. (2023). Is self-repair a silver bullet for code generation? *arXiv preprint arXiv:2306.09896*.

Panickssery, A., Bowman, S. R., and Feng, S. (2024). Llm evaluators recognize and favor their own generations.

Qwen, :, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Tang, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. (2024). Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. (2024). GPQA: A graduate-level google-proof q&a benchmark. In *COLM*.

Roy, S. and Roth, D. (2015). Solving general arithmetic word problems. In *EMNLP*.

Sareen, K., Moss, M. M., Sordoni, A., Agarwal, R., and Hosseini, A. (2025). Putting the value back in rl: Better test-time scaling by unifying llm reasoners with verifiers. *arXiv preprint arXiv:2505.04842*.

Shen, J., Yin, Y., Li, L., Shang, L., Jiang, X., Zhang, M., and Liu, Q. (2021). Generate & rank: A multi-task framework for math word problems. In *EMNLP*.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. R., and Yao, S. (2023). Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*.

Singhi, N., Bansal, H., Hosseini, A., Grover, A., Chang, K.-W., Rohrbach, M., and Rohrbach, A. (2025). When to solve, when to verify: Compute-optimal problem solving and generative verification for llm reasoning. *arXiv preprint arXiv:2504.01005*.

Snell, C. V., Lee, J., Xu, K., and Kumar, A. (2025). Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *ICLR*.

Song, Y., Zhang, H., Eisenach, C., Kakade, S. M., Foster, D., and Ghai, U. (2025). Mind the gap: Examining the self-improvement capabilities of large language models. In *ICLR*.

Stroebl, B., Kapoor, S., and Narayanan, A. (2024). Inference scaling flaws: The limits of llm resampling with imperfect verifiers. *arXiv preprint arXiv:2411.17501*.

Talmor, A., Herzig, J., Lourie, N., and Berant, J. (2019). Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *NAACL 2019*.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. (2023). Self-consistency improves chain of thought reasoning in language models. In *ICLR*.

Welleck, S., Lu, X., West, P., Brahman, F., Shen, T., Khashabi, D., and Choi, Y. (2023). Generating sequences by learning to self-correct. In *ICLR*.

Weng, Y., Zhu, M., Xia, F., Li, B., He, S., Liu, S., Sun, B., Liu, K., and Zhao, J. (2023). Large language models are better reasoners with self-verification. In *EMNLP Findings*.

Wu, Z., Zeng, Q., Zhang, Z., Tan, Z., Shen, C., and Jiang, M. (2024). Large language models can self-correct with key condition verification. In *EMNLP*.

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. (2025). Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yang, K., Deng, J., and Chen, D. (2022). Generating natural language proofs with verifier-guided search. In *EMNLP*.

Yu, F., Gao, A., and Wang, B. (2024). OVM, outcome-supervised value models for planning in mathematical reasoning. In *NAACL Findings*.

Yu, F., Li, Y., and Wang, B. (2025). Scaling flaws of verifier-guided search in mathematical reasoning. *arXiv preprint arXiv:2502.00271*.

Zhang, L., Hosseini, A., Bansal, H., Kazemi, M., Kumar, A., and Agarwal, R. (2025). Generative verifiers: Reward modeling as next-token prediction. In *ICLR*.

Zhang, Y., Khalifa, M., Logeswaran, L., Kim, J., Lee, M., Lee, H., and Wang, L. (2024). Small language models need strong verifiers to self-correct reasoning. In *ACL*.

Zhao, E., Awasthi, P., and Gollapudi, S. (2025). Sample, scrutinize and scale: Effective inference-time search by scaling verification. In *ICML*.

Zhou, A., Wang, K., Lu, Z., Shi, W., Luo, S., Qin, Z., Lu, S., Jia, A., Song, L., Zhan, M., and Li, H. (2024). Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. In *ICLR*.

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 and 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.

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

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:

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

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.

G Solver Accuracy by Dataset

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

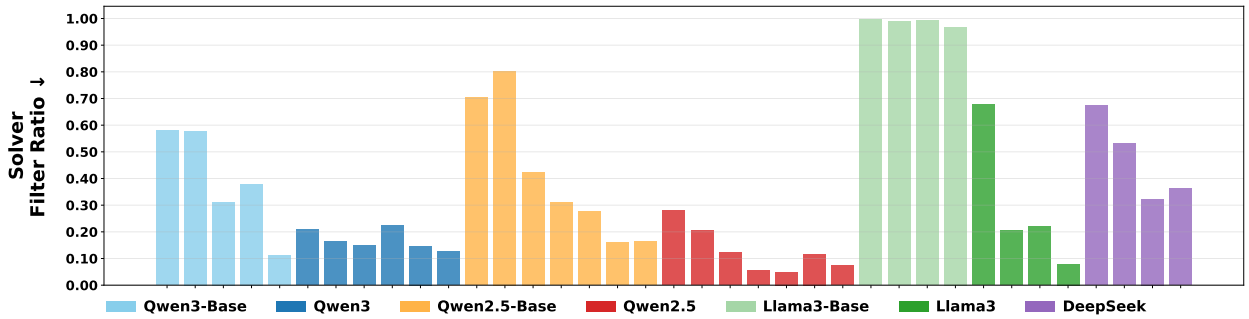

Figure 8: *Average ratio of filtered solver outputs for each model over all datasets. Base model families are suffixed by* ***-Base****. Models within each family are ordered in increasing size.*

H F1-Score and Precision Visualization

Figure 2 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 2 has a stronger negative impact on lowering F1-score than accuracy.

While Section 5.1 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.

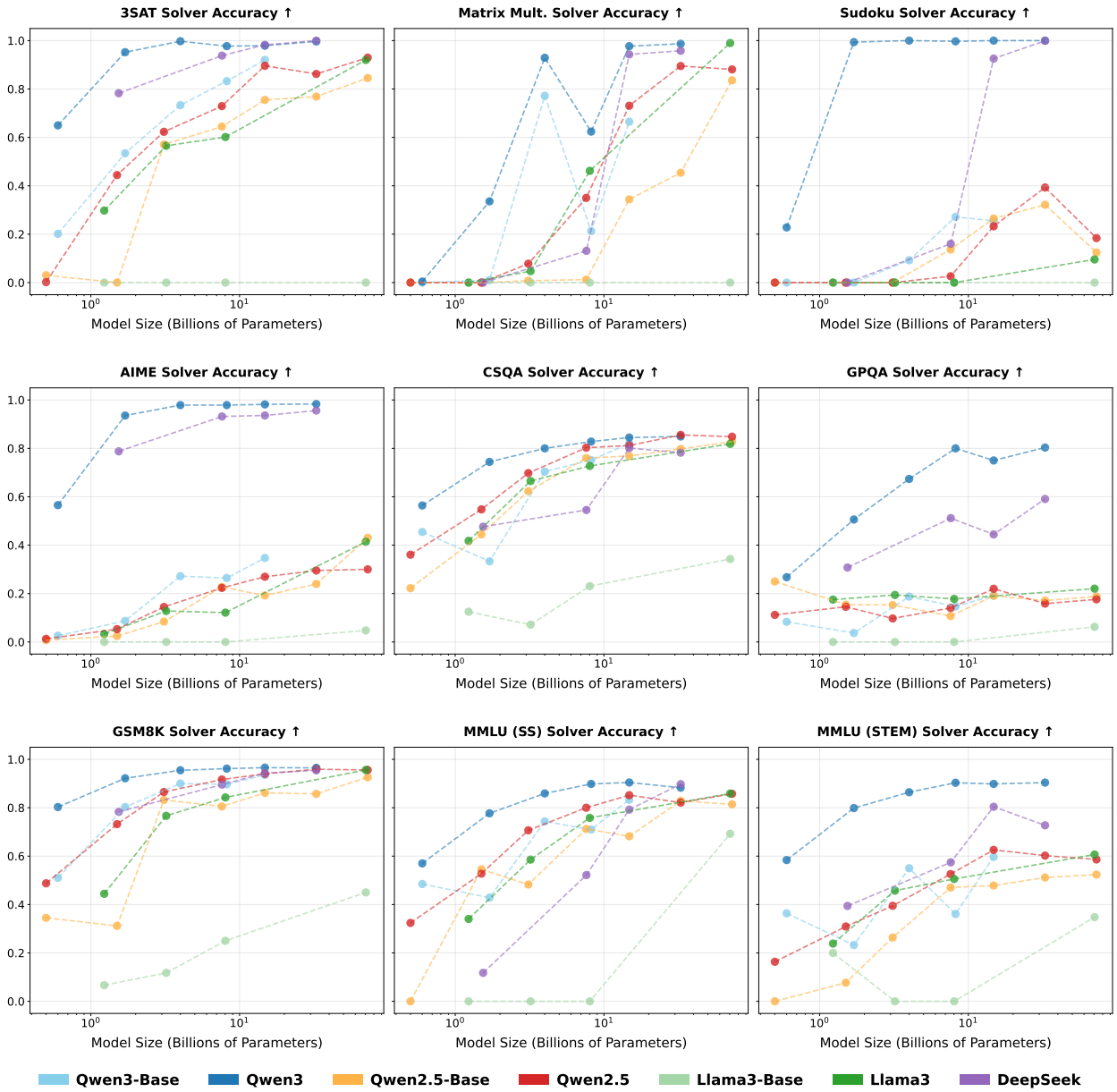

Figure 9: *The solver accuracies of 37 models on each dataset.*

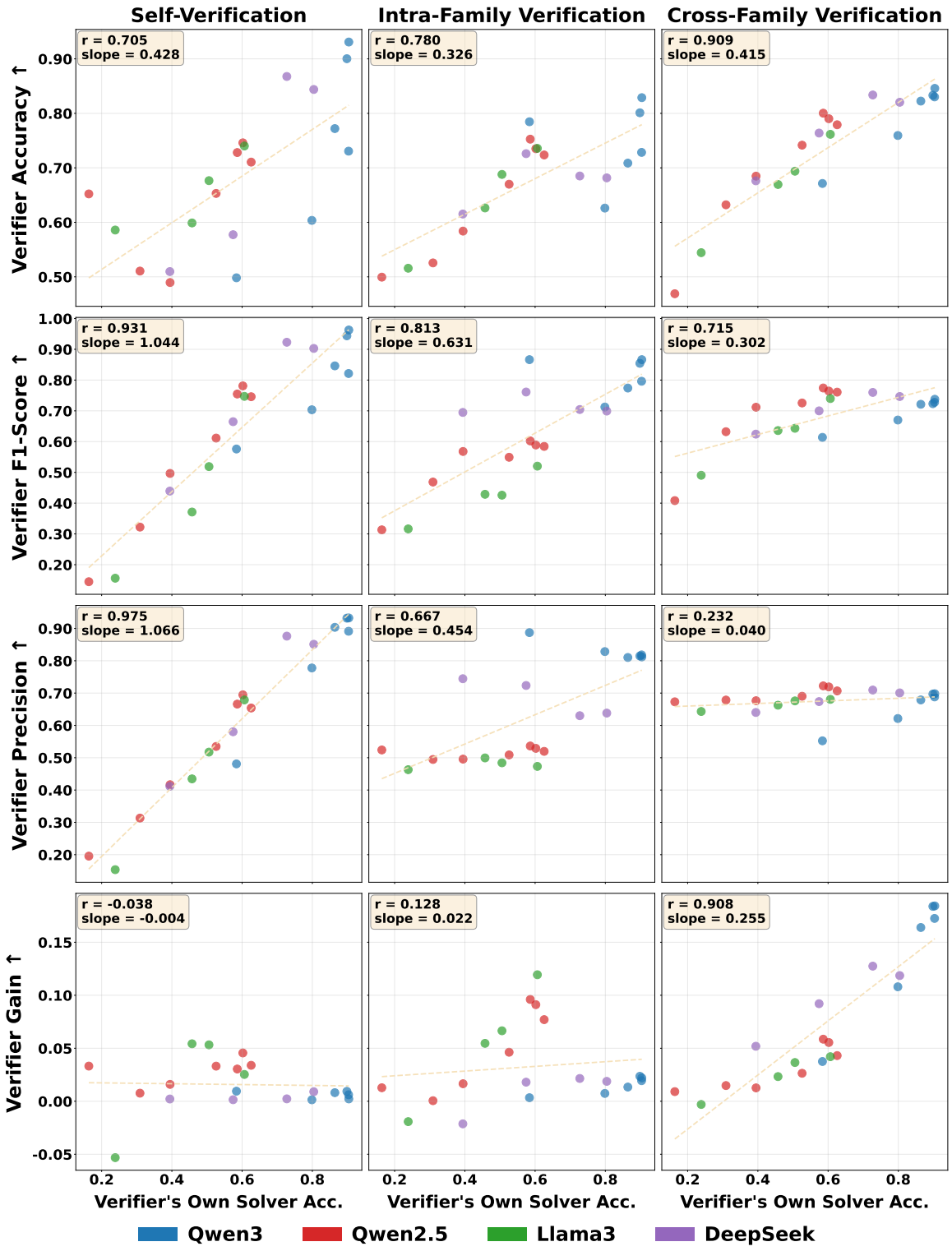

Figure 10: *Correlation between each model's verifier metrics (rows) and its own solver accuracy for all 21 post-trained models, averaged over all datasets. Each verifier metric is computed over three settings (columns): self-verification, intra-family verification, and cross-family verification. We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

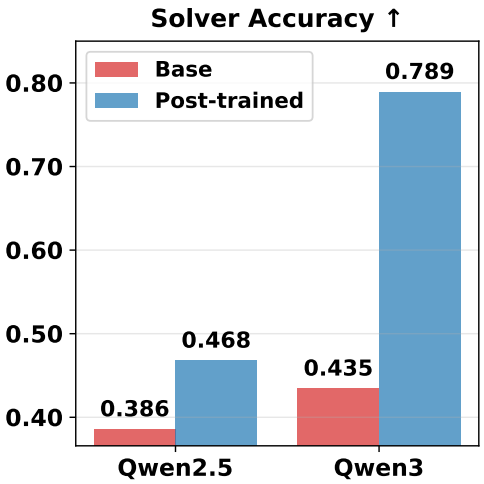

Figure 11: *Improvements in solver accuracies of* `Qwen2.5-Base` *and* `Qwen3-Base` *models from post-training.*