Variation in Verification: Understanding Verification Dynamics in Large Language Models

Yefan Zhou[1,2*], **Austin Xu**[2], **Yilun Zhou**[2], **Janvijay Singh**[2], **Jiang Gui**[1], **Shafiq Joty**[2]
[1]Dartmouth College, [2]Salesforce AI Research

Abstract

Recent advances in large language models (LLMs) have produced increasingly capable generators that can solve complex problems across diverse domains. Evaluating these generators' outputs has shifted from human assessment to automated verification using LLMs as verifiers. In this paradigm, verifier models assess the correctness of solutions produced by generator models, a framework now central to applications such as test-time scaling (TTS). In this study, we study generative verifiers, which perform verification as a next-token prediction task by generating chain-of-thought (CoT) reasoning followed by a binary verdict. We systematically analyze verification dynamics across three dimensions: problem difficulty, generator capability, and verifier generation capability, conducting empirical studies on 2.3k mathematical problems using 14 open-source models (2B to 72B parameter range) and GPT-4o. Our experiments reveal three key findings about verification effectiveness: (1) Problem difficulty affects recognizing correct responses; (2) Weak generators produce errors that are easier to detect than strong generators; (3) Verification ability is generally correlated with verifier generation ability, but correlation relationship varies with problem difficulty. These findings enable cost-effective strategies in TTS applications. Specifically, we identify two patterns that weak models can substitute for strong ones. First, given the same verifier, weak generators can nearly match stronger generators in post-verification TTS performance (e.g., a 9B model matches a 27B model). Second, weak verifiers can approximate strong verifiers in regimes where both achieve similar verification performance.

1 Introduction

Large language models (LLMs) have advanced rapidly in solving reasoning tasks such as mathematics and code generation [16, 42], yet their outputs remain unreliable, often containing subtle or confident mistakes [5, 16, 22, 39]. LLM verification [1, 12, 13, 22] has emerged as a central mechanism for identify such errors in a scalable manner. Recent work has increasingly focused on *generative verifiers* [24, 25, 47], which frame verification as next-token prediction: the model typically generates a chain-of-thought (CoT) reasoning trace and then outputs a binary verdict token. This approach has been shown to improve over earlier discriminative verifiers or reward models that assign scalar scores to candidate solutions [22], as it better leverages the inherent text-generation capabilities of LLMs. One valuable downstream application of automatic verification is test-time scaling (TTS), where additional inference-time compute is allocated to improve generation performance. A popular paradigm of TTS is the use of a verifier model to evaluate candidate responses, filter errors, and identify correct solutions. This approach underlies techniques such as rejection sampling [2], re-ranking [48], weighted majority voting [35, 37], and step-level generation [30].

Current practice often deploys strong verifiers, typically closed-source frontier models [14]. This practice rests on the assumption that verification quality scales with a verifier's generation capa-

*Contact: yefan.zhou@dartmouth.edu. The full version of this work can be found on arXiv.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Foundations of Reasoning in Language Models.

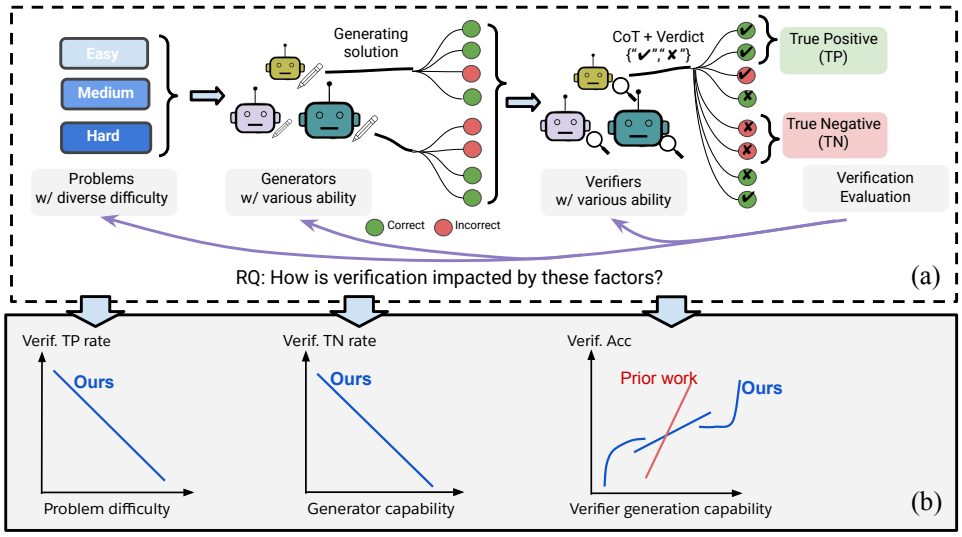


Figure 1: **Overview of our study on verification dynamics.** (a) We consider generative verification: an LLM generator produces a solution to a reasoning problem, and an LLM verifier conditions on the problem and solution to generate a verification CoT followed by a binary verdict ("Correct"/"Incorrect"). We design controlled experiments that vary problem difficulty, generator's generation capability, and verifier's generation capability, investigating how each of these factors influence verification performance. (b) Our analysis reveals three patterns: problem difficulty governs recognition of correct responses (true positives); generator generation capability determines error detectability (true negatives); and verifier generation capability correlates with performance in a difficulty-dependent manner, revealing non-linear regimes left uncovered by prior work [3].

bility [3, 18, 32]. However, verifying a solution is often easier than generating one from scratch, a phenomenon referred to as "verification asymmetry" [38]. For instance, solving a mathematical problem requires creativity to devise a right approach, whereas verification amounts to confirming the correctness of each deduction. This asymmetry appears in a number of fields. In convex optimization, dual certificates enable efficient validation of optimality of a proposed solution, while in cryptography, primality tests check if a number is prime without discovering new primes. These observations also suggest that the capabilities required for verification may not be fully captured by generation ability. Thus, studying verification as a distinct capability, rather than as a simple by-product of generation strength, warrants investigation.

Despite extensive research on generation dynamics and the factors that influence generation quality [7, 28, 39], the dynamics of verification remain largely unexplored. In particular, there is a lack understanding of how characteristics of problems and generators' responses, and model capabilities interact to determine verification effectiveness. Without understanding verification dynamics, one can risk misallocating computational resources by defaulting to expensive frontier models when simpler alternatives might suffice. This gap in understanding motivates our central research question:

What factors influence verification success?

In this paper, we present a systematic study of generative verification across three critical dimensions: problem difficulty, generation capability of generators, and generation capability of verifiers. Figure 1 provides an overview of our study. We quantify verification performance by measuring the probability of the verifier recognizing both correct and incorrect generated solutions in controlled experimental settings. Concretely, we focus on verifiable problems with objective ground-truth answers, e.g., mathematical reasoning. This choice allows us to rigorously validate both the verifier and the generator performance, while also simulating the reference-free evaluation settings in which verifiers are typically deployed in practice. While our experiments use math as a testbed, we believe the insights should extend to any domain where correctness can be reliably defined and checked.

Main Findings. Our study reveals that verifier generation capability is not the sole determinant of verification success, with *problem difficulty* and *generator capability* playing critical roles in different aspects, which are shown in Figure 1. Specifically:

- Problem difficulty primarily governs recognition of correct solutions;
- Generator capability influences error detection; i.e, errors made by weak generators are easier to detect than those by strong generators;
- Verifier's generation capability correlates with verification performance in a problem difficulty-dependent manner: saturated (or uncorrelated) for easy problems, linear for medium problems, and threshold-limited for hard problems.

Our empirical analysis includes 2347 problems from eight mathematical reasoning benchmarks, evaluated across 14 open-source models (2B to 72B parameters) and closed-source models.

Application to TTS. We demonstrate the practical implications of our findings for TTS. First, given the same verifier, the TTS performance of a weak generator can match the performance of a strong generator. For instance, Gemma2-9B achieves comparable performance to Gemma2-27B given a fixed verifier. Second, we identify specific regimes where weak verifiers can substitute for strong ones: with strong generators, on problems at both ends of the difficulty spectrum.

2 Related Work

LLM Verification. The deployment of LLMs as verifiers/judges has emerged as a central mechanism for scalable evaluation, with recent efforts focusing on training more effective small judge models through fine-tuning [36, 40, 41, 46]. Beyond single-model judges, recent work explores self-verification and multi-agent verification, where models reflect on or critique their own outputs [27] or collaborate in debate-style or hierarchical setups to improve reliability [17]. Several studies analyze the correlation between judge capability and evaluation performance. Krumdick et al. [18] demonstrate limitations without human grounding, while Tan et al. [32] provide benchmarks showing judge effectiveness, and Chen et al. [3] reveal inherent self-preference biases. Unlike these studies that conclude that verification quality primarily depends on model generation capability, we identified other factors that play equally important roles in determining verification performance.

Verification for Test-time Scaling. Recent studies investigate verification in TTS and explore cost-effectiveness strategies. Zhou et al. [48] introduce the JETTS benchmark, evaluating LLM-as-judges across several TTS methods. Angelopoulos et al. [1] develop policies to balance weak and strong raters for statistical efficiency. Stroebl et al. [31] analyze the fundamental limits of resampling with imperfect verifiers, showing diminishing returns as verifier quality decreases. Singhi et al. [29] study the trade-off between solving problems directly versus using verification, proposing compute-optimal strategies. Saad-Falcon et al. [26] propose Weaver, which aggregates weak verifiers with weak supervision to approach strong-verifier performance. While these studies focus on optimizing verification given fixed model qualities, we studied when weak model can approach strong ones based on problem and generator characteristics.

3 Experimental Setup

3.1 Preliminaries

Problem and Response Space. Let x denote a problem with ground-truth answer $y^*(x)$. A model response r to x consists of a CoT solution and a final answer $a(r)$, and we consider the response correct if $a(r) = y^*(x)$. As discussed in Section 1, our study uses verifiable problems with objective answers, allowing us to rigorously evaluate verifier outputs against ground-truth while simulating reference-free evaluation settings.

Generator and Verifier. A generator G maps a problem x to a distribution over responses, denoted $r \sim G(\cdot|x)$. A verifier V takes a problem–response pair (x, r) and outputs a judgment of correctness. In the binary case, $V(x, r) \in \{0, 1\}$, where 1 indicates acceptance and 0 indicates rejection. More generally, a generative verifier produces a verification CoT explaining its reasoning, followed by an explicit verdict such as "Correct" or "Incorrect."

Generation Capability. We measure the generation ability of a model using its *pass rate*. For a generator G and problem x,

$$p_G(x) = \Pr[a(r) = y^*(x) \mid r \sim G(\cdot|x)], \quad p_G(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} p_G(x)$$

Here $p_G(x)$ is the pass rate on a single problem, i.e., the probability that G solves x correctly on one sampled attempt. $p_G(\mathcal{D})$ is the pass rate aggregated over a dataset $\mathcal{D}$, which we use as the overall measure of a model's generation capability. It denotes the average probability that

the generator produces a correct solution across the dataset. Specifically, drawing K responses $\{r_1, \ldots, r_K\} \sim G(\cdot \mid x)$, we estimate

$$\hat{p}_G(x; K) = \frac{1}{K} \sum_{i=1}^{K} \mathbb{1}\big(a\,(r_i) = y^\star(x)\big), \quad \hat{p}_G(\mathcal{D}; K) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \hat{p}_G(x; K)$$

where $\mathbb{1}(\cdot)$ is the Indicator function. The empirical estimate $\hat{p}_G$ is equivalent to the unbiased pass@1 estimator used in the previous work [45], which adopts it as a measure of the generation capacity. We also measure a verifier's generation capability using the same metric by prompting it as a generator.

In practice, we sample 64 responses per model-problem pair. We use temperature 0.7 and top-p 1.0 as default sampling hyperparameters, and adopt recommended settings when available (e.g., Qwen3 [44]). Ground-truth correctness is established with `Math-Verify` [19], supplemented by fallback methods to reduce false negatives (details in Section A.2). These 64 responses per problem–model pair are used to estimate generation capability and problem difficulty.

Problem Difficulty. We define the difficulty of a problem as the average pass rate across a set of diverse generators $\mathcal{G}$, $d(x) = \frac{1}{|\mathcal{G}|} \sum_{G \in \mathcal{G}} \hat{p}_G(x)$. This score reflects how broadly solvable a problem is: if most generators succeed, $d(x)$ is high (easy problem), while if few succeed, $d(x)$ is low (hard problem). It provides a model-agnostic way to partition problems by difficulty, extending prior work [30], which measured difficulty relative to a single generator.

Verification Metrics and Evaluation. We evaluate verifiers using the following metrics. The true positive rate (TPR) is the probability of accepting a correct response, while the true negative rate (TNR) is the probability of rejecting an incorrect one:

$$\mathrm{TPR} = \Pr[V(x, r) = 1 \mid a(r) = y^*(x)], \quad \mathrm{TNR} = \Pr[V(x, r) = 0 \mid a(r) \neq y^*(x)].$$

We also report balanced accuracy, $\mathrm{Acc}_{\mathrm{bal}} = \frac{1}{2}(\mathrm{TPR} + \mathrm{TNR})$, which accounts for class imbalance. For verification evaluation, we subsample 8 responses from each 64-sample pool, balanced with 4 correct and 4 incorrect when possible. For very hard problems with fewer than 4 correct responses, we keep all correct ones and sample incorrect ones to reach 8 total (and vice versa for easy problems). Each verifier evaluates responses from all 15 models over the full test set using greedy decoding, unless a controlled subset is specified. The prompt template is given in Section C.

Verification-Augmented Test-time Scaling. We consider the TTS setting of sampling multiple responses from the generator and filtering with a verifier before evaluation. For each problem $x \in \mathcal{D}$, we sample K responses from the generator using a fixed temperature, with $K = 64$ in our experiments. Without verification, TTS performance is measured as $\hat{p}_G(\mathcal{D}; K)$, the empirical pass rate defined above. With verificaton, the verifier V evaluates each candidate, and only responses that are deemed "Correct" are retained. The performance of verification-augmented TTS is measured as

$$\hat{p}_{G,V}(\mathcal{D}; K) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \left(\frac{1}{K'} \sum_{i=1}^{K} \mathbb{1}\big(a(r_i) = y^\star(x),\, V(x, r_i) = 1\big) \right)$$

$$\text{where } K' = \sum_{i=1}^{K} \mathbb{1}\big(V(x, r_i) = 1\big).$$

which is interpreted as the conditional pass rate, i.e., the fraction of correct responses among those retained by the verifier. A corner case arises when the verifier rejects all responses ($K' = 0$); in this case, we set the metric to the generator's pass rate $\hat{p}_G(\mathcal{D})$, so evaluation reverts to randomly selecting from the original K responses, as in the non-verified setting. Further details appear in Section A.1. We define the *verification gain* as the difference relative to the performance without verification: $\Delta\hat{p} = \hat{p}_{G,V}(\mathcal{D}) - \hat{p}_G(\mathcal{D})$, which quantifies how much gain can be attributed to verification. Note that our formulation of TTS differs from the common setting where a single "best" response (e.g., by weighted majority vote) is selected and then evaluated. Instead, we report the empirical pass rate of the verifier-retained pool, which can be interpreted as the expected accuracy of uniformly sampling one response from that pool. This expectation-based view captures the average quality of verifier-retained responses without tying performance to a specific selection strategy.

3.2 Tasks and Models

Mathematical reasoning. We collect a total of 2347 problems from the test sets of eight mathematical reasoning benchmarks: GSM8K [4], MATH500 [11], OlympiadBench [10], AIME24/25 [21],

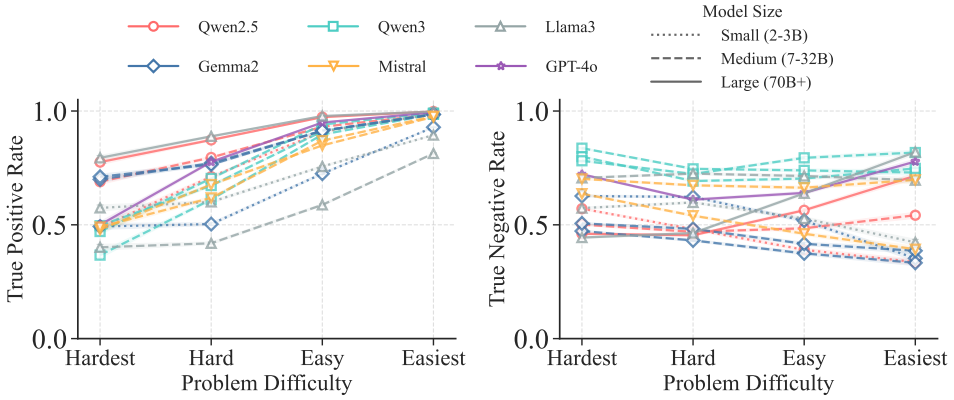
AMC23 [21], Minerva-Math [20], and BBEH Multi-step Arithmetic [15]. We use the entire test sets of these benchmarks, except for GSM8K, which we subsample from 1319 to 600 problems to balance difficulty distribution and reduce the proportion of easy problems.

Models. We construct our model collection by including 14 models from four open-source families: (1) Qwen2.5 at 3B, 7B, and 72B [34]; Qwen3 at 4B, 8B, and 32B [44]; (2) Llama-3.2 at 3B, Llama-3.1 at 8B, and Llama-3.3 at 70B [9]; (3) Gemma-2 at 2B, 9B, and 27B [33]; (4) Ministral 8B and Mistral-Small-24B; and one closed-source model GPT-4o [14]. All models are instruction-tuned versions by default. These models are prompted as generators to produce solutions to problems, and then prompted as verifiers to evaluate the responses. Our prompts are provided in Section C.

4 Experimental results

Our experiments focus on how problem difficulty and generator and verifier generation capability influence verification performance. We present the three research questions and main findings below.

- **RQ1: How does problem difficulty affect verification?** (Section 4.1) As problems become easier, TPR increases steadily, meaning verifiers becoming better at accepting correct responses However, TNR shows no predictable relationship with problem difficulty. This indicates that problem difficulty primarily influences correctness recognition but not error detection.
- **RQ2: How does the generator's generation capability influence verification?** (Section 4.2) As generators become stronger, TNR decreases substantially, while TPR increases only slightly. This indicates that generator capability primarily determines how detectable errors are, with stronger generators producing errors that are harder for verifiers to identify.
- **RQ3: How does verifier generation capability impact verification?** (Section 4.3) Verifier generation ability and verification performance are generally positively correlated, i.e., problems that a verifier can solve reliably are also easier to verify. However, the form of correlation depends heavily on the problem difficulty: a linear correlation occurs only for medium-difficulty problems.



(a) TPR (b) TNR

Figure 2: **Problem difficulty primarily shifts TPR while leaving TNR largely unchanged.** Each curve shows verifier performance across four difficulty groups, with the x-axis indicating problem difficulty and the y-axis reporting TPR (a) and TNR (b). Colors denote model families, and line styles indicate model size (small, medium, large).

4.1 How Does Problem Difficulty Affect Verification?

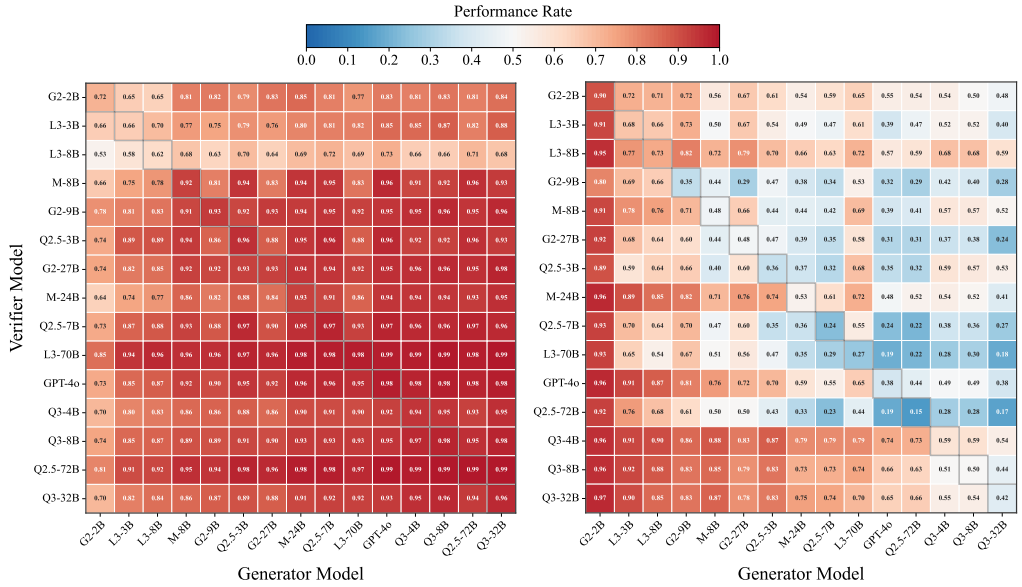
To examine how problem difficulty influences verification, we partition problems into four quartiles by their difficulty score $d(x)$. The lowest quartile ("Hardest") contains the 25% of problems with the smallest $d(x)$ values, while the highest quartile ("Easiest") contains the 25% with the largest values, with the middle two quartiles representing intermediate difficulty.

Problem difficulty primarily influences the verifier's ability to recognize correct responses. Our analysis is conducted at two levels of granularity: response level and problem level. Both analyses below reveal that problem difficulty mainly shapes the verifier's sensitivity to correct responses, while being uncorrelated with identifying incorrect responses.

At the response level, we compute the TPR and TNR of all responses within each quartile. As shown in Figure 2, TPR increases steadily as problems become easier, while TNR shows no clear trend.

This pattern is consistent across model families. At the problem level, we pool all responses from all generators for each problem and compute a single TPR and TNR per problem. The distribution of these metrics within each quartile is reported in Figure 8–9 of Section B.1. We observe that easier problems yield higher and more stable TPR, while harder problems exhibit lower and more variable TPR. In contrast, TNR distributions show no consistent correlation with problem difficulty. These results suggest that problem difficulty primarily drives verifier TPR, which then helps explain why and when the verification gains brought by weak and strong verifiers can converge in TTS. This connection is examined in Section 5.

4.2 How Does Generator Capability Influence Verification?



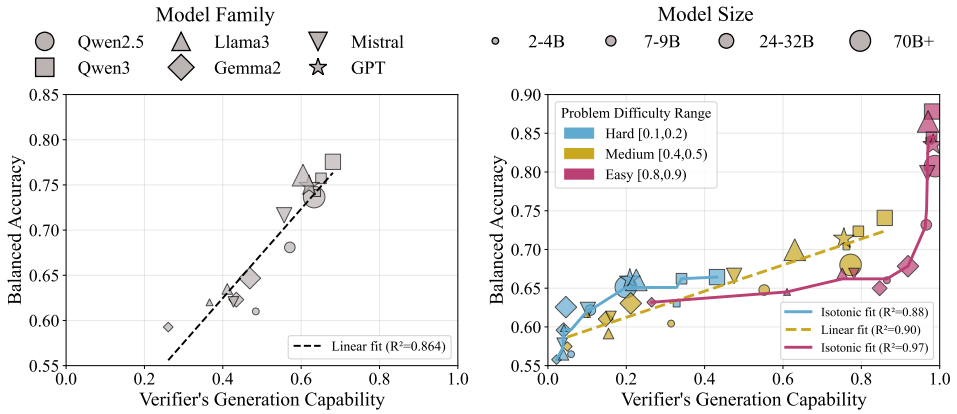
(a) TPR of verifier–generator pairs (b) TNR of verifier–generator pairs

Figure 3: **Generator capability influences verifier performance of identifying incorrect responses.** Heatmaps show (a) TPR and (b) TNR when pairing 15 verifier models (rows) with 15 generator models (columns). Rows and columns are ordered by models' generation capability, measured on the same subset of problems. Values denote mean verification performance over the subset, red means higher while blue means lower.

We study how generator capability affects verifier performance by pairing 15 verifier and 15 generator models. For fairness, we restrict evaluation to the subset of problems where each generator produces at least one incorrect response (for measuring TNR) and at least one correct response (for measuring TPR). This ensures all verifiers are tested on the same problems set, removing variance from problem selection. Figure 3 summarizes the results.

As shown in Figure 3a, TPR remains uniformly high across nearly all settings and increases further with stronger generators. The heatmap is dominated by red, with values typically above 0.8, indicating that most verifiers are already reliable at recognizing correct responses. As generator capability improves, TPR approaches 1.0. This suggests that generator strength influences recognition of correct responses in a relatively mild way. As we show next, generator capability plays a much stronger role in shaping error detection rate TNR.

Generator capability correlates with error detection in verification. In Figure 3b, moving from weaker generators on the left to stronger ones on the right, the heatmap shifts generally from red to blue, indicating a substantial decrease in TNR. For example, for the Qwen2.5-72B verifier, TNR drops from 0.68 on solutions generated by Llama-3.1-8B to 0.17 on those by Qwen3-32B. The overall pattern is consistent across nearly all verifiers: weaker generators produce errors that are easier to detect, while errors from stronger generators are harder to catch. These results show that generator capability strongly modulates the detection of incorrect responses. In Section 5, we examine how this dependence carries over to TTS, where verifier's high TNR on weak–medium generators yields the largest verification gains, higher than the same verifier achieves on strong generators.



(a) Average Acc_{bal} over all problems (b) Average Acc_{bal} on three problem difficulty

Figure 4: **Diverse correlation forms between verification performance and generation capability.** Scatter plots show balanced accuracy as a function of verifier's generation capability. Shapes indicate model family and marker size denotes model scale. (a) Averaged across all problems, verifier generation capability exhibits a strong linear correlation with Acc_{bal} ($R^2 = 0.864$). (b) Stratified by problem difficulty, correlations differ: on hard problems, isotonic regression outperforms linear (0.88 vs. 0.64), revealing a non-linear threshold effect; on medium problems, both fits are strong (0.96 vs. 0.90), indicating linearity; on easy problems, isotonic captures saturation (0.97) while linear fails (0.48).

4.3 How does Verifier Generation Capability Impact Verification?

We measure verifier generation capability, and evaluate verification performance with balanced accuracy (Acc_{bal}) on the entire test set. Each verifier is evaluated on responses from a range of generators, and we report results both averaged across all problems and stratified by problem difficulty. To characterize these trends, we fit linear and isotonic regression models [6] to each difficulty bin and report R^2 values as measures of fit quality.

Figure 4a shows a strong overall correlation between verifier generation capability and Acc_{bal}. This result is consistent with prior work showing that evaluator accuracy tends to track evaluator's task performance [32], with the relationship appearing nearly linear [3]. While this global trend validates findings in prior work [3], a closer inspection into the trend reveals highly non-linear regimes.

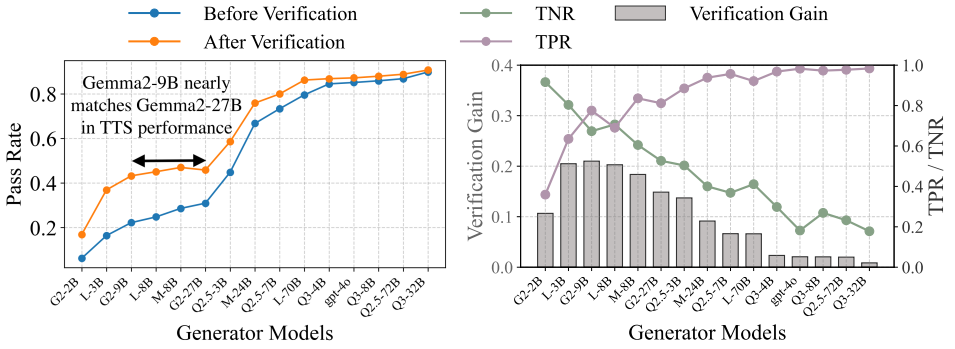
Verifier generation capability influences verification accuracy differently based on problem difficulty. The stratified analysis reveals that the correlation is regime-dependent and exhibits clear phase-transition behavior. We partition problems into 10 equal-width bins by problem difficulty $d(x) \in [0, 1]$. In Figure 4b, we present three representative intervals: *hard* $[0.1, 0.2)$, *medium* $[0.4, 0.5)$, and *easy* $[0.8, 0.9)$, with the full analysis across all intervals shown in Figure 10 of Section B.2. On hard problems (blue), Acc_{bal} improves as capability rises to roughly 0.1–0.2, but then plateaus around 0.65, with little further gains. On medium problems (yellow), accuracy rises steadily with capability, showing an essentially linear trend. On easy problems (red), capability saturates and no longer predicts accuracy, suggesting other factors explain the residual variation. Regression results (caption of Figure 4) confirm these patterns: linear models suffice for medium problems, whereas isotonic regression better captures the nonlinear plateau and saturation.

These findings highlight the need for a regime-aware verifier strategy. On hard problems, strong verifiers are not always necessary: once generation capability exceeds a threshold, performance plateaus, and weaker verifiers suffice. On medium problems, increasing verifier generation ability yields clear, consistent gains. On easy problems, the generation capability no longer predicts performance, so other verifier properties must guide selection.

5 Application to Test-Time Scaling

Our analysis in Section 4 is conducted with verification itself as the end goal. However, our findings have direct implications for TTS. We analyze two research questions in TTS settings that naturally arise out of our previous findings, and present our results below:

- **RQ4: Given a fixed verifier, can a weak generator match a stronger generator in TTS?** Weak generators can match stronger generators' post-verification performance (e.g., Gemma2-9B



(a) Pass rate before/after verification (b) Verification gain and TPR/TNR of the verifier

Figure 5: **TTS performance before and after verification when sweeping generator strength.** (a) Pass rate before (blue) and after (orange) adding a fixed verifier (GPT-4o), across generators ordered from weaker (left) to stronger (right) by generation capability. (b) Bar chart shows the verification gain $\Delta\hat{p} = \hat{p}_{G,V} - \hat{p}_G$ (left y-axis) for each generator. Lines show the verifier's TNR and TPR on the same datasets (right y-axis). Results reported on 119 problems with difficulty in $[0.5, 0.6)$.

matches Gemma2-27B). Verification gains peak at weak-medium generators due to high TNR filtering their errors while maintaining moderate TPR.

- **RQ5: Can weak verifiers match the gains of strong verifiers in TTS?** The verification gain gap between weak and strong verifiers narrows in three regimes: easy problems (high TPR for both), strong generators (low TNR for both), and very hard problems. The gap is largest for medium-difficulty problems with weak-medium generators.

5.1 Can a weak generator match a stronger generator in TTS?

We evaluate TTS with a fixed verifier (GPT-4o) by sweeping generator capability and reporting pass rates before and after verification, along with the verification gain $\Delta\hat{p}$ (Section 3.1).

Verification gain peaks for weak–medium generators, enabling them to match stronger models post-verification. As shown in Figure 5a, weak generators start with much lower pass rates but improve dramatically after verification, reaching levels comparable to larger models. For example, Gemma2-9B starts from a significantly lower baseline but, after verification, achieves a pass rate nearly matching Gemma2-27B, with the gap shrinking from about 8.7% to 2.6%

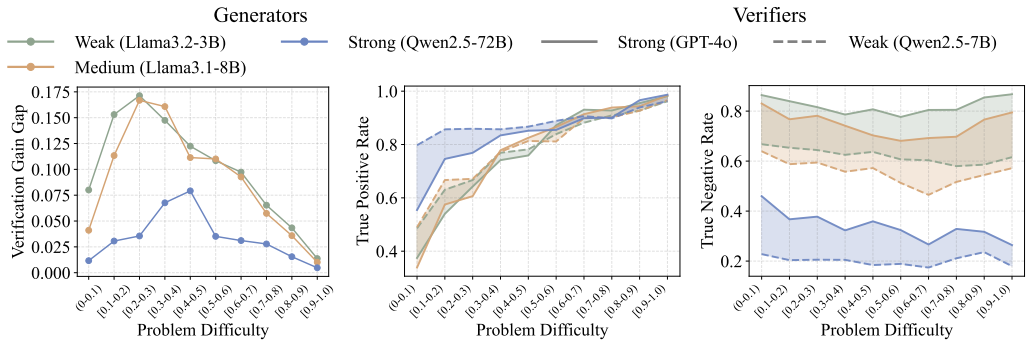
Figure 5b explains this phenomenon. The TNR and TPR curves in the figure reflects our finding in **RQ2** in Section 4.2: when generator strength increases (left to right), TNR decreases sharply, while TPR rises only modestly beyond the weakest model. This leads to the inverted-U pattern in verification gain (gray bars), which peaks at weak–medium strength. In this regime, high TNR enables effective error filtering while moderate TPR preserves sufficient correct responses, allowing weak–medium generators to catch up to larger ones in post-verification performance. For the strongest generators, however, errors are harder to identify without a substantially stronger verifier, leading to a collapse in TNR and limiting further verification gains.

Figure 5 highlights this effect on a specific difficulty slice ($d(x) \in [0.5, 0.6]$), while Figure 11 includes the complete analysis across the full difficulty spectrum. We observe consistent trends across different difficulty levels: in medium-difficulty problems, weak generators can match larger ones, confirming that the "catch-up" effect is not limited to a single difficulty band. Overall, with a fixed verifier, weak or medium generators combined with verification can achieve performance comparable to larger models, providing a more compute-efficient alternative without loss of quality.

5.2 Can weak verifiers match the gains of strong verifiers in TTS?

We analyze the gap in verification gains between a strong verifier and a weaker one across problem difficulty ranges and generator strengths, as shown in Figure 6. Our goal is to identify when this gap narrows, since such regimes suggest weak verifiers can substitute for strong ones.

The gap narrows on easier problems. As shown in Figure 6a, the difference in verification gain shrinks as problem difficulty decreases, which corresponds to the rising TPR in Figure 6b. This mirrors our earlier finding in **RQ1**: easier problems improve TPR for both weak and strong verifiers.



(a) Verification Gain Gap (b) TPR (c) TNR

Figure 6: **Weak vs. strong verifiers under varying problem difficulty and generator strength.** The x-axis shows problem difficulty, ordered from hardest to easiest measured relative to each generator. (a) Verification gain gap ($\Delta p_{V_{\text{strong}}} - \Delta p_{V_{\text{weak}}}$) between a strong verifier (GPT-4o) and a weaker one (Qwen2.5-7B) when applied to weak (Llama-3.2-3B), medium (Llama-3.1-8B), and strong (Qwen2.5-72B) generators. (b) TPR increases as problems become easier. (c) As generators strengthen, TNR decreases overall, and the TNR gap (shadow band) between strong and weak verifiers narrows.

Consequently, even weak verifiers reliably recognize correct responses on easy problems, leaving little room for strong ones to provide additional benefit. Noted that the practical impact of this narrowing gap is limited because verification itself becomes less useful on easy problems. As seen in Figure 12, verification gains decrease for both strong and weak verifiers on easy problems. Thus, while the gap between verifiers narrows, this occurs in a regime where verification provides minimal practical value.

The gap narrows as generators become stronger. Figure 6a shows that increasing generator capability reduces the difference between weak and strong verifiers. This is consistent with Section 4.2 (**RQ2**), where we observed that verifier's TNR decreases as generator capability increases. Because both weak and strong verifiers experience lower TNR, the gap between them also shrinks (see the narrowing shaded band between solid and dashed curves in Figure 6c).

The gap dips in the hardest problem regime. In the hardest bins, the verification-gain gap again becomes small. As shown in Figure 6b, weak verifiers even exhibit slightly higher TPR than strong verifiers, reflecting a more aggressive tendency to mark responses as correct. From another perspective, Figure 12 and **RQ3** show that balanced accuracy quickly saturates on very hard problems, leaving only small performance differences between verifiers.

These results indicate that strong verifiers provide the largest advantage in the medium-difficulty regime, particularly when paired with weak or medium-strength generators. In contrast, at the extremes such as very easy problems, very hard problems, or cases where generators are already very strong, weak verifiers can approximate the strong ones in TTS. However, these convergence regimes coincide with minimal verification benefit overall. As Figure 12a shows, verification gains for both verifiers drop below 0.05 for very hard problems and very easy problems, while strong generators (Qwen2.5-72B) yield peak gains of only 0.1. Thus, while weak and strong verifiers converge in these regimes, this occurs where verification adds little value to TTS.

6 Conclusion

We studied LLM verification across problem difficulty, generator capability, and verifier's generation capability, revealing that verification success depends on interactions between these factors rather than verifier's generation capability alone. We found that problem difficulty primarily shapes correctness recognition, generator capability influences error detectability, and verifier generation capability correlates with verification in a problem-difficulty-dependent patterns. These insights have practical implications on TTS. With a fixed verifier, weaker generators can approach the post-verification performance of stronger ones. Weak verifiers can also substitute for strong ones in specific regimes of problem difficulty and generator capability. This enables strategic model pairing that reduces computational costs while maintaining performance quality. Our results advocate for a regime-aware approach to allocate compute for verification. While our study focuses on math reasoning, these principles extend to other domains with verifiable outputs, warranting future investigation.

References

[1] Anastasios N Angelopoulos, Jacob Eisenstein, Jonathan Berant, Alekh Agarwal, and Adam Fisch. Cost-optimal active ai model evaluation. *arXiv preprint arXiv:2506.07949*, 2025.

[2] 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.

[3] Wei-Lin Chen, Zhepei Wei, Xinyu Zhu, Shi Feng, and Yu Meng. Do llm evaluators prefer themselves for a reason? *arXiv preprint arXiv:2504.03846*, 2025.

[4] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[5] 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.

[6] Jan de Leeuw, Kurt Hornik, and Patrick Mair. Isotone Optimization in R: Pool-Adjacent-Violators Algorithm (PAVA) and Active Set Methods. *Journal of Statistical Software*, 32(5): 1–24, 2009. doi: 10.18637/jss.v032.i05.

[7] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of transformers on compositionality. *Advances in Neural Information Processing Systems*, 36: 70293–70332, 2023.

[8] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

[9] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[10] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

[11] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[12] Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

[13] 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. *arXiv preprint arXiv:2310.01798*, 2023.

[14] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[15] Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K Jain, Virginia Aglietti, Disha Jindal, Peter Chen, et al. Big-bench extra hard. *arXiv preprint arXiv:2502.19187*, 2025.

[16] Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, et al. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *arXiv preprint arXiv:2504.09037*, 2025.

[17] Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. Maszero: Designing multi-agent systems with zero supervision. *arXiv preprint arXiv:2505.14996*, 2025.

[18] Michael Krumdick, Charles Lovering, Varshini Reddy, Seth Ebner, and Chris Tanner. No free labels: Limitations of llm-as-a-judge without human grounding. *arXiv preprint arXiv:2503.05061*, 2025.

[19] H. Kydlíček. Math-verify: Math verification library, 2025. URL `https://github.com/huggingface/math-verify`.

[20] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

[21] Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q. Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. Hugging Face repository, 2024. Available at `https://huggingface.co/datasets/AI-MO/NuminaMath-CoT`.

[22] 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 *The Twelfth International Conference on Learning Representations*, 2023.

[23] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *Conference on Language Modeling (COLM)*, 2025.

[24] Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*, 2025.

[25] Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative reward models. *arXiv preprint arXiv:2410.12832*, 2024.

[26] Jon Saad-Falcon, E Kelly Buchanan, Mayee F Chen, Tzu-Heng Huang, Brendan McLaughlin, Tanvir Bhathal, Shang Zhu, Ben Athiwaratkun, Frederic Sala, Scott Linderman, et al. Shrinking the generation-verification gap with weak verifiers. *arXiv preprint arXiv:2506.18203*, 2025.

[27] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[28] Parshin Shojaee*, Iman Mirzadeh*, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity, 2025. URL `https://ml-site.cdn-apple.com/papers/the-illusion-of-thinking.pdf`.

[29] 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.

[30] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

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

[32] Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. Judgebench: A benchmark for evaluating llm-based judges. *arXiv preprint arXiv:2410.12784*, 2024.

[33] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[34] Qwen Team. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[35] Han Wang, Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. Soft self-consistency improves language model agents. *arXiv preprint arXiv:2402.13212*, 2024.

[36] Peifeng Wang, Austin Xu, Yilun Zhou, Caiming Xiong, and Shafiq Joty. Direct judgement preference optimization. *arXiv preprint arXiv:2409.14664*, 2024.

[37] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[38] Jason Wei. The asymmetry of verification and verifier's law. `https://www.jasonwei.net/blog/asymmetry-of-verification-and-verifiers-law`, 2025. Accessed: 2025-09-06.

[39] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[40] Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason Weston, Ilia Kulikov, and Swarnadeep Saha. J1: Incentivizing thinking in llm-as-a-judge via reinforcement learning. *arXiv preprint arXiv:2505.10320*, 2025.

[41] Austin Xu, Yilun Zhou, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. J4r: Learning to judge with equivalent initial state group relative policy optimization. *arXiv preprint arXiv:2505.13346*, 2025.

[42] Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, et al. Towards large reasoning models: A survey of reinforced reasoning with large language models. *arXiv preprint arXiv:2501.09686*, 2025.

[43] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[44] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[45] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

[46] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

[47] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=Ccwp4tFEtE`.

[48] Yilun Zhou, Austin Xu, Peifeng Wang, Caiming Xiong, and Shafiq Joty. Evaluating judges as evaluators: The jetts benchmark of llm-as-judges as test-time scaling evaluators. *arXiv preprint arXiv:2504.15253*, 2025.

Appendix

A Additional Preliminaries and Setup

A.1 Verification-Augmented Test-time Scaling.

In Section 3, we have defined that TTS performance without verification is measured as $\hat{p}_G(\mathcal{D}; K)$, while the performance of verification-augmented TTS is measured as

$$\hat{p}_{G,V}(\mathcal{D}; K) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \left(\frac{1}{K'} \sum_{i=1}^{K} \mathbb{1}\{a(r_i) = y^\star(x),\, V(x, r_i) = 1\} \right)$$
$$\text{where } K' = \sum_{i=1}^{K} \mathbb{1}\{V(x, r_i) = 1\}.$$

There are two corner cases require special handling when using these metrics.

Case 1: The verifier rejects all candidates. If the verifier assigns "Incorrect" to every response, then $K' = 0$. In this case, there is no accepted response to evaluate. To ensure that the metric remains well defined and comparable to the baseline, we fall back to the generator's empirical pass rate:

$$\hat{p}_{G,V}(\mathcal{D}; K) = \hat{p}_G(\mathcal{D}; K).$$

This corresponds to randomly selecting one of the K samples, exactly as in the non-verified setting. Intuitively, if the verifier provides no usable signal, performance should revert to the generator alone.

Case 2: The verifier is undecided. In some situations, the verifier does not produce a clear verdict (e.g., neither "Correct" nor "Incorrect"), which we denote as $V = -1$. We refer to this as the verifier being undecided. To handle these undecided cases, we introduce a randomized acceptance rule: when $V = -1$, the response is retained with probability $q \in (0, 1)$ and rejected otherwise. In our implementation, we set $q = \frac{1}{2}$, corresponding to flipping a fair coin. To avoid randomness in evaluation, we use the deterministic expression corresponding to the expected value of the randomized coin-flip procedure. In this view, each undecided response contributes a fractional weight q to both numerator and denominator. This yields the estimator

$$\hat{p}_{G,V,q}(\mathcal{D}; K) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \frac{\sum_{i=1}^{K} \mathbb{1}\left\{a\left(r_i\right) = y^\star(x)\right\} \left(\mathbb{1}\left\{V\left(x, r_i\right) = 1\right\} + q\mathbb{1}\left\{V\left(x, r_i\right) = -1\right\}\right)}{\sum_{i=1}^{K} \left(\mathbb{1}\left\{V\left(x, r_i\right) = 1\right\} + q\mathbb{1}\left\{V\left(x, r_i\right) = -1\right\}\right)}$$

For the choice $q = \frac{1}{2}$, undecided responses are half-weighted in both numerator and denominator. This ensures that responses with uncertain verifier judgments still influence the metric proportionally, while keeping evaluation stable and unbiased.

A.2 Evaluating the Response Correctness

Here we detail the evaluation procedure for establishing response correctness, including fallback methods. Ground-truth correctness is determined using `Math-Verify` [19]. If `Math-Verify` fails to parse an answer or returns incorrect, we recheck with other string-matching verifiers from open-source repositories `lm-eval` [8], `Dr.GRPO` [23], and `Qwen2.5-Math` [43]. We further apply GPT-4.1-mini and Qwen2.5-72B to conduct reference-based evaluation and check the equivalence of the model prediction and ground-truth answers. The prompt template for LLM-based verification is provided in Section C.

B Additional results

B.1 Problem difficulty

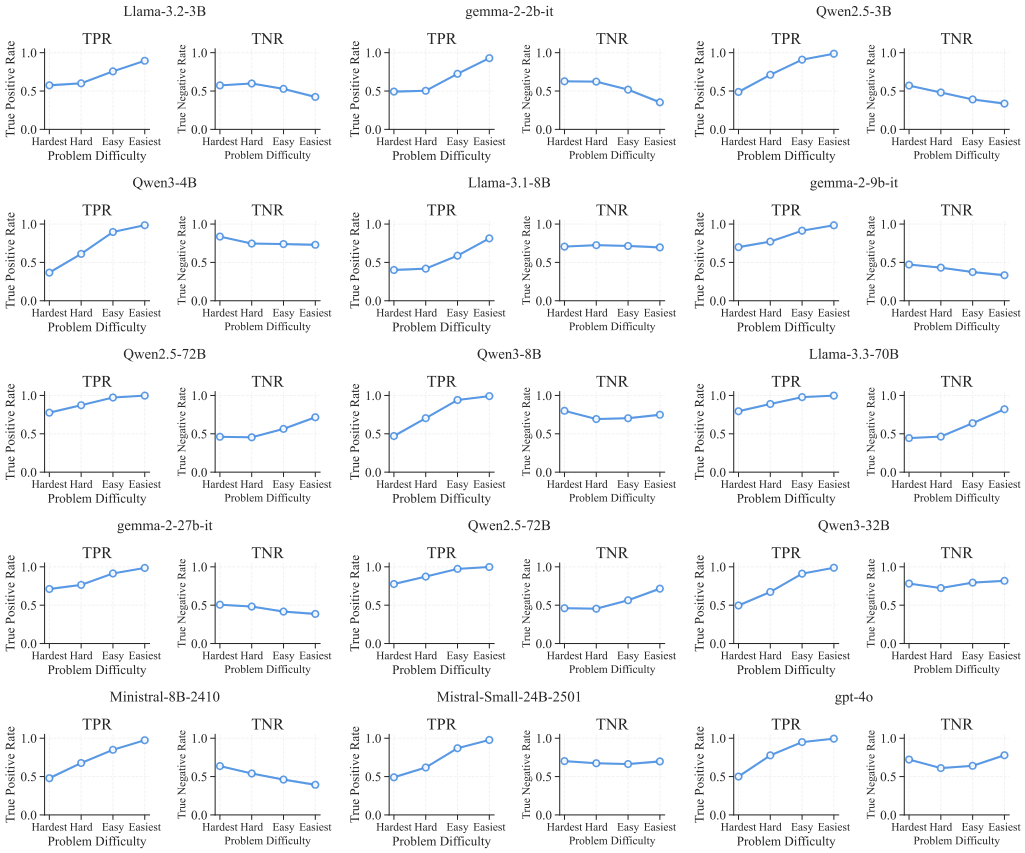


Figure 7: TPR and TNR trends across problem difficulty for each verifier model. Each panel shows the TPR and TNR of a single verifier across four difficulty quartiles. Across all models, TPR increases steadily as problems become easier, while TNR remains largely unchanged, confirming that problem difficulty mainly affects recognition of correct responses rather than rejection of incorrect ones.

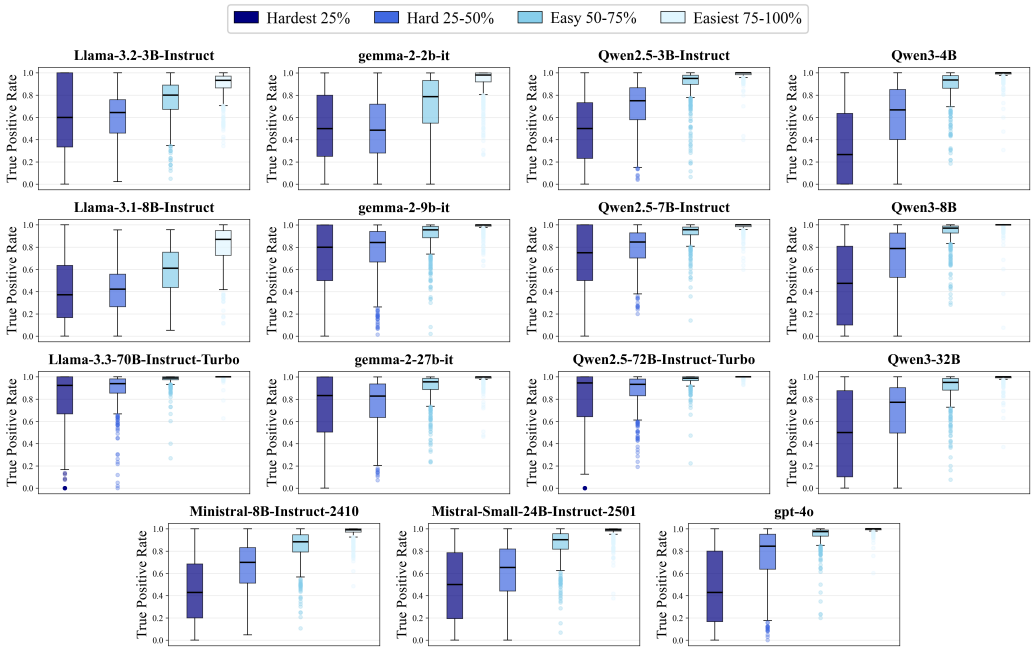


Figure 8: **Problem difficulty has a strong effect on** TPR **across problems.** Each boxplot shows the distribution of per-problem TPR for 15 verifier models, grouped by difficulty quartiles (hardest 25% to easiest 25%). Easier problems yield consistently higher and less variable TPR, while harder problems result in lower and more dispersed TPR.

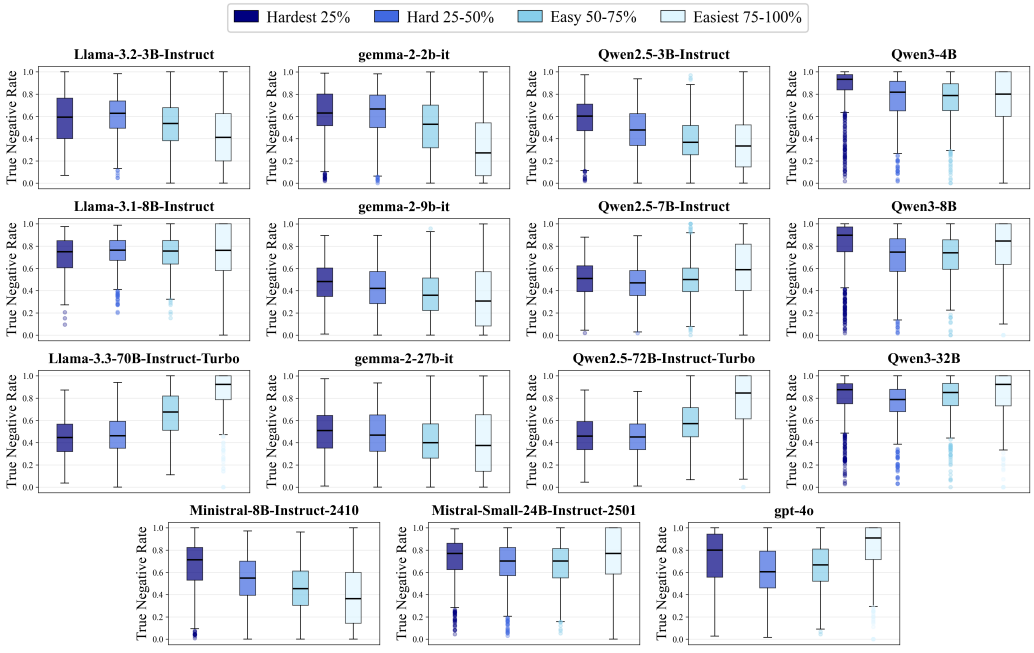


Figure 9: **Problem difficulty has little effect on** TNR **across problems.** Each boxplot shows the distribution of per-problem TNR for each verifier model, grouped by difficulty quartiles (hardest 25% to easiest 25%). In contrast to TPR, the TNR distributions remain narrow and stable across all quartiles, indicating that problem difficulty has minimal impact on rejecting incorrect responses.

B.2 Verifier generation capability

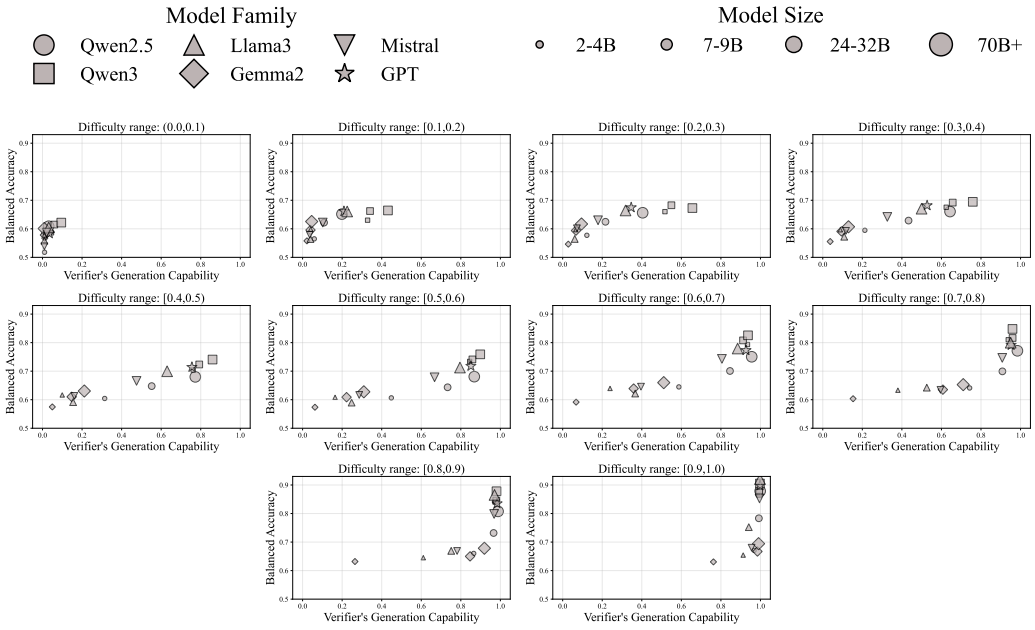


Figure 10: **Correlation between verification performance and generation capability across problem difficulty ranges.** Balanced accuracy as a function of verifier generation capability for difficulty ranges from (0.0,0.1) to [0.9,1.0). Performance shows three distinct regimes: threshold-limited plateaus on hard problems (0.0-0.4), sharp monotonic increases on medium problems (0.4-0.7), and high variance despite capability saturation on easy problems (0.7-1.0). Marker shapes indicate model family; sizes represent model scale.

B.3 Test-time scaling

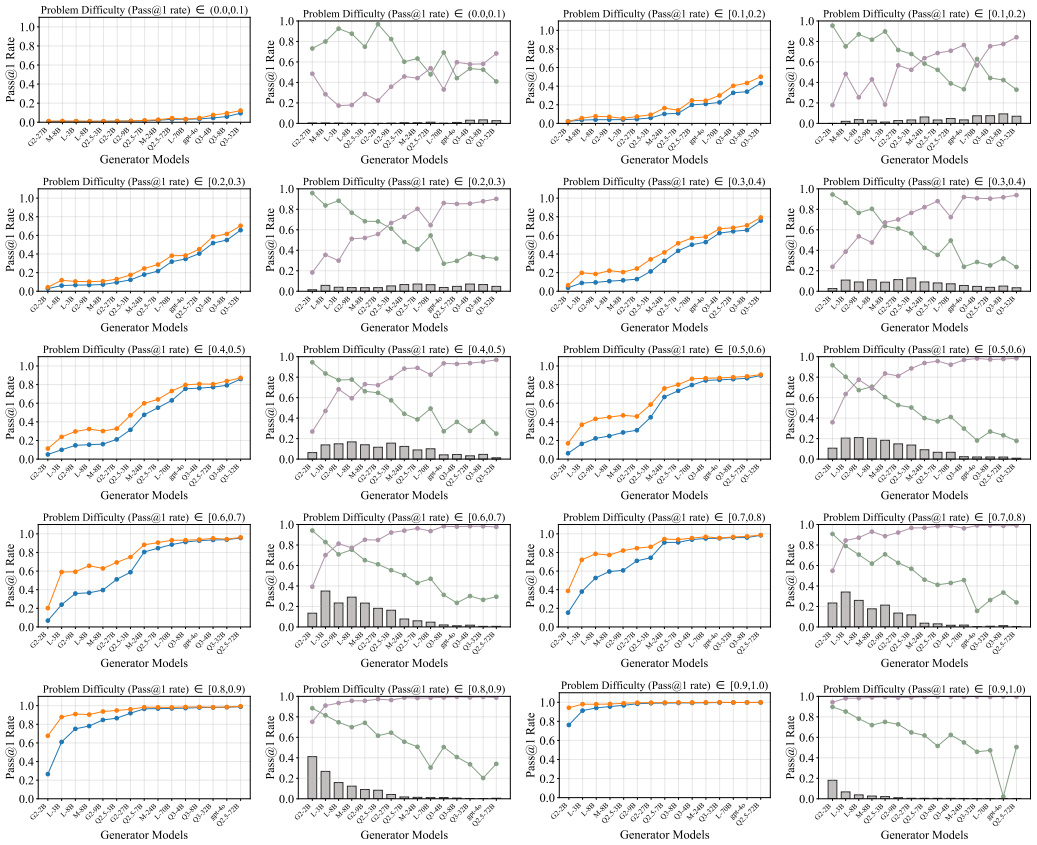
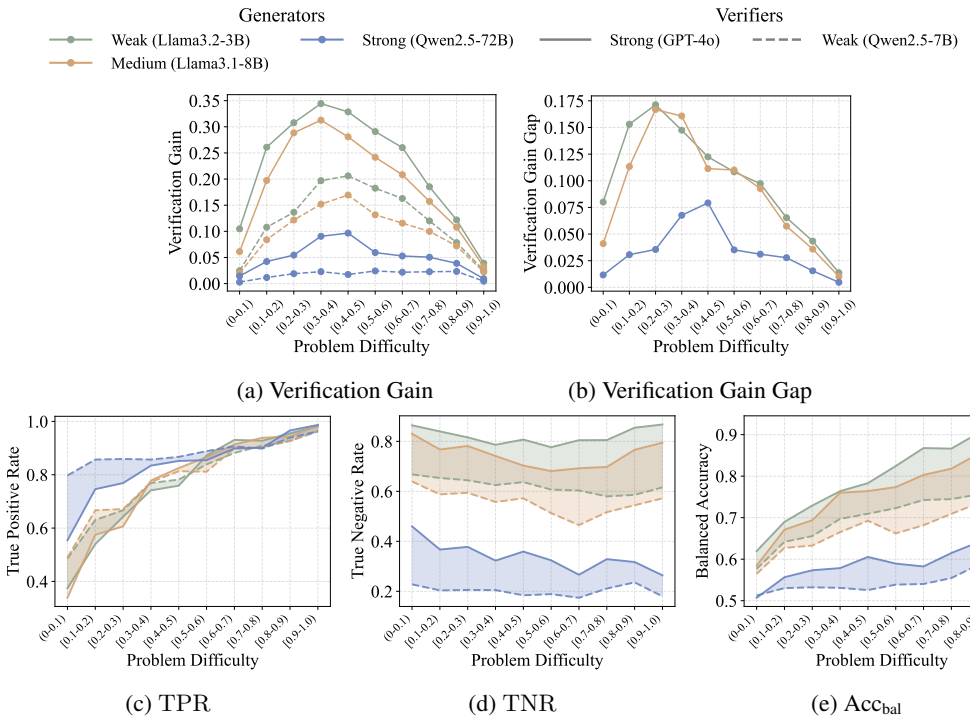


Figure 11: Verification-augmented TTS performance across the full range of problem difficulties. Each pair of figure corresponds to a different difficulty interval (measured by pass rate $d(x)$), with the left panel showing pass rates before (blue) and after (orange) verification, and the right panel showing verification gain Δp (bars) alongside the verifier's TNR (green) and TPR (purple). Compared to Figure 5, which focused only on problems with $d(x) \in [0.5, 0.6]$, this includes all possible difficulty range.



(a) Verification Gain (b) Verification Gain Gap

(c) TPR (d) TNR (e) Acc_{bal}

Figure 12: **Analyzing verification gain gaps and TPR/TNR/Acc_{bal} between weak and strong verifiers under varying problem difficulty and generator strength.** The x-axis of all subfigures shows problem difficulty measured relative to each generator. (a) Verification gain (Δp) for both strong (GPT-4o, solid lines) and weak (Qwen2.5-7B, dashed lines) verifiers across three generators: weak (Llama-3.2-3B), medium (Llama-3.1-8B), and strong (Qwen2.5-72B). Both verifiers show inverted-U patterns, with gains peaking at medium difficulty. (b) Verification gain gap ($\Delta p_{V_{\text{strong}}} - \Delta p_{V_{\text{weak}}}$) between strong and weak verifiers. The gap is largest for medium-difficulty problems and narrows at both extremes (very easy and very hard problems). (c) TPR increases steadily as problems become easier for all generator-verifier combinations. (d) TNR shows higher values with weaker generators and decreases as generators become stronger, with the gap between weak and strong verifiers narrowing for stronger generators. (e) Balanced accuracy (Acc_{bal}) improves with problem easiness. Shaded regions visualize the performance gap between strong and weak verifiers for each generator. The gap is smallest at hard problems.

C Prompt templates

Response Generation Prompt. Here, we provide the prompt for generating model responses to math questions. For each model, we use its default system prompt as specified in the model documentation.

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

Verification Evaluation Prompt. Below we present the prompt template used to evaluate verification performance.

```
### System Prompt
Please act as an impartial judge and evaluate the correctness of the response provided by an AI
assistant to the user prompt displayed below. You will be given the assistant's response.

When evaluating the assistant's response, identify any mistakes or inaccurate information. Be as
objective as possible. Avoid any biases, such as order of responses, length, or stylistic elements
like formatting.

Before providing an your final verdict, think through the judging process and output your thoughts
as an explanation

After providing your explanation, you must output only one of the following choices as your final
verdict with a label:

1. The response is correct: [[Correct]]
2. The response is incorrect: [[Incorrect]]

Use the following template:
Explanation: Your detailed thought process as an explanation.
Verdict: [[Correct]] or [[Incorrect]].

### User Prompt
<|User Prompt|>
{question}

<|The Start of Assistant's Answer|>
{response}
<|The End of Assistant's Answer|>
```

Fallback Verification Prompt. When Math-Verify returns unparsable or incorrect results, we employ LLM-as-judge as a fallback mechanism for correctness verification. Below we provide the prompt template used for this secondary verification step:

```
### User Prompt
Given a math problem, its correct answer, and the model's generated answer, determine if the
model's generated answer is correct.

VALIDATION CRITERIA:
1. Identify the final answer, which is usually put inside \\boxed{{answer}} or **answer**.
2. The answer must be mathematically equivalent to the correct answer
3. The answer must be complete with a clear final result
4. The answer must not just contain similar numbers - it must reach the correct conclusion
5. If the generated answer contains multiple different final answers or is ambiguous about which is
the final answer, mark it as 'False'

IMPORTANT: Just having the same numbers as the ground truth is NOT sufficient - the model must
actually solve the problem correctly and provide the correct final answer in the designated format.

Respond with 'True' if the answer is correct and complete, and 'False' if it is incorrect or
incomplete.
Directly provide your judgement 'True' or 'False' without any other description.

Problem: {problem}
Correct Answer: {ground_truth_answer}
Model's Generated Answer: {model_response}
Your judgement:
```