

# 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 VARIATION IN VERIFICATION: UNDERSTANDING VERIFICATION DYNAMICS IN LARGE LANGUAGE MODELS

005 **Anonymous authors**

006 Paper under double-blind review

## 009 ABSTRACT

011 Recent advances have shown that scaling test-time computation enables large  
 012 language models (LLMs) to solve increasingly complex problems across diverse  
 013 domains. One effective paradigm for test-time scaling (TTS) involves LLM gen-  
 014 erators producing multiple solution candidates, with LLM verifiers assessing the  
 015 correctness of these candidates without reference answers. In this paper, we study  
 016 generative verifiers, which perform verification by generating chain-of-thought  
 017 (CoT) reasoning followed by a binary verdict. We systematically analyze verifi-  
 018 cation dynamics across three dimensions – problem difficulty, generator capability,  
 019 and verifier generation capability – through empirical studies on 12 benchmarks  
 020 across mathematical reasoning, knowledge, and natural language reasoning tasks  
 021 using 14 open-source models (2B to 72B parameter range) and GPT-4o. Our ex-  
 022 periments reveal three key findings about verification effectiveness: (1) Easy problems  
 023 allow verifiers to more reliably certify correct responses; (2) Weak generators pro-  
 024 duce errors that are easier to detect than strong generators; (3) Verification ability  
 025 is generally correlated with the verifier’s own problem-solving capability, but this  
 026 relationship varies with problem difficulty. These findings reveal opportunities for  
 027 optimizing basic verification strategies in TTS applications. First, given the same  
 028 verifier, some weak generators can nearly match stronger ones in post-verification  
 029 TTS performance (e.g., the Gemma2-9B to Gemma2-27B performance gap shrinks  
 030 by 75.7%). Second, we identify cases where strong verifiers offer limited advan-  
 031 tages over weak ones, as both fail to provide meaningful verification gains, suggest-  
 032 ing that verifier scaling alone cannot overcome fundamental verification challenges.

## 032 1 INTRODUCTION

034 Large language models (LLMs) have advanced rapidly in solving reasoning tasks such as  
 035 mathematics and code generation, yet their outputs remain unreliable, often containing subtle or  
 036 obvious mistakes (Ke et al., 2025; Lightman et al., 2023). LLM based verification (Angelopoulos  
 037 et al., 2025; Huang et al., 2024; Mao et al., 2024; Pan et al., 2025) has emerged as a central  
 038 mechanism to identify such errors in a scalable manner. Recent work has increasingly focused  
 039 on *generative verifiers* (Liu et al., 2025d; Mahan et al., 2024; Zhang et al., 2025), which frame  
 040 verification as next-token prediction: the model typically generates a chain-of-thought (CoT)  
 041 reasoning trace and then outputs a binary verdict token. This approach has been shown to outperform  
 042 earlier discriminative verifiers or scalar reward models (RMs, Lightman et al., 2023), as it better  
 043 leverages the inherent text-generation capabilities of LLMs. One valuable downstream application  
 044 of automatic verification is test-time scaling (TTS), where additional inference-time compute is  
 045 allocated to improve generation performance. A popular paradigm of TTS is the use of a verifier  
 046 model to evaluate candidate responses, filter errors, and identify correct solutions. This approach  
 047 underlies techniques such as rejection sampling (Brown et al., 2024), re-ranking (Zhou et al., 2025),  
 048 weighted majority voting (Wang et al., 2024a; 2023), and step-level generation (Snell et al., 2025).

049 Current practice in LLM verification often deploys strong, typically closed-source frontier models as  
 050 verifiers. This practice rests on the assumption that verification quality scales with a verifier’s capa-  
 051 bility to solve the same problem (i.e., its generation capability), a correlation demonstrated in recent  
 052 work (Chen et al., 2025c; Krumdick et al., 2025; Tan et al., 2025). However, this practice may be sub-  
 053 optimal given that verifying a solution is often easier than generating one from scratch, a phenomenon  
 referred to as “verification asymmetry” (Wei, 2025). This asymmetry appears in several fields. In  
 convex optimization, dual certificates enable efficient validation of optimality of a proposed solution,

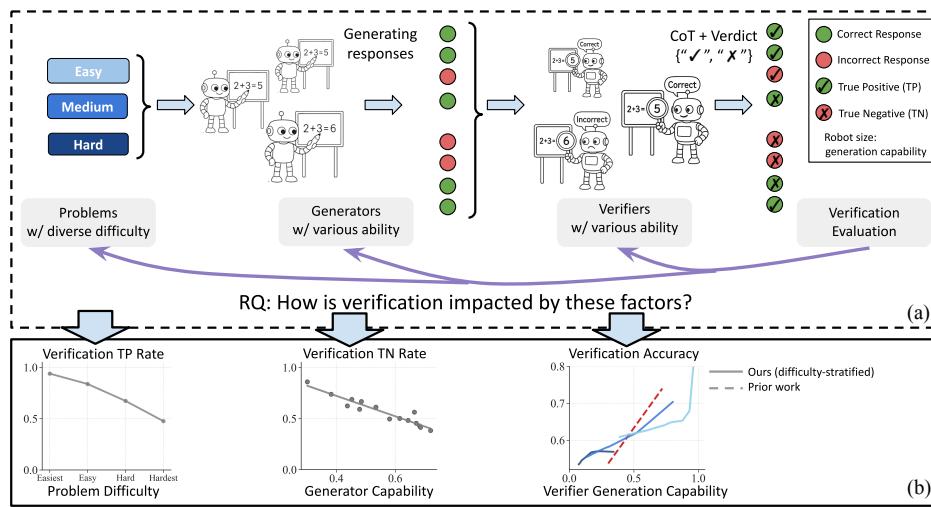


Figure 1: **Overview of our study on verification dynamics.** (a) We consider generative verification: an LLM generator produces a solution to a problem, and an LLM verifier conditions on the problem and solution to generate a verification CoT followed by a binary verdict (“Correct”/“Incorrect”). We design controlled experiments that vary problem difficulty, generator generation capability, and verifier generation capability, investigating how each of these factors influences 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 in prior work. The three plots were generated by aggregating benchmark data across three domains and averaging performance metrics over 15 models.

while in factorization, verifying correctness is trivial compared to finding the prime factors. Thus, it is worth investigating verification as a distinct capability rather than merely a byproduct of generation.

Despite extensive research on generation dynamics and the factors influencing generation quality (Allen-Zhu & Li, 2025; Chen et al., 2024; Ye et al., 2025), the dynamics of verification remain largely unexplored. In particular, little is known about how problem characteristics, properties of generated 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 dimensions—problem difficulty, generator capability, and verifier generation capability—shown in Figure 1. We quantify verification performance by measuring the probability of the verifier recognizing both correct and incorrect generated solutions in controlled experimental settings. We focus on verifiable problems with objective ground-truth answers in mathematical reasoning, knowledge question-answering (QA), and natural language (NL) reasoning domains. This allows us to objectively measure verifier and generator performance, while simulating the reference-free evaluation settings where verifiers are typically deployed in practice, e.g., in TTS. While our experiments use these domains as a testbed, we believe the insights should extend to any domain where correctness can be reliably defined and checked.

**Main Findings.** While prior work showed that verifier generation capability correlates with verification performance (Chen et al., 2025c; Krumdick et al., 2025; Tan et al., 2025), we reveal that two additional factors, *problem difficulty* and *generator capability*, also critically influence verification success, as illustrated in Figure 1. Our analysis reveals:

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

108 Our empirical analysis includes 2,347 math problems from eight datasets, 1,196 knowledge QA  
 109 problems, and 901 NL reasoning problems, evaluated across 14 open-source models and GPT-4o.  
 110

111 **Application to TTS.** We demonstrate the practical implications of our findings for TTS. First, given  
 112 the same verifier, the TTS performance of a weak generator can nearly match the performance of  
 113 a strong generator. For instance, Gemma2-9B achieves comparable performance to Gemma2-27B  
 114 when both use the same verifier, GPT-4o. Second, we identify regimes where a strong verifier (e.g.,  
 115 GPT-4o) offers no additional benefit and can be replaced by a weaker verifier (e.g., Qwen2.5-7B),  
 116 with both providing limited gains. This occurs with strong generators or with problems at either  
 117 extreme of the difficulty spectrum.

## 2 RELATED WORK

119 **Automatic Evaluation.** The deployment of LLMs as evaluators has emerged as a central mechanism  
 120 for scalable assessment, with efforts focusing on training specialized small evaluators through  
 121 fine-tuning (Wang et al., 2025; Whitehouse et al., 2025; Xu et al., 2025; Zhang et al., 2025). Beyond  
 122 reference-based verifiers (Chen et al., 2025a; Liu et al., 2025b), verification approaches include  
 123 self-verification (Chen et al., 2023; Huang et al., 2023; 2024; Kumar et al., 2024; Shinn et al.,  
 124 2023), where models reflect on or critique their own outputs, and multi-agent verification (Li et al.,  
 125 2023; Lifshitz et al., 2025; Zhuge et al., 2024), where multiple agents collaborate in debate-style or  
 126 hierarchical setups to improve reliability. Prior work identifies several factors influencing evaluation  
 127 performance. One important factor is evaluator generation capability. Krumdick et al. (2025)  
 128 find that evaluator performance changes significantly based on whether the evaluator is capable  
 129 of answering the question or not. Tan et al. (2025) demonstrate the correlation between pairwise  
 130 judging ability and generation ability on the same set of problems. Chen et al. (2025b) observe  
 131 linear relationships between evaluation improvements and reasoning-required sample proportions  
 132 in fine-tuned evaluators. Chen et al. (2025c) show a strong positive correlation between generation  
 133 capability and evaluation accuracy. Our work extends these findings by identifying unexplored  
 134 factors that influence evaluation performance. We also demonstrate that the relationship between the  
 135 evaluator generation capability and evaluation quality is more nuanced than previously understood.

136 **Verification for Test-Time Scaling.** Early studies explore how to effectively apply verification meth-  
 137 ods to improve TTS performance. Snell et al. (2025) show RMs improve various TTS approaches,  
 138 including Best-of-N and beam search, while Liu et al. (2025a) find that compute-optimal strategies  
 139 vary with policy models and problem difficulty. Recent work explores alternatives to discriminative  
 140 RMs: Zhang et al. (2025) show trained generative verifiers outperform RMs in Best-of-N, and  
 141 Zhou et al. (2025)’s JETTS benchmark demonstrates generative evaluators match outcome RMs  
 142 in reranking. While verification benefits from increased model size and test-time compute, recent  
 143 work addresses how to reduce these computational costs. Saad-Falcon et al. (2025) propose a  
 144 framework to aggregate weak verifiers to approach strong ones; Angelopoulos et al. (2025) balance  
 145 weak/strong evaluators for efficiency; Stroebel et al. (2024) analyze fundamental limits of resampling  
 146 with imperfect verifiers; and Singhi et al. (2025) propose strategies to balance solving-verification  
 147 trade-offs. Our work studies the factors driving verification and explores their implications for TTS.

## 3 EXPERIMENTAL SETUP

### 3.1 PRELIMINARIES

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

153 **Generator and Verifier.** A generator  $G$  maps a problem  $x$  to a distribution over responses, denoted  
 154  $r \sim G(\cdot|x)$ . A verifier  $V$  takes a problem-response pair  $(x, r)$  and outputs a judgment of correctness.  
 155 In the binary case,  $V(x, r) \in \{0, 1\}$ , where 1 indicates acceptance and 0 indicates rejection. More  
 156 generally, a generative verifier produces a verification CoT explaining its reasoning, followed by an  
 157 explicit verdict such as “Correct” or “Incorrect.” The prompt templates are provided in Appendix A.

158 **Generation Capability.** We measure the generation capability of a model using its *pass rate*. For  
 159 a generator  $G$  and problem  $x$ , we define  $p_G(x) = \Pr[a(r) = y^*(x) \mid r \sim G(\cdot|x)]$  as the pass rate  
 160 on a single problem, i.e., the probability that  $G$  solves  $x$  correctly on one sampled attempt. We define  
 161  $p_G(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} p_G(x)$  as the pass rate aggregated over a dataset  $\mathcal{D}$ , which we use as the overall

162 measure of a model’s generation capability. Since the verifiers we study are generic LLMs (e.g., GPT-  
 163 4o), we measure a verifier’s generation capability using the same metric by prompting it as a generator.  
 164

165 We estimate these pass rates empirically as  $\hat{p}_G(x; K)$  and  $\hat{p}_G(\mathcal{D}; K)$  by sampling  $K$  responses per  
 166 model-problem pair. Since  $K$  is fixed at 64 throughout, we often omit  $K$  for simplicity, writing  
 167  $\hat{p}_G(x)$  and  $\hat{p}_G(\mathcal{D})$ . We use temperature 0.7 and top-p 1.0 as default sampling hyperparameters, and  
 168 adopt recommended settings when available (e.g., temperature 0.7 and top-p 0.8 for the non-thinking  
 169 mode of Qwen3). Ground-truth correctness is established with `Math-Verify` (Kydlíček, 2025),  
 170 supplemented by LLM-as-a-judge grading to reduce false negatives (details in Appendix B.1). These  
 171 64 responses per problem–model pair are used to estimate generation capability and problem difficulty.  
 172

173 **Problem Difficulty.** We define the difficulty of a problem as the average pass rate across a set of  
 174 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  
 175 is: if most generators succeed,  $d(x)$  is high (easy problem), while if few succeed,  $d(x)$  is low (hard  
 176 problem). It provides a model-agnostic way to partition problems by difficulty, extending prior  
 177 work (Snell et al., 2025), which measured difficulty relative to a single generator.  
 178

179 **Verification Metrics and Evaluation.** We evaluate verifiers using true positive rate (TPR), the  
 180 probability of the verifier accepting a correct response:  $\text{TPR} = \mathbb{E}[V(x, r) \mid a(r) = y^*(x)]$ , and true  
 181 negative rate (TNR), the probability of rejecting an incorrect response:  $\text{TNR} = \mathbb{E}[1 - V(x, r) \mid  
 182 a(r) \neq y^*(x)]$ .<sup>1</sup> We also report balanced accuracy,  $\text{Acc}_{\text{bal}} = \frac{1}{2}(\text{TPR} + \text{TNR})$ , which accounts for  
 183 class imbalance. For verification evaluation, we subsample 8 responses from each 64-sample pool,  
 184 balanced with 4 correct and 4 incorrect when possible. For very hard problems with fewer than 4  
 185 correct responses, we keep all correct ones and sample incorrect ones to reach 8 total (and vice versa  
 186 for easy problems). Each verifier evaluates responses from all 15 models over the full test set using  
 187 greedy decoding, unless a controlled subset is specified.  
 188

189 **Verification-Augmented Test-time Scaling.** We consider the TTS setting of sampling multiple  
 190 responses from the generator and filtering with a verifier before evaluation. For each problem  $x \in \mathcal{D}$ ,  
 191 we sample  $K$  responses from the generator using a fixed temperature, with  $K = 64$  in our experiments.  
 192 Without verification, TTS performance is measured as  $\hat{p}_G(\mathcal{D}; K)$  (or  $\hat{p}_G(\mathcal{D})$ ), the empirical pass rate  
 193 defined above. With verification, the verifier  $V$  evaluates each candidate, and only responses deemed  
 194 “Correct” are retained for evaluation. The performance of verification-augmented TTS is measured as  
 195

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

196 where  $K' = \sum_{i=1}^{K'} V(x, r_i)$ . This metric represents the conditional pass rate, i.e., the fraction  
 197 of correct responses among those retained by the verifier. A corner case arises when the verifier  
 198 rejects all responses ( $K' = 0$ ); in this case, we set the metric to the generator’s pass rate  $\hat{p}_G(\mathcal{D})$ ,  
 199 so evaluation reverts to selecting from the original  $K$  responses in the non-verified setting. We define  
 200 the *verification gain* from verifier  $V$  as the difference relative to the performance without verification,  
 201  $\Delta \hat{p}_V = \hat{p}_{G,V}(\mathcal{D}) - \hat{p}_G(\mathcal{D})$ , which quantifies how much gain can be attributed to verification. Note  
 202 that our formulation of TTS differs from the common setting where a single “best” response (e.g.,  
 203 by majority vote) is selected and then evaluated. Instead, we report the empirical pass rate of the  
 204 verifier-retained pool, which can be interpreted as the expected accuracy of uniformly sampling one  
 205 response from that pool. This expectation-based view captures the average quality of verifier-retained  
 206 responses without tying performance to a specific selection strategy.  
 207

### 3.2 TASKS AND MODELS

208 **Mathematical Reasoning.** We collect a total of 2,347 problems from the test sets of eight mathe-  
 209 matical reasoning benchmarks: GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021),  
 210 OlympiadBench (He et al., 2024), AIME24/25 (Li et al., 2024), AMC23 (Li et al., 2024), Minerva-  
 211 Math (Lewkowycz et al., 2022), and BBEH Multi-step Arithmetic (Kazemi et al., 2025). We use the  
 212 entire test sets of these benchmarks, except for GSM8K, from which we subsample 600 of 1,319  
 213 problems to balance difficulty distribution and reduce the proportion of easy problems.  
 214

215 <sup>1</sup>If the verifier generates an invalid output (e.g., due to the CoT running out of max generation length), we  
 216 treat it as an uninformative verdict of “Correct” and “Incorrect” each with probability of 50%. Computationally,  
 217 we set  $V(x, r) = 0.5$  in this case, and also in Equation 1.

216 **Knowledge.** We use a subset of MMLU-Pro (Wang et al., 2024b) as our knowledge category. We  
 217 randomly subsample 10% from each of its 14 disciplines, yielding 1,196 problems. MMLU-Pro  
 218 consists of college-level multiple-choice questions spanning STEM, humanities, and social sciences.  
 219

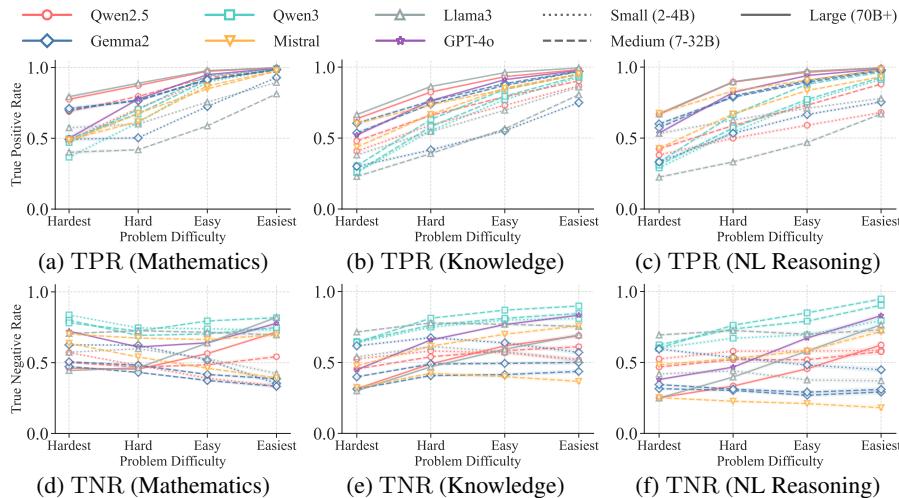
220 **Natural Language Reasoning.** We collect 901 multiple-choice problems from three benchmarks.  
 221 (1) ReClor (validation set, Yu et al., 2020), a multiple-choice benchmark requiring logical analysis  
 222 of short passages. (2) FOLIO (Han et al., 2022), a first-order logic reasoning benchmark in natural  
 223 language. (3) GPQA Diamond (Rein et al., 2024), a dataset that consists of graduate-level multiple-  
 224 choice science questions, requiring multi-step reasoning.

225 **Models.** We use 14 open-source models from four families: (1) Qwen2.5 at 3B, 7B, and 72B (Team,  
 226 2024); Qwen3 at 4B, 8B, and 32B (Yang et al., 2025); (2) Llama-3.2 at 3B, Llama-3.1 at 8B, and  
 227 Llama-3.3 at 70B (Grattafiori et al., 2024); (3) Gemma-2 at 2B, 9B, and 27B (Team et al., 2024); (4)  
 228 Minstral 8B and Mistral-Small-24B; and one closed-source model GPT-4o (Hurst et al., 2024). All  
 229 models are instruction-tuned versions by default. Each model is used both as a generator and a verifier.  
 230 We use abbreviated model names in figures for space efficiency; see Appendix B.2 for mappings.  
 231

## 4 EXPERIMENTAL RESULTS

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

- 234 • **RQ1: How does problem difficulty affect verification?** (Section 4.1) TPR increases steadily  
 235 with decreasing problem difficulty, meaning verifiers better recognize correct responses on easier  
 236 problems. However, TNR shows no predictable relationship with problem difficulty. This indicates  
 237 that problem difficulty primarily influences correctness recognition.
- 238 • **RQ2: How does the generator’s generation capability influence verification?** (Section 4.2)  
 239 As generators become stronger, TNR decreases substantially while TPR increases only slightly.  
 240 This reveals that generator capability primarily determines error detectability: stronger generators  
 241 produce errors that are harder for verifiers to identify.
- 242 • **RQ3: How does verifier generation capability impact verification?** (Section 4.3) Verifier  
 243 generation capability and verification performance are generally positively correlated. However,  
 244 the form of correlation depends heavily on problem difficulty: linear correlation occurs in medium-  
 245 difficulty problems, while nonlinear patterns appear in other difficulty levels.



261 **Figure 2: Problem difficulty primarily affects TPR of verification.** Each curve shows verifier  
 262 performance across four difficulty groups, with the  $x$ -axis indicating problem difficulty and the  $y$ -axis  
 263 reporting TPR (a-c) and TNR (d-f). Colors denote model families, and line styles indicate model size.  
 264

### 4.1 HOW DOES PROBLEM DIFFICULTY AFFECT VERIFICATION?

265 To examine how problem difficulty influences verification, we partition problems into four equal-sized  
 266 quartiles by their difficulty score  $d(x)$ , termed “hardest”, “hard”, “easy”, and “easiest”.

267 **Problem difficulty primarily influences the verifier’s ability to recognize correct responses.** Our  
 268 analysis is conducted at two levels of granularity: response level and problem level. Both analyses

270 reveal that problem difficulty mainly shapes the verifier’s sensitivity to correct responses, while not  
 271 consistently affecting its ability to identify incorrect responses.  
 272

273 At the response level, we compute the TPR and TNR of all responses within each difficulty quartile.  
 274 As shown in Figure 2, TPR increases steadily as problems become easier, while TNR shows no clear  
 275 trend. This pattern is consistent across model families and domains. At the problem level, we pool  
 276 responses from all generators for each problem and compute a single TPR and TNR per problem. The  
 277 distribution of these metrics within each quartile is reported in Figures 9 and 10 of Appendix C.1. We  
 278 observe that easier problems yield higher and more stable TPR, while harder problems exhibit lower  
 279 and more variable TPR. In contrast, TNR distributions show no consistent correlation with problem  
 280 difficulty. In Appendix C.2, we show that the main verification dynamics about TPR we identified  
 281 generalize to reasoning models, while extended reasoning provides benefits and alters TNR behavior.  
 282

283 To understand this pattern, case studies in Figure 32 show that verifiers tend to generate their  
 284 own reference solutions for comparison during verification. As the problem difficulty increases,  
 285 these verifier-generated answers become increasingly incorrect, producing false negatives (FNs)  
 286 that reduce TPR. Beyond case studies, we support this explanation with a large-scale analysis.  
 287 We use an independent LLM-as-judge to detect whether a verification CoT contains solving mis-  
 288 tacles. Specifically, we apply GPT-4.1-mini with  
 289 two prompts (in Appendix A): (i) to determine  
 290 whether the verifier’s CoT shows evidence of  
 291 generating its own solution or claim; (ii) if so, to  
 292 detect whether that generated content contains  
 293 mathematical or reasoning mistakes. A verifi-  
 294 cation CoT is labeled as containing a solving  
 295 mistake only if both conditions are satisfied. We  
 296 run this analysis on 367,920 GPT-4o verification  
 297 responses, the same ones for computing TPR.  
 298 Because the number of correct responses varies across difficulty bins (harder problems yield fewer  
 299 correct responses), we sample 5,000 correct responses per difficulty bin. For each bin, we compute  
 300 the percentage of FN and the percentage of FN that contain solving mistakes. We perform 10,000  
 301 bootstrap resamples and report the mean and 95% confidence intervals (CIs). As we show in Figure 3,  
 302 as problem difficulty increases, the verifier increasingly makes mistakes in generating the reference  
 303 answer, with 39.1% of verification FN containing a reference answer mistake in the hard set of prob-  
 304 lems. In contrast, on easy problems, there are substantially fewer FN and reference answer-related  
 305 mistakes, showing that faulty reference generation is a large driving factor of FN.  
 306

## 307 4.2 HOW DOES GENERATOR CAPABILITY INFLUENCE VERIFICATION?

308 We study how generator capability affects verifier performance by having each verifier evaluate  
 309 responses from each generator. Generators of different capabilities may produce extreme response  
 310 distributions, e.g., weak generators may produce no correct response on hard problems within 64  
 311 samples. To ensure fair comparison, we compute TPR on problem subsets where all generators  
 312 produce at least one correct response. Analogously, TNR is computed on problems where all  
 313 generators produce at least one incorrect response. Details are provided in Appendix B.3.  
 314

315 As shown in Figures 4a to 4c, TPR remains uniformly high across nearly all settings and increases  
 316 further with stronger generators. The heatmap is dominated by red colors, with values mostly  
 317 above 0.7, indicating that most verifiers are already reliable at recognizing correct responses. As  
 318 generator capability improves, TPR approaches 1.0. This suggests that generator strength influences  
 319 recognition of correct responses in a relatively mild way.  
 320

321 **Generator capability correlates with error detection in verification.** In Figures 4d to 4f, moving  
 322 from weaker generators on the left to stronger ones on the right, the heatmap shifts generally from  
 323 red to blue, indicating a substantial decrease in TNR. For example, in the Mathematics domain,  
 324 for the Qwen2.5-72B verifier, TNR drops from 0.68 on solutions generated by Llama-3.1-8B to  
 325 0.17 on those by Qwen3-32B. The overall pattern is consistent across three domains and nearly all  
 326

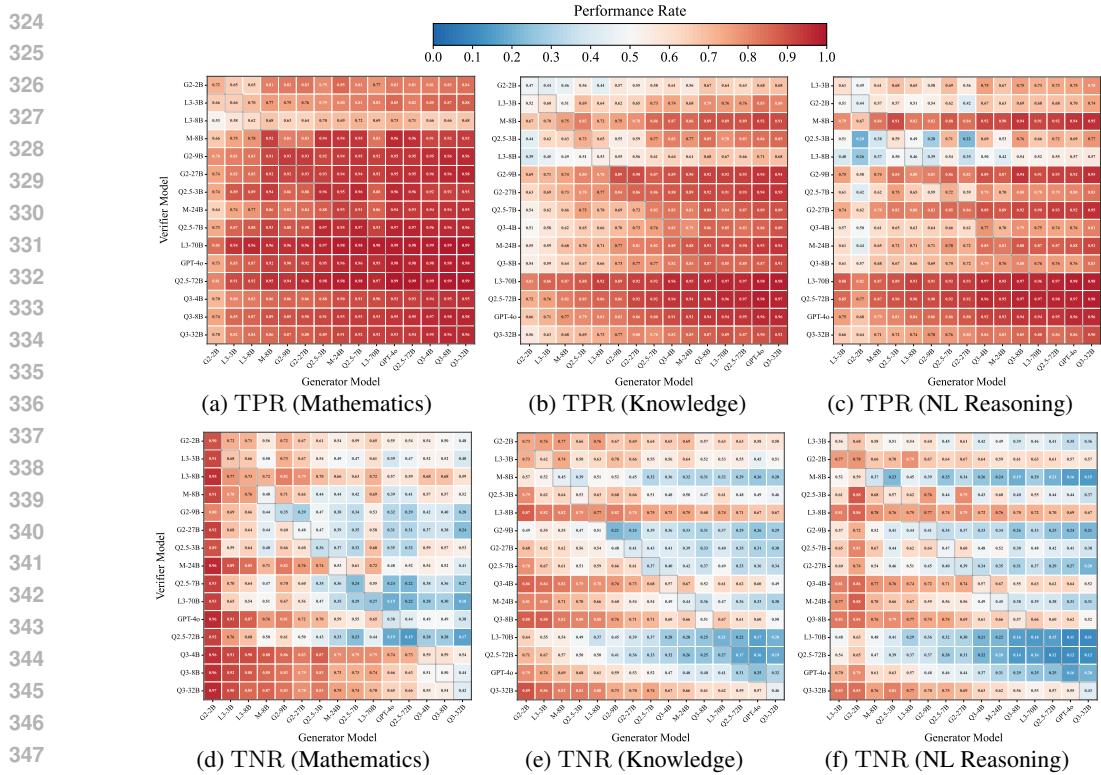


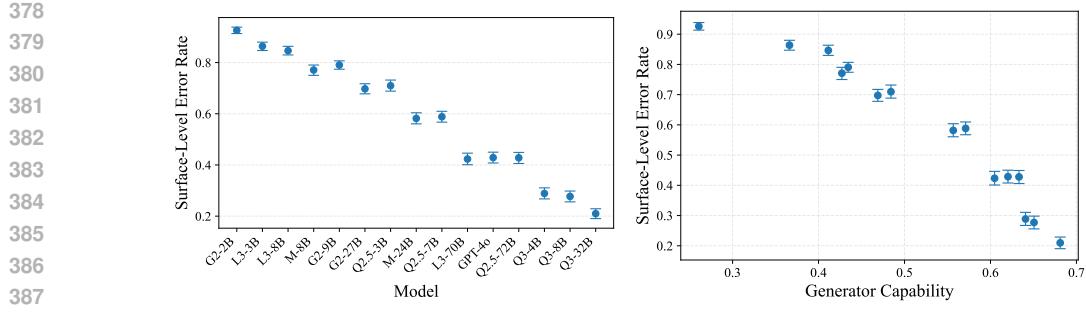
Figure 4: **Generator capability influences verifier performance of identifying incorrect responses.** Heatmaps show (a-c) TPR and (d-f) TNR when pairing 15 verifier models (rows) with 15 generator models (columns). Rows and columns are ordered by models' generation capability computed on all problems of each domain. Values indicate mean performance over the evaluation subset.

verifiers. These results show that generator capability strongly modulates the detection of incorrect responses. In Appendix C.2, we show that this finding generalizes to reasoning models.

We interpret this phenomenon first through case studies in Figure 33. Strong generators produce internally consistent reasoning chains where early mistakes (e.g., missed cases) propagate coherently, yielding well-structured but incorrect solutions that cause the verifier false positives. Weak generators produce surface-level errors such as self-contradictions, facilitating verifier rejection. We support this explanation further with a large-scale analysis. We use an independent LLM-as-judge to determine whether a generator’s CoT contains surface-level errors. We examine three categories of surface-level errors: (i) self-contradictions, (ii) basic arithmetic or factual mistakes, and (iii) incomplete responses lacking a final answer. The three prompts for detecting each error are included in Appendix A, and we use GPT-4.1-mini as the judge. We apply this analysis to 98,782 incorrect generator responses, the same ones used for TNR. For each problem, we randomly sample one incorrect response from each of the 15 generators and compute the proportion of these responses that contain surface-level errors. We perform 10,000 bootstrap resamples and report the mean and 95% CIs. The results in Figure 5 show a clear trend: as generator capability increases, the frequency of surface-level errors decreases. This confirms that our case study presents a systematic mechanism of response quality.

#### 4.3 HOW DOES VERIFIER GENERATION CAPABILITY IMPACT VERIFICATION?

We measure verifier generation capability and evaluate verification performance using balanced accuracy ( $\text{Acc}_{\text{bal}}$ ) on the entire test set. Each verifier is evaluated on responses from all generators, and we report results both averaged across all problems and stratified by problem difficulty. To characterize the relationship between generation capability and verification performance, we employ locally weighted regression (Cleveland, 1979) with a bandwidth of 0.6 to fit nonparametric curves. We compare  $R^2$  values between nonparametric and linear fits to assess linearity. We also report the Pearson correlation coefficient (Benesty et al., 2009) as another measure of linear correlation.



(a) Surface-level Error Rate vs. Model Name (b) Surface-level Error Rate vs. Model Capability

Figure 5: **Interpreting RQ2 finding by analyzing surface-level errors in generators’ solutions.**  
 (a) The  $x$ -axis lists generator models sorted by generation capability; the  $y$ -axis shows the percentage of responses containing surface-level errors. (b) The  $x$ -axis shows the capability values of the same generators; the  $y$ -axis is the same as in (a). Error bars indicate 95% CIs.

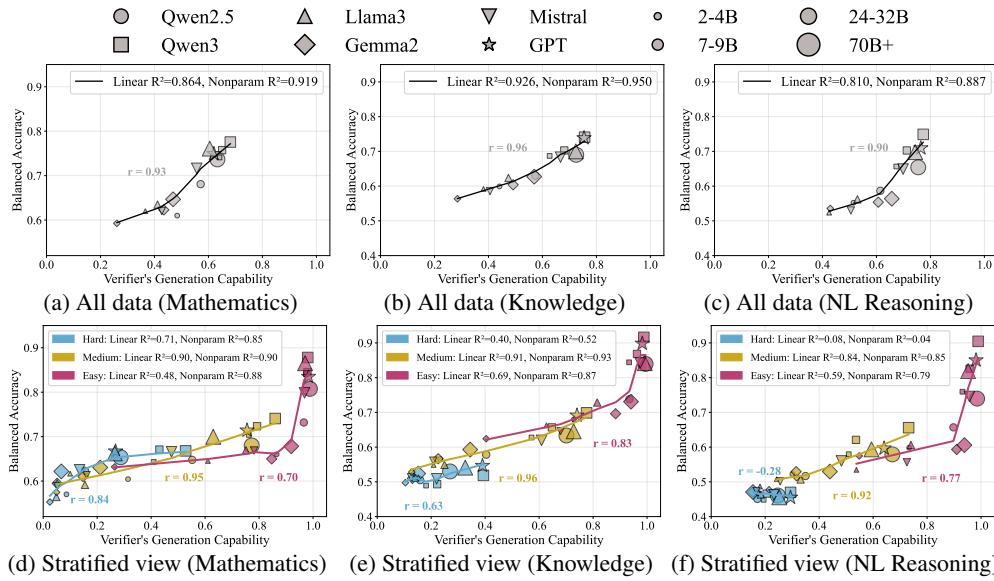


Figure 6: **Distinct correlation forms between verification performance and generation capability.**  
 Solid lines represent nonparametric fits to the data;  $r$  indicates the Pearson correlation coefficient. (a-c) Averaged across all problems, verifier generation capability exhibits a strong linear correlation with balanced accuracy. (d-f) When stratified by problem difficulty, distinct correlation patterns emerge.

Figures 6a to 6c show a strong overall correlation between verifier generation capability and verification accuracy, with NL reasoning showing less linearity than other domains. This result is consistent with prior work showing that evaluator accuracy tends to track the evaluator’s task performance, with the relationship appearing nearly linear. While this global trend validates findings in prior work (Chen et al., 2025c; Tan et al., 2025), a closer inspection of the trend reveals highly non-linear regimes.

**Verifier generation capability influences verification accuracy differently based on problem difficulty.** Stratified analysis reveals regime-dependent correlation with phase-transition behavior. We partition problems into 10 equal-width bins by difficulty  $d(x)$  and analyze three representative intervals: *hard* [0.1, 0.3], *medium* [0.4, 0.5], and *easy* [0.8, 0.9] in Figures 6e, 6f and 23a. For hard problems (blue), verification accuracy shows minimal improvement with increasing capability. Mathematics plateaus around 0.65 accuracy after initial gains, while other domains remain flat throughout. Notably, verifiers achieve below-random accuracy on hard NL Reasoning problems, which we analyze in Appendix C.3. Medium problems (yellow) exhibit steady accuracy increases with capability, indicating strong linear relationships. This is confirmed by linear and nonparametric fits that yield nearly identical  $R^2$  values, with  $r > 0.9$ . Easy problems exhibit a threshold effect

432 at the  $x$ -axis around 0.9: below this threshold, the relationship is linear; above it, small capability  
 433 improvements yield large verification gains. Hard and easy regimes show nonlinearity with  
 434 nonparametric  $R^2$  exceeding linear  $R^2$  by 0.1–0.2 and  $r < 0.85$ . The exception is NL Reasoning  
 435 on hard problems, where both fits yield near-zero  $R^2$ , indicating no meaningful capability-accuracy  
 436 relationship. Appendix C.4 provides additional results, including analysis across all difficulty  
 437 intervals (Figure 14) and discusses implications of these findings.

## 438 439 5 APPLICATION TO TEST-TIME SCALING (TTS)

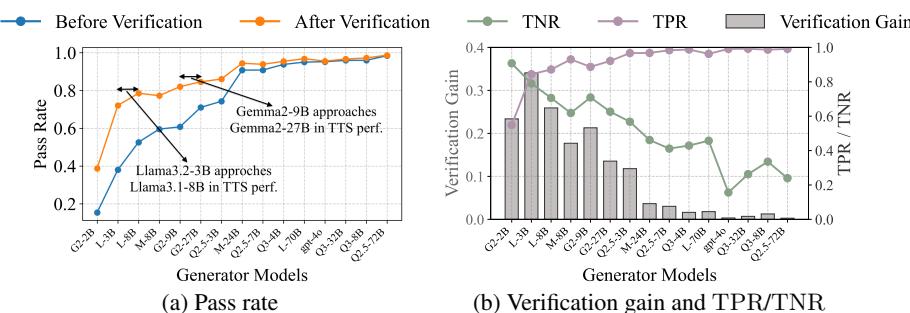
440 Our analysis in Section 4 is conducted with verification itself as the end goal. However, our findings  
 441 have direct implications for TTS. We analyze two research questions in TTS settings that naturally  
 442 arise out of our previous findings, and present our results below:

- 443 • **RQ4: Given a fixed verifier, can a weak generator match a stronger generator in TTS?** (Sec-  
 444 tion 5.1) Weak generators can nearly match stronger generators’ post-verification performance.  
 445 Verification gains peak at weak-medium generators by achieving a high error detection rate (TNR)  
 446 while maintaining a moderately high correctness recognition rate (TPR).
- 447 • **RQ5: Can weak verifiers match the gains of strong verifiers in TTS?** (Section 5.2) The  
 448 verification gain gap between weak and strong verifiers narrows at both low and high problem  
 449 difficulty extremes, and when using strong generators.

450 The following sections present results on the Mathematics domain, with complete results across all  
 451 three domains in Appendices C.5 and C.6.

### 452 453 5.1 CAN WEAK GENERATORS MATCH STRONGER GENERATORS IN TTS?

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



466 **Figure 7: TTS performance before and after verification when varying generator strength.**  
 467 Results are reported on problems with difficulty in the range [0.7, 0.8) from the Mathematics  
 468 domain, including 181 problems. (a) Pass rate before (blue) and after (orange) adding a fixed  
 469 verifier (GPT-4o). The generators in the  $x$ -axis are ordered from weaker (left) to stronger (right)  
 470 by generation capability measured on the problem subset. (b) Bar chart shows the verification gain  
 471  $\Delta\hat{p}_V$  (left  $y$ -axis) for each generator. Lines show the verifier’s TNR and TPR (right  $y$ -axis).

472 **Verification gain peaks for weak–medium generators, enabling them to approach stronger  
 473 models post-verification.** As shown in Figure 7a, weak generators start with much lower pass  
 474 rates but improve dramatically after verification, reaching levels comparable to larger models. For  
 475 example, Gemma2-9B starts from a significantly lower baseline but, after verification, achieves a  
 476 pass rate nearly matching Gemma2-27B. The performance gap shrinks from 10.3% to 2.5%, closing  
 477 75.7% of the original difference. Figure 7b explains this phenomenon: as generator strength increases  
 478 (left to right), TNR decreases sharply while TPR rises only modestly, consistent with **RQ2** findings.  
 479 Consequently, verification gain (gray bars) peaks at weak–medium generators. These generators  
 480 achieve high TNR for effective error filtering while maintaining moderate TPR to preserve correct  
 481 responses. For the strongest generators, errors become harder to identify, causing TNR decline  
 482 and limiting gains. In Appendix C.5, we show the findings derived from the Mathematics domain  
 483 generalize well to two other domains in Figure 15. We also provide additional evidence confirming  
 484 the generalizability of these findings. First, verification gains peak for weak–medium generators  
 485 across a broad range of problem difficulties ( $d(x) \geq 0.3$ ) in all domains (Figures 16 to 18). Second,  
 486 performance gaps of most weak and strong model pairs can be reduced by verification when  
 487 evaluated on the entire domain datasets (Figure 19), mostly achieving 30–50% reduction.

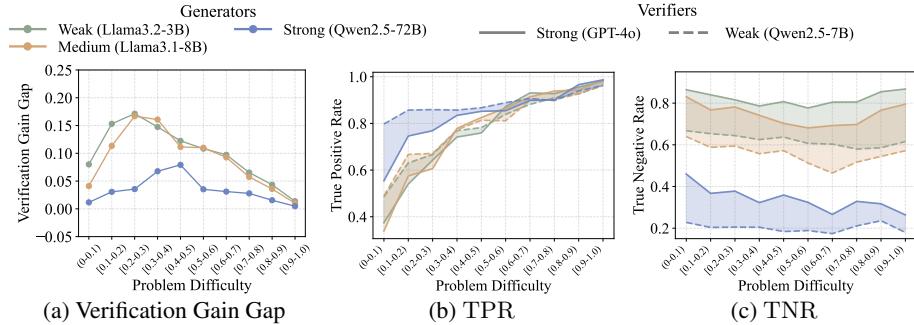


Figure 8: **Weak vs. strong verifiers under varying problem difficulty and generator strength.** The  $x$ -axis shows problem difficulty, ordered from hardest to easiest, measured relative to each generator. (a) Verification gain gap between strong and weak verifiers when applied to weak, medium, and strong generators. (b) TPR increases as problems become easier; shadow band indicates TPR gap between two verifiers. (c) As generators strengthen, TNR decreases overall and the TNR gap (shadow band) narrows. Results are from the Mathematics domain.

## 5.2 CAN WEAK VERIFIERS MATCH THE GAINS OF STRONG VERIFIERS IN TTS?

We analyze the verification gain gap between a strong verifier (GPT-4o) and a weaker one (Qwen2.5-7B) across problem difficulty ranges and generator strengths. The verification gain  $\Delta\hat{p}_V$  is defined in Section 3 and the gap between verifier is  $\Delta\hat{p}_{V_{\text{strong}}} - \Delta\hat{p}_{V_{\text{weak}}}$ . Our goal is to identify when this gap narrows, as such regimes suggest weak verifiers can substitute for strong ones.

**The gap narrows on the extremes of problem difficulty** As shown in Figure 8a, the verification gain gap shrinks as problems become easier, which corresponds to the rising TPR for both weak and strong verifiers seen in Figure 8b. This aligns with our **RQ1** findings that easier problems improve TPR for all verifiers. Even weak verifiers reliably recognize correct responses on easy problems, leaving little room for strong ones to provide additional benefit. At the opposite extreme, the gap also narrows on the hardest problems. As discussed in **RQ3** and shown in Figures 21d to 21f, increasing verifier generation capability (or scaling up to larger models) fails to improve verification accuracy on hard problems, resulting in only marginal performance differences between weak and strong verifiers.

**The gap narrows as generators become stronger.** Figure 8 shows that increasing generator capability reduces the difference between weak and strong verifiers. This is consistent with **RQ2** (Section 4.2), where we observed that the verifier’s TNR decreases as the generator capability increases. As both weak and strong verifiers experience lower TNR, the gap between them also shrinks, shown as the narrowing shaded band in Figure 8c).

The results here are obtained from the Mathematics domain, and we show the findings generalize to two other domains in Appendix C.6. In regimes of very easy/hard problems or when evaluating strong generators’ responses, weak verifiers provide gains to TTS performance comparable to strong verifiers. However, these convergence regimes coincide with minimal verification benefit overall. Figures 21a to 21c shows verification gains drop to 0.1 or below for both verifiers on easy and hard problems, verification on strong generators yields peak gains of only 0.1, precisely where the gap narrows. Thus, while weak and strong verifiers converge in these regimes, this convergence occurs where both provide minimal practical value. This reveals that scaling verifiers from 7B models to GPT-4o fails to overcome fundamental verification challenges, with GPT-4o providing limited improvement over small open-source models in the identified regimes.

## 6 CONCLUSION

We study LLM verification across problem difficulty, generator capability, and verifier generation capability, revealing that verification success depends on their interactions. We find that problem difficulty primarily shapes correct solution recognition, generator capability influences error detectability, and verifier generation capability correlates with verification in problem difficulty-dependent patterns. We examine the implications of these findings for verification deployment in TTS, identifying both opportunities and limitations. Stronger generators may not be necessary, as weaker generators can approach the post-verification performance of stronger ones when paired with a fixed verifier. This suggests potential for strategic model pairing that could reduce computational costs in verifier-based TTS methods. Our results also identify regimes where investing in larger verifiers yields no benefit, such as when evaluating responses from strong generators or problems at difficulty extremes.

540 ETHICS STATEMENT  
541

542 This work exclusively evaluates large language models on publicly available academic benchmarks  
543 containing mathematical reasoning, knowledge, and natural language problems with objective ground-  
544 truth answers. All experiments involve automated evaluation of model outputs without human subject  
545 participation. The datasets used are established research benchmarks designed for educational  
546 problem-solving tasks. Our study aims to understand verification dynamics to improve the compu-  
547 tational efficiency of LLM systems, posing no ethical concerns regarding privacy, harmful content  
548 generation, or potential misuse.

549 REPRODUCIBILITY STATEMENT  
550

551 We provide comprehensive details to ensure reproducibility of our findings. All experiments use  
552 publicly available datasets and open-source/commercial LLMs. We specify the model names, versions,  
553 dataset sources, and inference hyperparameters in Section 3.2. Complete prompt templates for both  
554 generation and verification tasks are provided in Appendix A. The mathematical formulations of all  
555 metrics, along with estimation procedures and aggregation methods, are formally defined and clearly  
556 described in Section 3.1 and Appendix B.

557 REFERENCES  
558

559 Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation.  
560 In *The Thirteenth International Conference on Learning Representations*, 2025.

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

563 Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In  
564 *Noise reduction in speech processing*, pp. 1–4. Springer, Berlin, Heidelberg, 2009.

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

568 Ding Chen, Qingchen Yu, Pengyuan Wang, Wentao Zhang, Bo Tang, Feiyu Xiong, Xinchi Li,  
569 Minchuan Yang, and Zhiyu Li. xverify: Efficient answer verifier for reasoning model evaluations.  
570 *arXiv preprint arXiv:2504.10481*, 2025a.

571 Nuo Chen, Zhiyuan Hu, Qingyun Zou, Jiaying Wu, Qian Wang, Bryan Hooi, and Bingsheng He.  
572 Judgeirm: Large reasoning models as a judge. *arXiv preprint arXiv:2504.00050*, 2025b.

573 Qiguang Chen, Libo Qin, Jiaqi WANG, Jingxuan Zhou, and Wanxiang Che. Unlocking the capabilities  
574 of thought: A reasoning boundary framework to quantify and optimize chain-of-thought. In *The  
575 Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

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

578 Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to  
579 self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

580 Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang,  
581 Denghui Zhang, Tong Zhang, et al. Rm-r1: Reward modeling as reasoning. *arXiv preprint  
582 arXiv:2505.02387*, 2025d.

583 W.S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the  
584 American Statistical Association*, 74(368):829–836, 1979. doi: 10.1080/01621459.1979.10481038.

585 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser,  
586 Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John  
587 Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*,  
588 2021.

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

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

602 Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James  
 603 Coady, David Peng, Yujie Qiao, Luke Benson, et al. Folio: Natural language reasoning with  
 604 first-order logic. *arXiv preprint arXiv:2209.00840*, 2022.

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

609 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,  
 610 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*,  
 611 2021.

612 Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large  
 613 language models can self-improve. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceed-  
 614 ings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1051–  
 615 1068, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/  
 616 2023.emnlp-main.67. URL <https://aclanthology.org/2023.emnlp-main.67/>.

617 Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song,  
 618 and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth  
 619 International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=IkmD3fKBPQ>.

620 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-  
 621 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint  
 622 arXiv:2410.21276*, 2024.

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

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

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

631 Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli,  
 632 Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via  
 633 reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

634 H. Kydlíček. Math-verify: Math verification library, 2025. URL <https://github.com/huggingface/math-verify>.

635 Sungjae Lee, Hyejin Park, Jaechang Kim, and Jungseul Ok. Semantic exploration with adaptive  
 636 gating for efficient problem solving with language models. *arXiv preprint arXiv:2501.05752*,  
 637 2025.

638 Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ra-  
 639 masesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative  
 640 reasoning problems with language models. *Advances in neural information processing systems*,  
 641 35:3843–3857, 2022.

648 Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul,  
 649 Longhui Yu, Albert Q. Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths  
 650 with 860k pairs of competition math problems and solutions. Hugging Face repository, 2024.  
 651 Available at <https://huggingface.co/datasets/AI-MO/NuminaMath-CoT>.  
 652

653 Ruosen Li, Teerth Patel, and Xinya Du. Prd: Peer rank and discussion improve large language model  
 654 based evaluations. *arXiv preprint arXiv:2307.02762*, 2023.

655 Shalev Lifshitz, Sheila A. McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time  
 656 compute with multiple verifiers. In *Second Conference on Language Modeling*, 2025. URL  
 657 <https://openreview.net/forum?id=LriQ3NY9uL>.  
 658

659 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan  
 660 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth*  
 661 *International Conference on Learning Representations*, 2023.

662 Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Binqing Qi, Wanli Ouyang, and Bowen  
 663 Zhou. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *arXiv preprint*  
 664 *arXiv:2502.06703*, 2025a.

665 Shudong Liu, Hongwei Liu, Junnan Liu, Linchen Xiao, Songyang Gao, Chengqi Lyu, Yuzhe Gu,  
 666 Wenwei Zhang, Derek F Wong, Songyang Zhang, et al. Compassverifier: A unified and robust  
 667 verifier for llms evaluation and outcome reward. *arXiv preprint arXiv:2508.03686*, 2025b.

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

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

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

676 Yujun Mao, Yoon Kim, and Yilun Zhou. Champ: A competition-level dataset for fine-grained analyses  
 677 of llms' mathematical reasoning capabilities. In *Findings of the Association for Computational*  
 678 *Linguistics ACL 2024*, pp. 13256–13274, 2024.

679 Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari,  
 680 Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, Joseph E. Gonzalez,  
 681 Matei Zaharia, and Ion Stoica. Why do multiagent systems fail? In *ICLR 2025 Workshop on*  
 682 *Building Trust in Language Models and Applications*, 2025. URL <https://openreview.net/forum?id=wM521FqPvI>.

683 David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani,  
 684 Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In  
 685 *First Conference on Language Modeling*, 2024.

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

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

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

702 Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute  
 703 optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth Inter-*  
 704 *national Conference on Learning Representations*, 2025. URL [https://openreview.net/](https://openreview.net/forum?id=4FWAwZtd2n)  
 705 [forum?id=4FWAwZtd2n](https://openreview.net/forum?id=4FWAwZtd2n).

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

709 Shaoning Sun, Jiachen Yu, Zongqi Wang, Xuewei Yang, Tianle Gu, and Yujiu Yang. S2j: Bridg-  
 710 ing the gap between solving and judging ability in generative reward models. *arXiv preprint*  
 711 *arXiv:2509.22099*, 2025.

712 Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Yuan Tang, Alejandro Cuadron, Chenguang  
 713 Wang, Raluca Popa, and Ion Stoica. Judgebench: A benchmark for evaluating LLM-based  
 714 judges. In *The Thirteenth International Conference on Learning Representations*, 2025. URL  
 715 [https://openreview.net/](https://openreview.net/forum?id=G0dksFayVq)[forum?id=G0dksFayVq](https://openreview.net/forum?id=G0dksFayVq).

717 Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya  
 718 Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al.  
 719 Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*,  
 720 2024.

721 Qwen Team. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

723 Han Wang, Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. Soft self-consistency improves  
 724 language model agents. In *Proceedings of the 62nd Annual Meeting of the Association for*  
*Computational Linguistics (Volume 2: Short Papers)*, 2024a.

726 PeiFeng Wang, Austin Xu, Yilun Zhou, Caiming Xiong, and Shafiq Joty. Direct judgement preference  
 727 optimization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language*  
*Processing*, November 2025.

729 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha  
 730 Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language  
 731 models. In *The Eleventh International Conference on Learning Representations*, 2023. URL  
 732 [https://openreview.net/](https://openreview.net/forum?id=1PL1NIMMrw)[forum?id=1PL1NIMMrw](https://openreview.net/forum?id=1PL1NIMMrw).

733 Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming  
 734 Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-  
 735 task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:  
 95266–95290, 2024b.

738 Jason Wei. The asymmetry of verification and verifier’s law. [https://www.jasonwei.net/](https://www.jasonwei.net/blog/asymmetry-of-verification-and-verifiers-law)  
 739 [blog/asymmetry-of-verification-and-verifiers-law](https://www.jasonwei.net/blog/asymmetry-of-verification-and-verifiers-law), 2025.

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

744 Austin Xu, Yilun Zhou, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. J4r: Learning to judge  
 745 with equivalent initial state group relative policy optimization. *arXiv preprint arXiv:2505.13346*,  
 746 2025.

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

750 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang  
 751 Gao, Chengan Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*,  
 752 2025.

754 Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1,  
 755 grade-school math and the hidden reasoning process. In *The Thirteenth International Conference*  
*on Learning Representations*, 2025.

756 Weihao Yu, Zihang Jiang, Yanfei Dong, and Jiashi Feng. Reclor: A reading comprehension dataset  
757 requiring logical reasoning. *arXiv preprint arXiv:2002.04326*, 2020.

758

759 Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal.  
760 Generative verifiers: Reward modeling as next-token prediction. In *The 4th Workshop on Mathe-*  
761 *matical Reasoning and AI at NeurIPS'24*, 2024. URL [https://openreview.net/forum?](https://openreview.net/forum?id=CxHRoTLmPX)  
762 [id=CxHRoTLmPX](https://openreview.net/forum?id=CxHRoTLmPX).

763 Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal.  
764 Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International*  
765 *Conference on Learning Representations*, 2025.

766

767 Yilun Zhou, Austin Xu, PeiFeng Wang, Caiming Xiong, and Shafiq Joty. Evaluating judges as  
768 evaluators: The JETTS benchmark of LLM-as-judges as test-time scaling evaluators. In *Forty-*  
769 *second International Conference on Machine Learning*, 2025. URL [https://openreview.](https://openreview.net/forum?id=CgJEHynkJt)  
770 [net/forum?id=CgJEHynkJt](https://openreview.net/forum?id=CgJEHynkJt).

771 Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbulin, Yunyang Xiong,  
772 Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. Agent-as-a-judge:  
773 Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810 APPENDIX  
811812 THE USE OF LARGE LANGUAGE MODELS (LLMs)  
813814 LLMs were used exclusively as writing assistance tools in preparing this manuscript. Specifically,  
815 we employed LLMs for grammar checking. An LLM-based image generation tool was used to  
816 create the robot caricature in Figure 1. All research ideation, experimental design, analysis, and  
817 scientific conclusions are entirely the work of the authors. The LLMs played no role in the conception  
818 of research questions, methodology development, or interpretation of results. Authors take full  
819 responsibility for all content in this paper, including any text refined with LLM assistance.  
820821 A PROMPT TEMPLATES  
822823 **Response Generation Prompt.** Here, we provide the prompts to generate model responses to  
824 questions from three domains. For each model, we use its default system prompt as specified in the  
825 model documentation.  
826

## 827 Mathematical datasets

828  
829     ### User Prompt  
830     {problem}  
831     Please reason step by step, and put your final answer within \boxed{{}}.  
832

## 833 Knowledge: MMLU-Pro

834  
835     ### User Prompt  
836     The following are multiple choice questions (with answers) about {category}. Think step  
837     by step and then output the answer in the format of "The answer is (X)" where X is the  
838     correct letter choice.  
839     Question: {question}  
840     Options:  
841     {options}  
842     Answer: Let's think step by step.  
843

## 844 Natural Language Reasoning: ReClor

845  
846     ### User Prompt  
847     Answer the following multiple choice question. The last line of your response should be  
848     of the following format: 'Answer: \$LETTER' (without quotes) where \$LETTER is one of  
849     ABCD. Think step by step before answering.  
850     {context}  
851     {question}  
852     {choices}  
853     Output your answer strictly following this format:  
854     Reasoning: <step-by-step reasoning>  
855     Answer: <your choice>  
856

## 857 Natural Language Reasoning: FOLIO

858  
859     ### User Prompt  
860     {question}  
861     Based on the above, can the following be inferred?  
862     {conclusion}  
863

```

864
865     Think step-by-step before outputting your answer, which must be "Yes", "No", or
866     "Uncertain"
867
868     Output your answer strictly following this format:
869     Reasoning: Your reasoning here
870     Answer: Yes or No or Uncertain

```

### 871 Natural Language Reasoning: GPQA Diamond

```

872
873     ### User Prompt
874     Answer the following multiple choice question. The last line of your response should be
875     of the following format: 'Answer: $LETTER' (without quotes) where $LETTER is one of
876     ABCD. Think step by step before answering.
877
878     Question: {question}
879     Options:
880     {choices}
881
882     Output your answer strictly following this format:
883     Reasoning: <step-by-step reasoning>
884     Answer: <your choice>

```

885  
**886 Verification Evaluation Prompt.** Below, we present the prompt template used to evaluate verification  
 887 performance.

```

888
889     ### System Prompt
890     Please act as an impartial judge and evaluate the correctness of the response provided
891     by an AI assistant to the user prompt displayed below. You will be given the assistant's
892     response.
893
894     When evaluating the assistant's response, identify any mistakes or inaccurate
895     information. Be as objective as possible. Avoid any biases, such as order of responses,
896     length, or stylistic elements like formatting.
897
898     Before providing an your final verdict, think through the judging process and output
899     your thoughts as an explanation
900
901     After providing your explanation, you must output only one of the following choices as
902     your final verdict with a label:
903
904     1. The response is correct: [[Correct]]
905     2. The response is incorrect: [[Incorrect]]
906
907     Use the following template:
908     Explanation: Your detailed thought process as an explanation.
909     Verdict: [[Correct]] or [[Incorrect]].
910
911     ### User Prompt
912     <|User Prompt|>
913     {question}
914
915     <|The Start of Assistant's Answer|>
916     {response}
917     <|The End of Assistant's Answer|>

```

918  
**919 Verification Evaluation Prompt (Solve-then-Verify).** Below, we present a variant of the prompt  
 920 template used to evaluate verification performance. This template is adapted from the idea of Chen  
 921 et al. (2025d); Sun et al. (2025) that prompts the model to solve the problem first and use its own  
 922 solution as a reference for verification.

```

923
924     ### System Prompt
925     Please act as an impartial judge and evaluate the correctness of the response provided
926     by an AI assistant to the user prompt displayed below. You will be given the assistant's
927     response.
928
929     First, you MUST solve the question yourself and put your final answer following the
930     format requested in <|User Prompt|>. Provide your own solution with final answer before
931

```

```

918
919 proceeding to the evaluation. When evaluating the candidate's response, you MUST refer
920 to your own solution.
921 Be as objective as possible. Avoid any biases, such as order of responses, length, or
922 stylistic elements like formatting.
923 After providing your explanation, you must output only one of the following choices as
924 your final verdict with a label:
925 1. The response is correct: [[Correct]]
926 2. The response is incorrect: [[Incorrect]]
927 Use the following template:
928 Solution: Your own reasoning and final answer to the problem.
929 Explanation: Your detailed thought process as an explanation.
930 Verdict: [[Correct]] or [[Incorrect]].
```

```

931     ### User Prompt
932     <|User Prompt|>
933     {question}
934
935     <|The Start of Assistant's Answer|>
936     {response}
937     <|The End of Assistant's Answer|>
```

937 **Verification Evaluation Prompt (Step-by-Step).** Below, we adopt a variant of the verification  
 938 template from Zhang et al. (2024). This template only has a user prompt.

```

939
940
941     ### User Prompt
942     You are a math teacher. Grade the Solution, verifying correctness step by step.
943     At the end of the Solution verification, when you give your final grade, write it in the
944     form "Verification: Is the answer correct (Yes/No)? X", where X is either \boxed{{Yes}}
945     or \boxed{{No}}.
946     Question: {question}
947     Solution: {response}
```

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

```

951
952
953     ### User Prompt
954     Given a math problem, its correct answer, and the model's generated answer, determine if
955     the model's generated answer is correct.
956
957     VALIDATION CRITERIA:
958     1. Identify the final answer, which is usually put inside \boxed{{answer}} or
959     **answer**.
960     2. The answer must be mathematically equivalent to the correct answer
961     3. The answer must be complete with a clear final result
962     4. The answer must not just contain similar numbers - it must reach the correct
963     conclusion
964     5. If the generated answer contains multiple different final answers or is ambiguous
965     about which is the final answer, mark it as 'False'
966
967     IMPORTANT: Just having the same numbers as the ground truth is NOT sufficient - the
968     model must actually solve the problem correctly and provide the correct final answer in
969     the designated format.
970
971     Respond with 'True' if the answer is correct and complete, and 'False' if it is
972     incorrect or incomplete.
973     Directly provide your judgement 'True' or 'False' without any other description.
974
975     Problem: {problem}
976     Correct Answer: {ground_truth_answer}
977     Model's Generated Answer: {model_response}
978     Your judgement:
```

**Verification Analysis Prompt for Detecting the Solving Behavior.** Below we present the prompt template used to analyze if verification CoTs contain solving behavior. This prompt asks the verifier to solve the problem first and then

972     **Verification Analysis Prompt for Detecting the Solving Behavior.** Below we present the pro  
973     template used to analyze if verification CoTs contain solving behavior. This prompt asks the veri  
974     to solve the problem first and then  
975  
976  
977     ### System Prompt  
978     Please act as an impartial analyzer and determine whether the AI assistant, when  
979     evaluating a candidate response, generates its own solution (full or partial) as a  
980     reference - either explicitly or implicitly.  
981  
982     \*\*Key Indicators that the assistant GENERATED its own solution [[Yes]]:\*\*  
983     - The assistant determines what the correct answer should be (even implicitly)  
984     - The assistant works through the problem logic independently (e.g., "the premises  
985     actually say X", "what can be inferred is Y")  
986     - The assistant constructs its own interpretation of what follows from the problem  
987     statement  
988     - The assistant makes claims about what the "correct reasoning" would be  
989  
990     \*\*Key Indicators that the assistant ONLY INSPECTED the candidate [[No]]:\*\*  
991     - The assistant only points out errors in the candidate's reasoning flow without  
992     determining the correct answer  
993     - The assistant only identifies missing steps or unsupported leaps in the candidate's  
994     reasoning chain  
995     - The assistant only verifies consistency within the candidate's own reasoning chain  
996     - Focus is purely on "the candidate failed to justify X" rather than "X is actually Y"  
997  
998     \*\*Important:\*\* Even implicit or partial solutions count as generating a solution. If the  
999     assistant reveals what it believes to be true/false about the problem, it has generated  
1000     a solution.  
1001  
1002     \*\*Example 1:\*\*  
1003  
1004     <|User Prompt|>  
1005     {example1\_question}  
1006  
1007     <|The Start of Candidate Response|>  
1008     {example1\_response}  
1009     <|The End of Candidate Response|>  
1010  
1011     <|The Start of Assistant's Evaluation|>  
1012     {example1\_evaluation}  
1013     <|The End of Assistant's Evaluation|>  
1014  
1015     \*\*Expected Analysis:\*\*  
1016     {example1\_analysis}  
1017  
1018     ---  
1019  
1020  
1021     Now analyze the following case. Look carefully for ANY statement where the assistant  
1022     determines what is actually true, correct, valid, or inferable - these indicate the  
1023     assistant generated its own solution.  
1024  
1025     Use the following template:  
1026     Explanation: Your reasoning for why the assistant did or did not generate its own  
1027     solution. Quote specific phrases that reveal solution generation.  
1028     Verdict: [[Yes]] or [[No]]  
1029  
1030     ### User Prompt  
1031     <|User Prompt|>  
1032     {question}  
1033  
1034     <|The Start of Candidate Response|>  
1035     {response}  
1036     <|The End of Candidate Response|>  
1037  
1038     <|The Start of Assistant's Evaluation|>  
1039     {evaluation}  
1040     <|The End of Assistant's Evaluation|>

**Verification Analysis Prompt for Detecting the Reasoning Mistakes.** Below, we present the prompt template used to analyze if verification CoTs contain any basic reasoning mistakes.

### System Prompt

```

1026
1027 You are an impartial checker. Your task is to determine whether the evaluator made a
1028 mistake in its reasoning when evaluating the candidate response.
1029
1030 You may use the provided gold_answer - whether it is a full explanation or just a final
1031 answer - as the reference for what is correct.
1032
1033 You MUST ignore the evaluator's final verdict entirely. Do NOT read it, do NOT interpret
1034 it, and do NOT allow it to influence your judgment in any way.
1035
1036 If the evaluator's reasoning contradicts the gold_answer, or the evaluator asserts
1037 incorrect mathematical or logical statements, then the evaluator's reasoning is
1038 incorrect.
1039
1040 If the evaluator's reasoning is consistent with the gold_answer and contains no
1041 incorrect claims, then the evaluator's reasoning is correct.
1042
1043 After reviewing the evaluation, output:
1044
1045 Explanation: A brief explanation of whether the evaluator's reasoning is correct or
1046 incorrect.
1047 Verdict: [[Correct]] or [[Incorrect]]
1048
1049 #### User Prompt
1050 <|User Prompt|>
1051 {question}
1052
1053 <|Correct Answer of the Problem|>
1054 {correct_answer}
1055
1056 <|The Start of Candidate Response|>
1057 {response}
1058 <|The End of Candidate Response|>
1059
1060 <|The Start of Assistant's Evaluation|>
1061 {evaluation}
1062 <|The End of Assistant's Evaluation|>

```

### Generator Response Analysis Prompt for Identifying Surface-level Error

```

1055 -----
1056 #### System prompt for checking self-contradictory error
1057 -----
1058 You are an impartial analyzer. You will be given a question and a candidate response.
1059 The candidate response is known to be incorrect.
1060
1061 Your ONLY task is to determine whether the candidate response is internally consistent.
1062
1063 #### Consistency Definitions ####
1064
1065 Definition of "Consistent":
1066 - The reasoning does not contradict itself.
1067 - The steps follow a coherent internal logic.
1068 - The response may contain wrong assumptions, missing cases, or wrong math,
1069 but it is still consistent if it does not conflict with itself.
1070
1071 Definition of "Not Consistent":
1072 - The candidate response contains an explicit or implicit contradiction.
1073 - It states mutually incompatible claims.
1074 - Its reasoning steps cannot simultaneously be true based on its own premises.
1075
1076 #### Strict Prohibitions ####
1077
1078 IMPORTANT:
1079 - DO NOT check correctness.
- DO NOT compare to the true answer.
- ONLY evaluate whether the reasoning contradicts itself.
1080
1081 Now analyze the following case. Respond using the format below:
1082
1083 Explanation: Briefly explain why the candidate response is internally consistent or
1084 inconsistent.
1085 Verdict: [[Yes]] (consistent) or [[No]] (not consistent)
1086
1087 -----
1088 #### System prompt for checking basic math error

```

```

1080
1081 -----
1082 You are an impartial analyzer. You will be given a question and a candidate response.
1083 The candidate response is known in advance to be incorrect.
1084
1085 Your ONLY task is to determine whether the candidate response contains
1086 basic calculation mistakes or basic mathematical knowledge errors.
1087
1088 #### What counts as a basic math error ####
1089 A response contains a basic math error if it includes:
1090 - Incorrect arithmetic (wrong addition, subtraction, multiplication, division).
1091 - Incorrect simplification or algebraic manipulation.
1092 - Incorrect fundamental math facts
1093 (e.g., parity mistakes, misidentifying primes, incorrect factorial values).
1094 - Incorrect counting in basic combinatorics (e.g., number of simple outcomes, pairs).
1095 - Incorrect use of basic probability rules (e.g., wrong multiplicative or additive rule).
1096
1097 #### What does NOT count as a basic math error ####
1098 A response does NOT contain a basic math error if:
1099 - All arithmetic steps are numerically correct,
1100 - Basic mathematical facts are used correctly,
1101 - Basic counting/probability rules are applied correctly,
1102 even if the overall reasoning is wrong,
1103 even if conclusions are incorrect,
1104 even if important steps are missing.
1105
1106 #### Strict Prohibitions ####
1107 IMPORTANT:
1108 - DO NOT evaluate correctness of the final answer.
1109 - DO NOT compare the candidate response to the true answer.
1110 - DO NOT judge consistency or high-level reasoning.
1111 - ONLY check for basic arithmetic or elementary math knowledge mistakes.
1112
1113 Use the following format:
1114
1115 Explanation: Briefly explain whether the candidate response contains basic calculation
1116 or math-knowledge errors.
1117 Verdict: [[Yes]] (contains a basic math error) or [[No]] (does not contain a basic math
1118 error)
1119
1120 -----
1121 #### System prompt for checking incompleteness error
1122 -----
1123 You are an impartial analyzer. You will be given a question and a candidate response.
1124 The candidate response is known in advance to be incorrect.
1125
1126 Your ONLY task is to determine whether the candidate response actually provides a final
1127 answer to the question.
1128
1129 #### What counts as providing a final answer (Verdict: [[Yes]]) ####
1130 A candidate response PROVIDES a final answer if:
1131 - It clearly states a specific final value, equation, or expression that answers the
1132 question,
1133 even if the value is wrong.
1134 - The final answer may be given in any clear format, such as:
1135 - Inside \boxed{...}
1136 - In LaTeX math, e.g., $\\frac{25}{36}$
1137 - As plain text, e.g., "The probability is 2/3."
1138 - In bold or marked as **Final Answer:** followed by a concrete number or expression.
1139
1140 #### SPECIAL CASE: Token Limit Cutoff ####
1141 If the response is clearly CUT OFF due to token limits (e.g., ends mid-sentence,
1142 mid-word,
1143 mid-equation, or stops abruptly), then:
1144 - You must treat this as the model ATTEMPTING to provide a final answer.
1145 - In this case, output Verdict: [[Yes]].
1146
1147 #### What counts as NOT providing a final answer (Verdict: [[No]]) ####
1148 A candidate response does NOT provide a final answer if:
1149 - It only gives high-level discussion, explanation, or strategy without stating a
1150 concrete result.
1151 - It says the answer is hard to compute, suggests using a calculator, or leaves the
1152 result as "you can now compute" without doing it.
1153 - It trails off with partial work (e.g., sets up an expression but never evaluates it or
1154 never clearly claims it as the final answer).
1155 - It only restates the problem, gives definitions, or discusses approaches without
1156 committing to an explicit outcome.
1157
1158 #### Strict Prohibitions ####
1159 IMPORTANT:

```

1134  
 1135     - DO NOT evaluate correctness of the final answer.  
 1136     - DO NOT compare the candidate response to the true answer.  
 1137     - DO NOT judge consistency or high-level reasoning.  
 1138     - ONLY check for providing a final answer.  
 1139  
 1140     Use the following format:  
 1141  
 1142     Explanation: Briefly explain whether the candidate response provides a final answer.  
 1143     Verdict: [[Yes]] (provides a final answer) or [[No]] (does not provide a final answer)

## B ADDITIONAL PRELIMINARIES AND SETUP

### B.1 DETAILS OF MATH PROBLEMS CORRECTNESS CHECK

1144  
 1145  
 1146  
 1147 Here we detail the evaluation procedure for establishing response correctness, including fallback  
 1148 methods. Ground-truth correctness is determined using Math-Verify (Kydlíček, 2025). If  
 1149 Math-Verify fails to parse an answer or returns incorrect, we recheck with other string-matching  
 1150 verifiers from open-source repositories lm-eval (Gao et al., 2024), Dr .GRPO (Liu et al., 2025c),  
 1151 and Qwen2.5-Math (Yang et al., 2024). We further apply GPT-4.1-mini and Qwen2.5-72B to con-  
 1152 duct reference-based evaluation and check the equivalence of the model prediction and ground-truth  
 1153 answers. The prompt template for LLM-based verification is provided in Appendix A.

### B.2 MODEL NAMING CONVENTIONS

1154  
 1155  
 1156 Throughout this paper, we use abbreviated model names in figures and tables to improve readability  
 1157 and space efficiency. Table 1 provides the complete mapping between abbreviations and full model  
 1158 names. All models referenced are instruction-tuned versions unless otherwise specified.

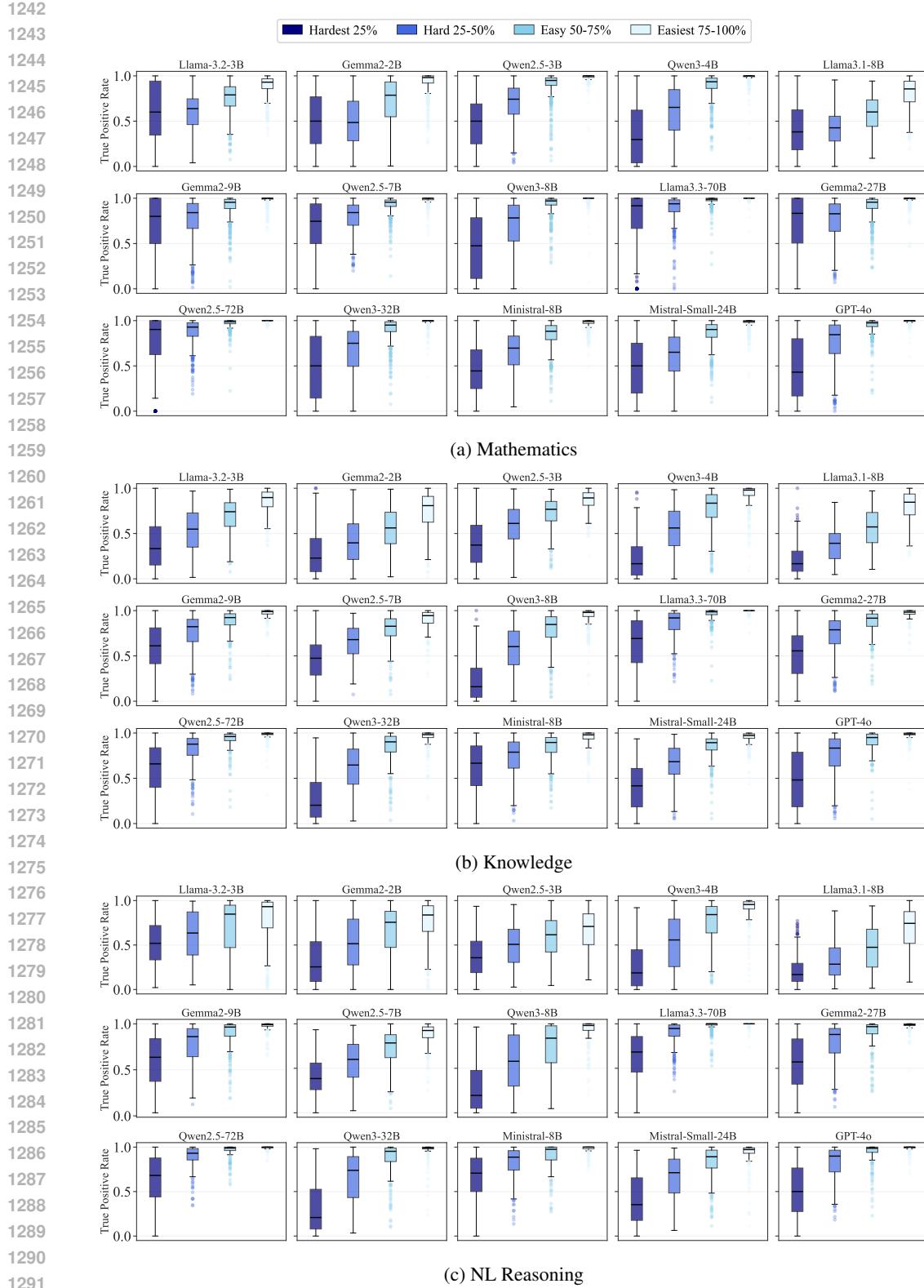
1159  
 1160 Table 1: Mapping between abbreviated model names used in figures and their full names. All models  
 1161 are instruction-tuned versions.

Abbreviation	Full Model Name	Abbreviation	Full Model Name
G2-2B	Gemma2-2B	Q3-4B	Qwen3-4B
G2-9B	Gemma2-9B	Q3-8B	Qwen3-8B
G2-27B	Gemma2-27B	Q3-32B	Qwen3-32B
L3-3B (L-3B)	Llama3.2-3B	M-8B	Minstral-8B
L3-8B (L-8B)	Llama3.1-8B	M-24B	Mistral-Small-24B
L3-70B (L-70B)	Llama3.3-70B	gpt-4o	GPT-4o
Q2.5-3B	Qwen2.5-3B		
Q2.5-7B	Qwen2.5-7B		
Q2.5-72B	Qwen2.5-72B		

### B.3 DETAILED EXPERIMENTAL SETUP OF RQ2

1174  
 1175  
 1176 Due to significant differences in generator capability, when measuring TPR, for some very difficult  
 1177 problems, none of the 64 responses sampled from a weak model are correct. To ensure fair evaluation  
 1178 unaffected by intrinsic problem difficulty, we exclude these problems and keep only those where  
 1179 every generator produces at least one correct response. We apply analogous filtering for TNR,  
 1180 keeping only problems where each generator produces at least one incorrect response. Beyond  
 1181 filtering problems, we also carefully balance how many responses we evaluate from each generator.  
 1182 As described in Section 3, we subsample 8 responses from each generator’s 64-sample pool for  
 1183 verification evaluation, aiming for 4 correct and 4 incorrect when possible. However, across these  
 1184 8-response subsets, stronger generators may have produced more correct responses than weaker  
 1185 ones. This would bias our metrics by creating different denominators per generator. To address this,  
 1186 we randomly select one correct response per problem from each generator’s 8-response pool when  
 1187 computing TPR (and analogously for TNR). We repeat this evaluation with random selections eight  
 1188 times and report the mean.

1188 **C ADDITIONAL RESULTS**  
11891190 **C.1 MORE DETAILS IN PROBLEM DIFFICULTY ANALYSIS**  
11911192 In Section 4.1, we show that problem difficulty primarily influences the verifier’s ability to recognize  
1193 correct responses. As discussed in the main paper, our analysis is conducted at two levels of  
1194 granularity: response level and problem level. Figure 2 shows results at the response level. Figures 9  
1195 and 10 show results at the problem level, summarizing the distribution of TPR and TNR across  
1196 difficulty quartiles. Together, these results confirm our main finding that problem difficulty strongly  
1197 correlates with TPR but has no systematic effect on TNR.  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241



**Figure 9: Problem difficulty correlates with verification TPR on per-problem level across three domains.** Each boxplot shows the distribution of per-problem TPR for 15 verifier models, grouped by difficulty quartiles. TPR exhibits a strong positive correlation with problem easiness: easier problems consistently yield higher and less variable TPR.

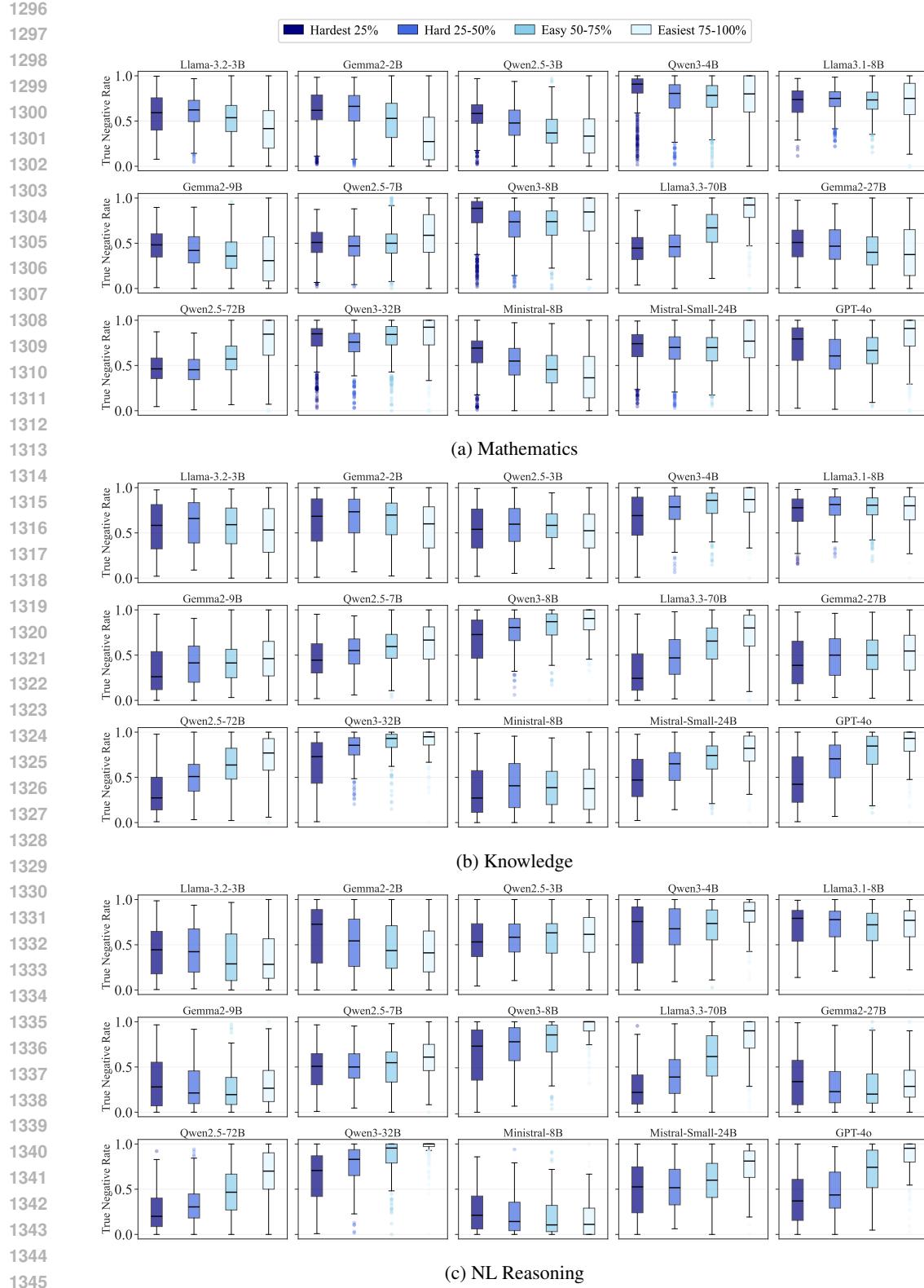


Figure 10: **Problem difficulty shows no systematic correlation with verification TNR on per-problem level across three domains.** Each boxplot shows the distribution of per-problem metrics for 15 verifier models, grouped by difficulty quartiles. TNR doesn't show obvious correlation with problem difficulty, exhibiting inconsistent trends across models.

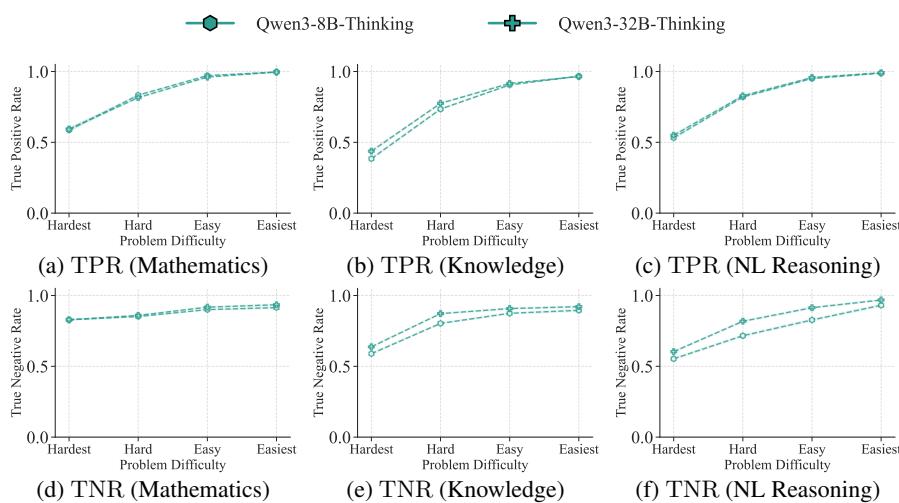
1350  
1351 C.2 ANALYSIS ON REASONING MODEL  
1352

1353 Our main analysis focuses on instruction-tuned models, which represent the typical setting for  
1354 verification systems in current practice, including recent judge models (Tan et al., 2025; Wang  
1355 et al., 2025) and verifier work (Liu et al., 2025b; Zhang et al., 2025). We prioritize models without  
1356 extensive CoT reasoning because verification often demands low-latency solutions, particularly for  
1357 reinforcement learning training and TTS applications where rapid evaluation is critical. However, a  
1358 recent trend involves training long-reasoning evaluators (Chen et al., 2025d; Whitehouse et al., 2025)  
1359 that generate extended CoT before making verification decisions. To examine whether our findings  
1360 generalize to this emerging paradigm, we include two reasoning models (Qwen3-8B-Thinking and  
1361 Qwen3-32B-Thinking<sup>2</sup>) and analyze how they perform across our research questions. These models  
1362 generate longer reasoning traces before producing binary verdicts, representing the state-of-the-art in  
1363 reasoning-enhanced verification.

1364 In Figures 11 and 12, we evaluate the conclusion of **RQ1** (Section 4.1). We observe that reasoning  
1365 models exhibit the same TPR pattern as instruction-tuned models: easier problems consistently  
1366 yield higher TPR across all three domains. This indicates that the fundamental relationship between  
1367 problem difficulty and correctness recognition persists with extended reasoning. However, reasoning  
1368 models exhibit a notable difference in TNR behavior. Unlike instruction-tuned models, where  
1369 TNR showed no systematic relationship with problem difficulty, both reasoning models demonstrate  
1370 improved TNR as problems become easier across all three domains. This pattern suggests that, with  
1371 extended reasoning, error detection becomes easier when problems become easier.

1372 In Figure 13, we evaluate the findings of **RQ2** (Section 4.2) on reasoning models and find that they  
1373 maintain the core patterns observed in instruction-tuned models. TPR remains consistently high with  
1374 mild increases as generator strength increases, while TNR decreases more significantly (goes from  
1375 red to white) with stronger generators. This indicates that the fundamental challenge of detecting  
1376 errors from capable generators persists despite enhanced reasoning capabilities.

1377 These findings demonstrate that reasoning models offer some advantages for error detection on easier  
1378 problems while preserving the core verification dynamics we identified. Problem difficulty continues  
1379 to govern correctness recognition, and generator capability primarily influences error detectability  
1380 across different verification paradigms.



1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
Figure 11: **Verification performance of reasoning models across problem difficulty at the per-response level.** TPR (a-c) and TNR (d-f) for Qwen3-8B-Thinking and Qwen3-32B-Thinking across difficulty quartiles in three domains. Both reasoning models show increasing TPR and TNR as problem difficulty decreases.

<sup>2</sup>We use the suggested sampling hyperparameter (temperature 0.6, top-p 0.95).

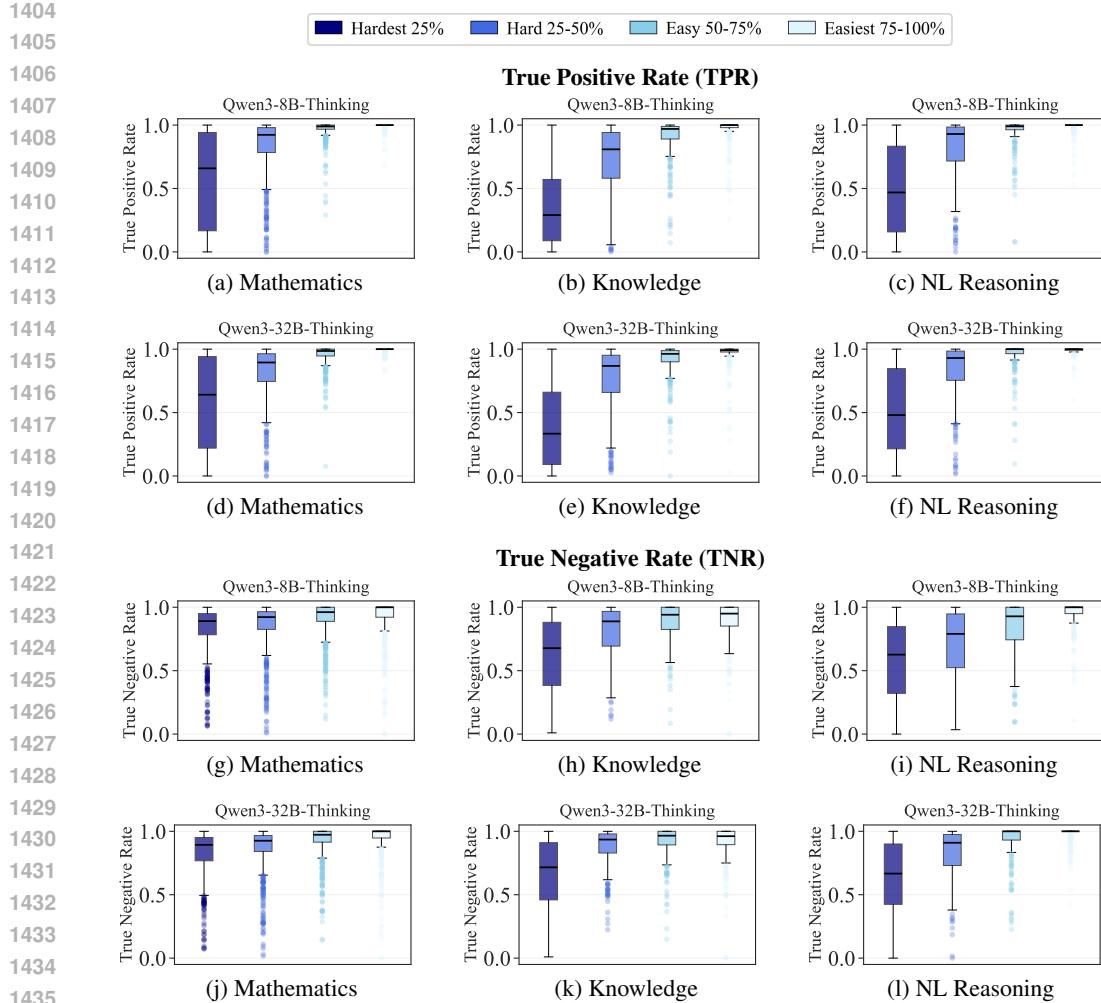


Figure 12: **Verification metrics for reasoning models across difficulty quartiles at the per-problem level.** Each boxplot shows the distribution of per-problem TPR and TNR for Qwen3-8B-Thinking and Qwen3-32B-Thinking across difficulty quartiles in three domains. Both TPR and TNR distributions shift higher and become less variable as problems become easier.

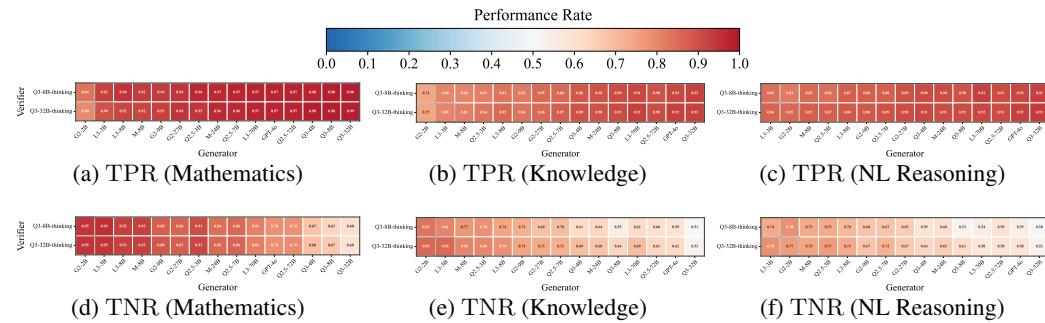


Figure 13: **Reasoning models as verifiers paired with generators of varying capability.** TPR (a-c) and TNR (d-f) for Qwen3-8B-Thinking and Qwen3-32B-Thinking verifiers when evaluating responses from 15 generator models across three domains. Generators are ordered left-to-right by increasing generation capability, measured separately for each domain. Red indicates higher performance, blue indicates lower performance.

1458  
1459

## C.3 EXPLANATION OF BELOW-RANDOM VERIFICATION PERFORMANCE ON HARD PROBLEMS

1460  
1461  
1462  
1463  
1464

In Figure 6f, we observe that verifiers achieve balanced accuracy below the random baseline of 0.5 on hard problems from the NL Reasoning domain, a result that needs explanation. This phenomenon can occur in reference-free evaluation when verifiers employ a “solve-and-match” verification strategy, where they attempt to solve the problem independently and then compare their answer with the generator’s response.

1465  
1466  
1467  
1468  
1469  
1470  
1471

For NL Reasoning tasks with 3-way or 4-way multiple choice formats, this mechanism can produce below-random performance when verifiers consistently fail to solve hard problems correctly. In such cases, the verifier never correctly identifies true positive responses ( $TPR = 0$ ) because it always produces wrong answers that don’t match correct generator responses. However, it can still identify some true negatives when both the generator and verifier happen to select the same wrong answer. For three-way choices, the  $TNR = 0.5$ . With  $TPR$  near zero and  $TNR$  remaining positive, the balanced accuracy falls below 0.5.

1472  
1473  
1474  
1475  
1476  
1477

This phenomenon is specific to tasks with limited answer spaces. The affected problems are those in the hard set with  $d(x) < 0.3$ , where even strong models achieve very low pass rates. It occurs in NL Reasoning because this domain includes three-way multiple-choice questions from datasets like FOLIO. It does not occur in Mathematics, where responses are open-ended strings, or in Knowledge domains with 10-way multiple choice, where the large answer space dilutes the effect.

1478  
1479

## C.4 ADDITIONAL RESULTS OF VERIFIER GENERATION CAPABILITY

1480  
1481  
1482  
1483  
1484  
1485

Here we present additional results for **RQ3** from Section 4.3, providing correlation analysis between verifier generation capability and verification accuracy across the entire problem difficulty range in Figure 14. The results confirm our finding from the main paper that the correlation form varies with problem difficulty: medium problems show strong positive linear relationships, while hard and easy problems exhibit non-linear trends.

1486  
1487  
1488  
1489  
1490  
1491

**Threshold-limited pattern on difficult problems.** The threshold-limited regime appears in difficult problems, for example, in Figure 14 (a) in the difficulty range  $[0.2, 0.3]$ . In this regime, increasing the verifier’s generation capability does not translate into noticeably better verification, which is observed as the saturation pattern. As an example, Qwen3-32B (largest rectangle) has a generation capability of about 0.6, yet its verification performance is nearly identical to Qwen2.5-72B (circle) with a lower generation capability of around 0.4. Both models plateau around a verification score of 0.7.

1492  
1493  
1494  
1495  
1496  
1497

The underlying reason is that on extremely difficult problems, even the verifiers with the best generation capability are not that good ( $\leq 0.6$  pass rate). As a result, pushing generation ability doesn’t yield many gains, as the models still lack the intrinsic ability to “handle” these problems. This implies that reliable verification requires the verifier to reach a sufficiently high level of generation capability. When models fall below this requirement on difficult problems, verification performance plateaus.

1498  
1499  
1500  
1501  
1502

**Transition to a linear pattern on medium problems.** When moving to medium-difficulty problems, such as those in Figure 14 (a) range  $[0.4, 0.5]$ , the relationship becomes more linear. In this regime, Qwen3-32B reaches a higher generation capability (around 0.8), and its verification performance increases accordingly to 0.75. This improvement allows Qwen3-32B to outperform another model with a generation capability of around 0.5 and verification around 0.7, indicating a linear pattern.

1503  
1504  
1505  
1506  
1507  
1508

In this intermediate regime, we see the most linear correlation. These problems are of average difficulty, and models typically demonstrate a baseline level of ability in solving such problems, as measured by generation capability. As such, we find that this regime matches well with regimes analyzed in prior work (Chen et al., 2025c; Tan et al., 2025). Therefore, this region exhibits the most intuitive behavior: Model verification ability tracks directly with its ability to understand (i.e., solve) the problem. It’s important to note that these problems are solvable (i.e., not extremely difficult).

1509  
1510  
1511

**Saturated pattern on easy problems.** In Figure 14(a) for the difficulty range  $[0.9, 1.0]$ , Qwen2.5-7B (medium circle) already reaches nearly 1.0 in generation capability, the same as the stronger Qwen3-32B. Their difference on the  $x$ -axis is therefore essentially zero, yet their verification performance differs by about 0.1. This is the saturated pattern.

1512 The underlying reason is that differences in verification performance come from factors other than the  
 1513 ability to solve the problem itself. One such factor is whether the verifier can detect more subtle or  
 1514 advanced mistakes made by strong generators. As illustrated in the case studies in Figure 33, a model  
 1515 may be able to solve a problem correctly but still fail to recognize certain non-obvious or high-level  
 1516 errors in another model’s reasoning. Thus, perfect generation capability is not sufficient for achieving  
 1517 strong verification performance on easy problems.

1518 These findings highlight the need for regime-aware verifier strategies. On hard problems, strong  
 1519 verifiers are unnecessary as performance plateaus regardless of capability. On medium problems,  
 1520 selecting models with better generation capability consistently yields better verification. On easy  
 1521 problems, selecting higher-capability models works well among weak-to-medium verifiers, but  
 1522 strong models with similar capabilities show vastly different verification performance. Thus, optimal  
 1523 selection of strong verifiers requires supplementary benchmarking or alternative evaluation metrics.

### 1524 C.5 ADDITIONAL RESULTS OF GENERATOR ANALYSIS IN TEST-TIME SCALING

1525 This subsection provides complementary results for the generator analysis presented in Section 5.1,  
 1526 demonstrating the generalizability of our findings across domains and problem difficulties. Figure 15  
 1527 extends the analysis from the main paper to Knowledge and NL Reasoning domains. Our central  
 1528 finding from **RQ4** holds consistently. As Figures 15a to 15c shows, verification gains peak at  
 1529 weak-medium generator strength, enabling these generators to substantially close performance gaps  
 1530 with stronger models. The underlying mechanism driving this phenomenon, identified in **RQ2**,  
 1531 remains consistent across domains. Figures 15d to 15f shows that, as generator strength increases,  
 1532 TNR decreases sharply while TPR rises only modestly. For the strongest generators, the collapsed  
 1533 TNR limits verification gains as errors become increasingly difficult to detect. This brings high  
 1534 verification gains at weak-medium generator levels. In the main paper, we show results on problems  
 1535 with difficulty range  $d(x) \in [0.7, 0.8]$  in Figure 7. Here, Figures 16 to 18 report results across the  
 1536 entire difficulty range for three domains, respectively. Figure 19 shows the percentage of performance  
 1537 gap closed by verification for all weak-to-strong generator pairs, computed on all problems within  
 1538 each domain.

1539  
 1540  
 1541  
 1542  
 1543  
 1544  
 1545  
 1546  
 1547  
 1548  
 1549  
 1550  
 1551  
 1552  
 1553  
 1554  
 1555  
 1556  
 1557  
 1558  
 1559  
 1560  
 1561  
 1562  
 1563  
 1564  
 1565

1566

1567

1568

1569

1570

1571

1572

1573

1574

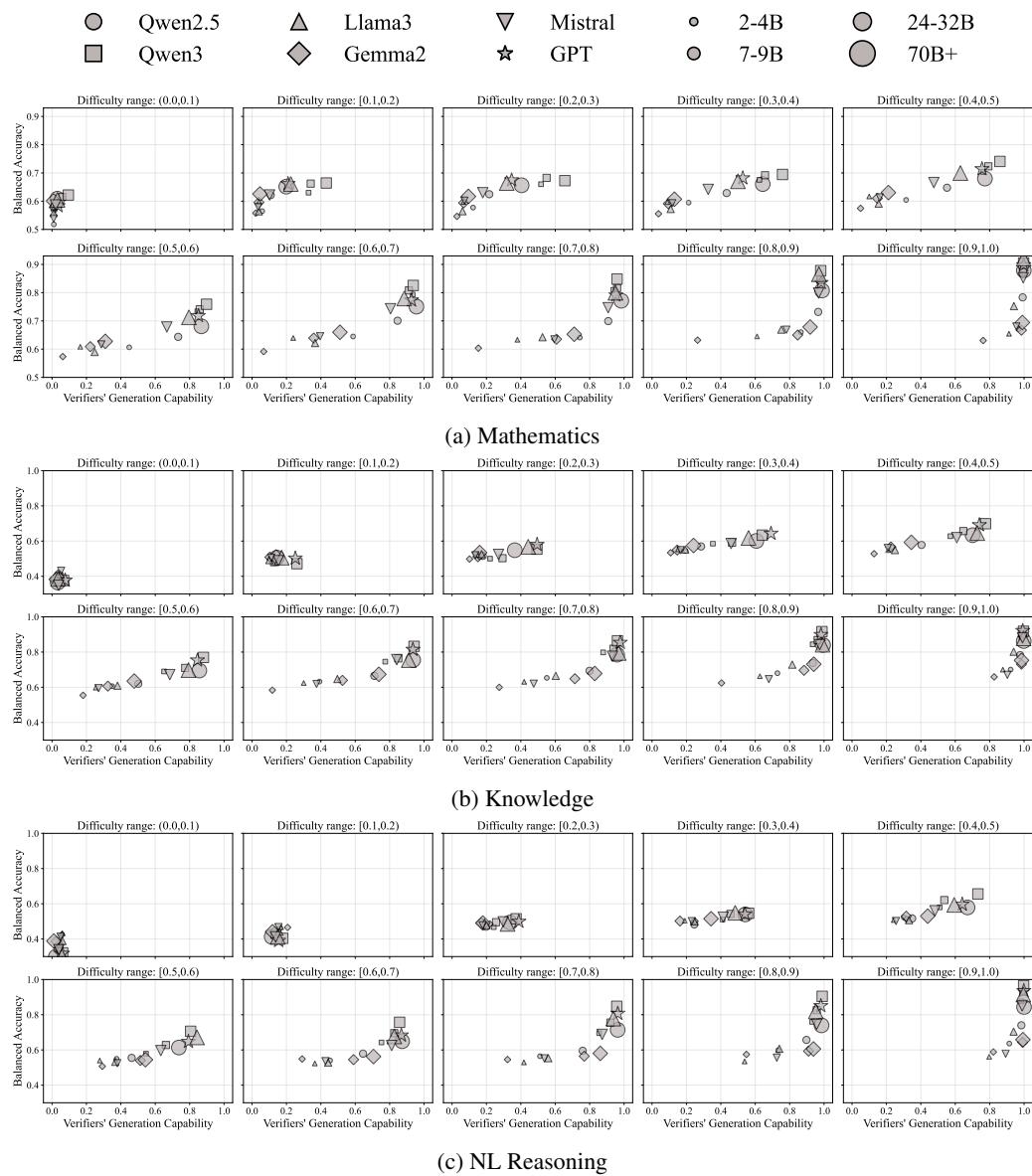


Figure 14: **Correlation between verification performance and generation capability across problem difficulty ranges on three domains.** Balanced accuracy as a function of verifier generation capability for difficulty ranges from (0.0,0.1) to [0.9,1.0). Performance exhibits three distinct regimes: plateaus on hard problems, strong positive correlation on medium problems, and high variance with saturated capability on easy problems. Marker shapes indicate model family; sizes represent model scale.

1610

1611

1612

1613

1614

1615

1616

1617

1618

1619

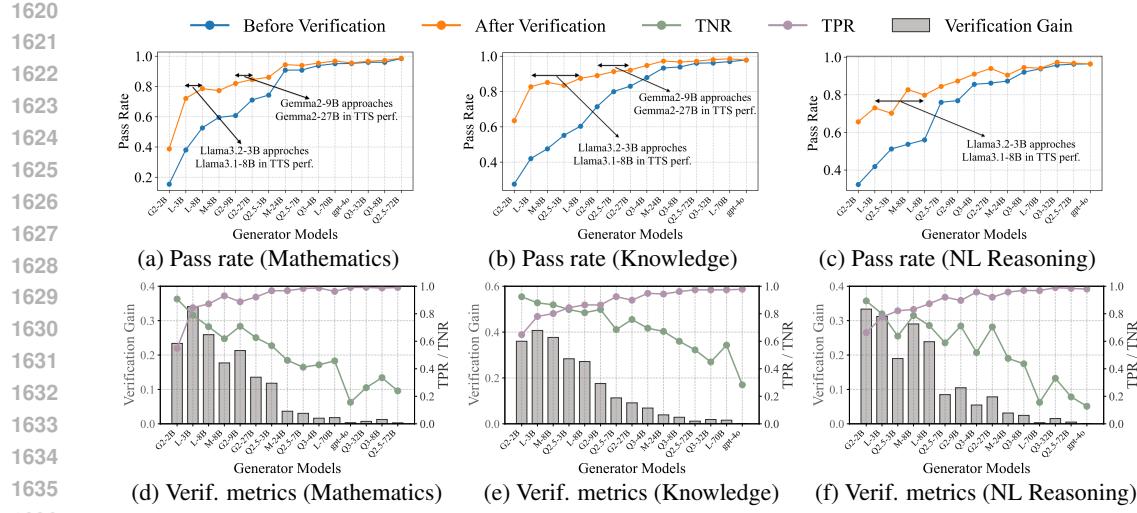


Figure 15: **TTS performance before and after verification when sweeping generator strength.** (a-c) Pass rate before (blue) and after (orange) adding a fixed verifier (GPT-4o), across generators ordered from weaker (left) to stronger (right) by generation capability. (d-f) Bar chart shows the verification gain  $\Delta\hat{p}_V$  (left y-axis) for each generator. Lines show the verifier's TNR and TPR on the same datasets (right y-axis). Results are reported on problems with difficulty in the range [0.7, 0.8) for three domains. Problem counts across domains: 181 (Mathematics), 154 (Knowledge), 97 (NL Reasoning).

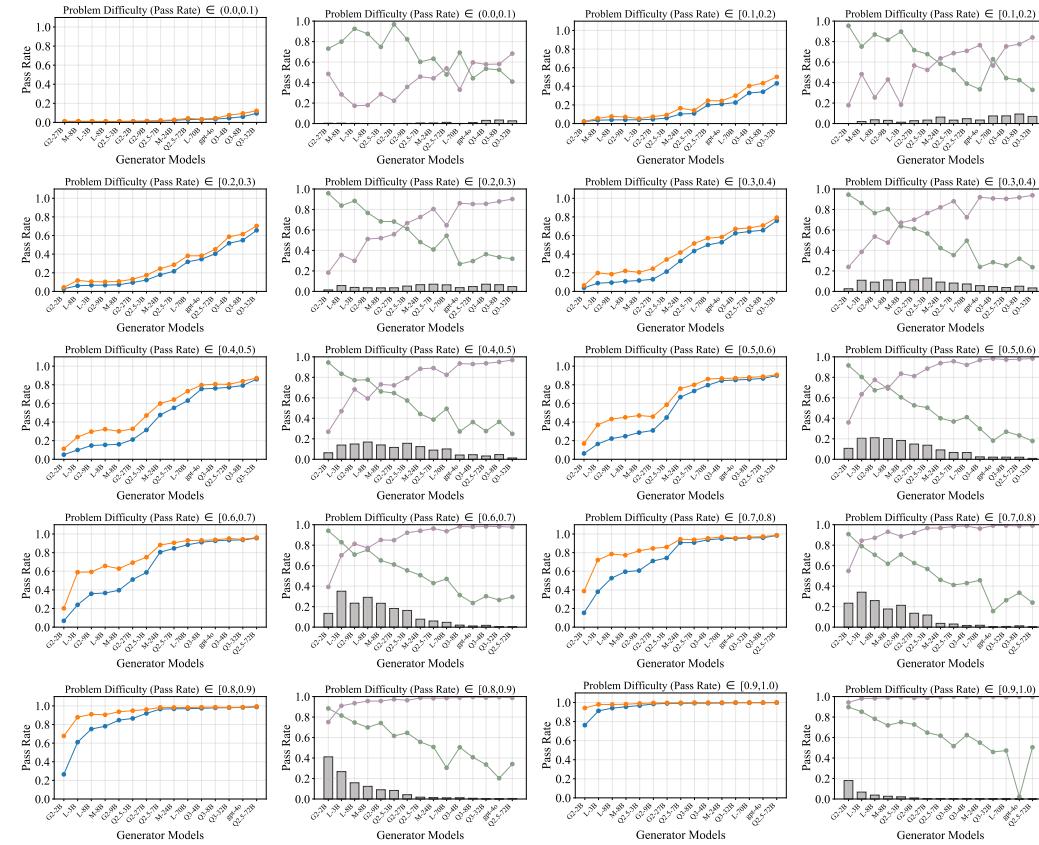


Figure 16: **Verification-augmented TTS performance across the full range of problem difficulties, shown here for the Mathematics domain.** Each pair of figure corresponds to a different difficulty interval (measured by pass rate  $d(x)$ ), with the left panel showing pass rates before (blue) and after (orange) verification, and the right panel showing verification gain  $\Delta\hat{p}_V$  (bars) alongside the verifier's TNR (green) and TPR (purple). Compared to Figure 7, which focused only on problems with  $d(x) \in [0.7, 0.8)$ , this includes the entire difficulty range.

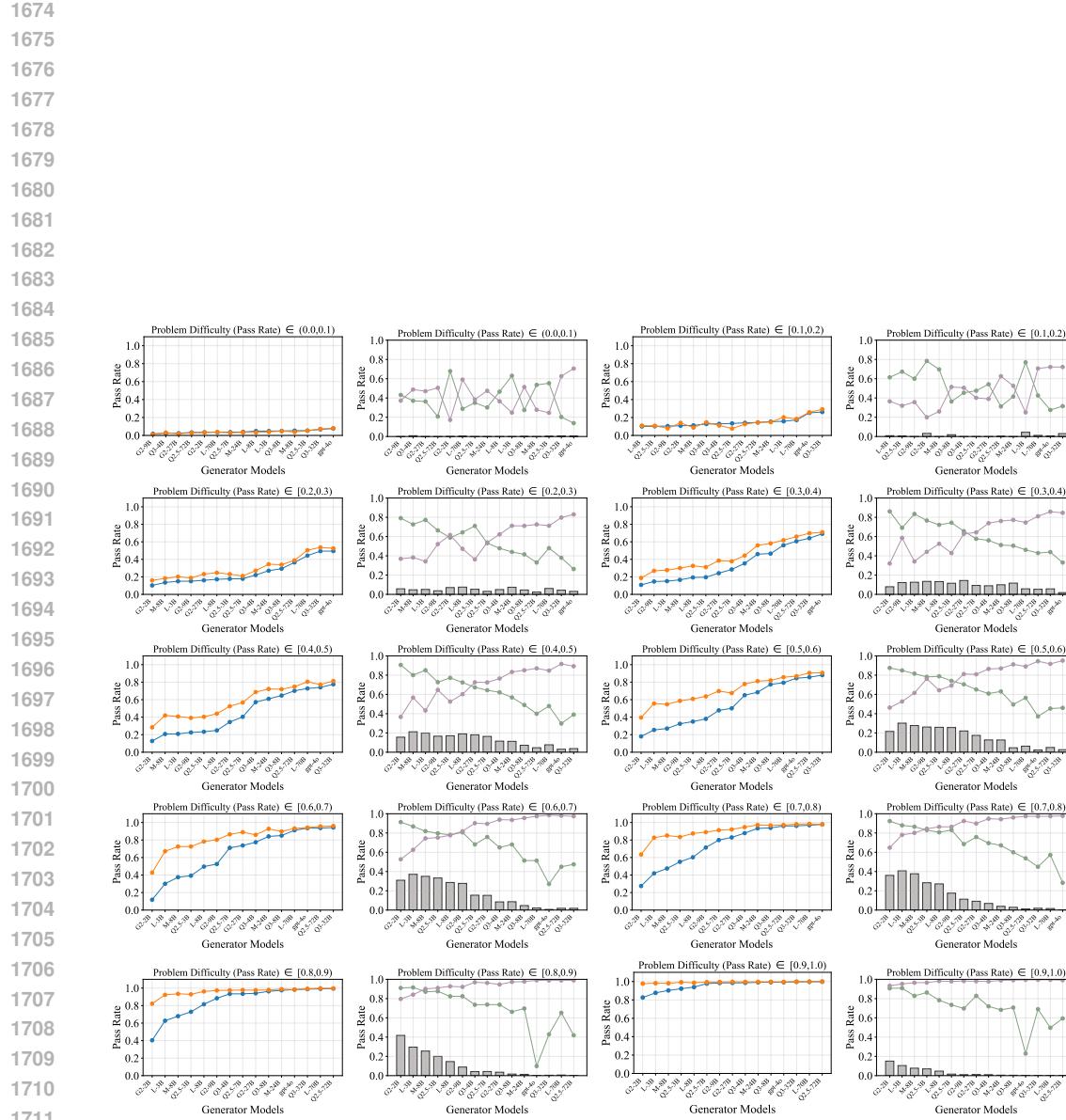


Figure 17: **Verification-augmented TTS performance across the full range of problem difficulties, shown here for the Knowledge domain.** Each pair of figure corresponds to a different difficulty interval (measured by pass rate  $d(x)$ ), with the left panel showing pass rates before (blue) and after (orange) verification, and the right panel showing verification gain  $\Delta\hat{p}_V$  (bars) alongside the verifier’s TNR (green) and TPR (purple). Compared to Figure 7, which focused only on problems with  $d(x) \in [0.7, 0.8)$ , this includes the entire difficulty range.

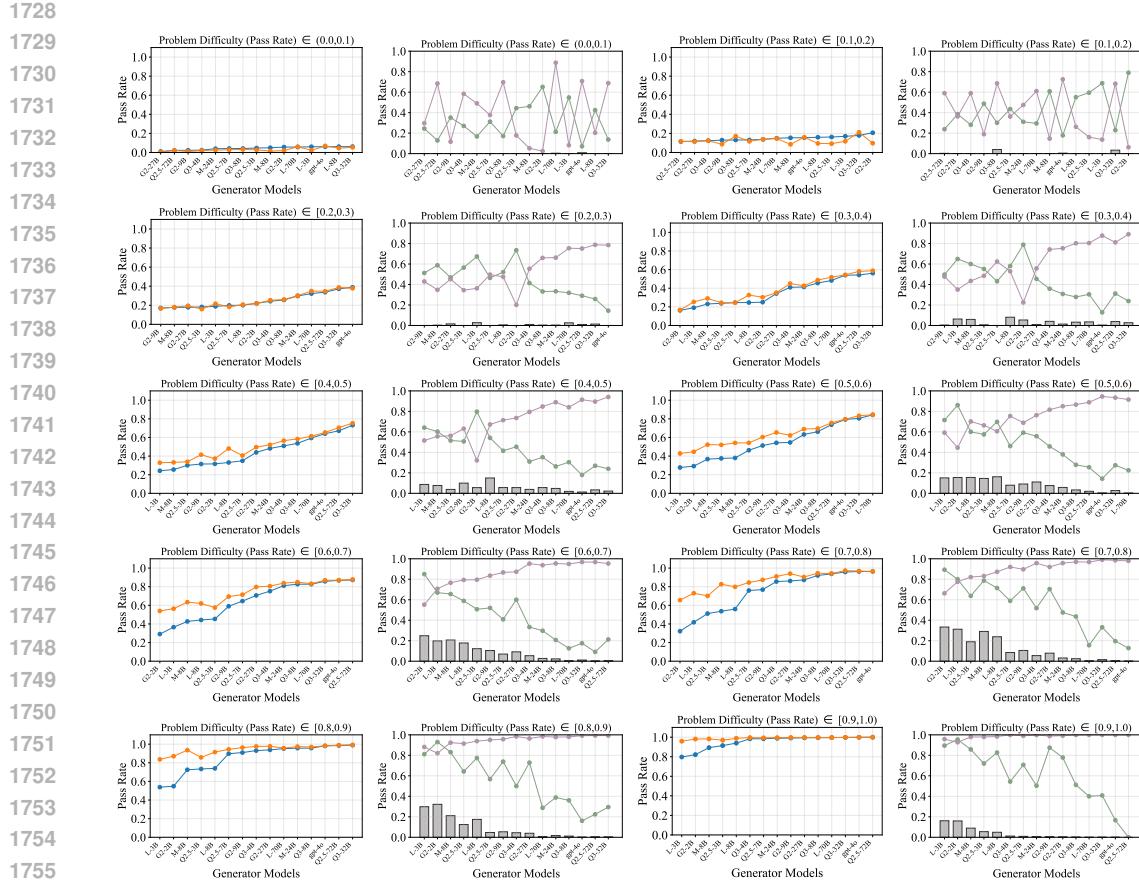


Figure 18: **Verification-augmented TTS performance across the full range of problem difficulties, shown here for the NL reasoning domain.** Each pair of figure corresponds to a different difficulty interval (measured by pass rate  $d(x)$ ), with the left panel showing pass rates before (blue) and after (orange) verification, and the right panel showing verification gain  $\Delta\hat{p}_V$  (bars) alongside the verifier’s TNR (green) and TPR (purple).

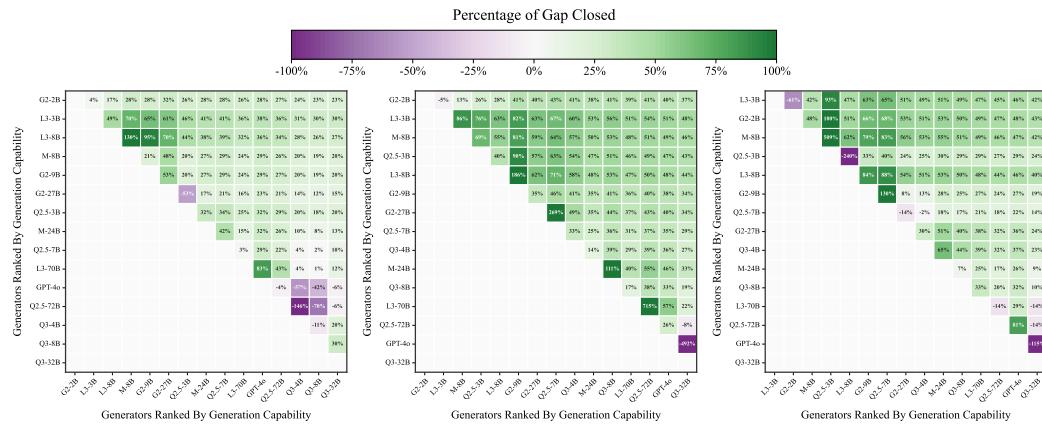
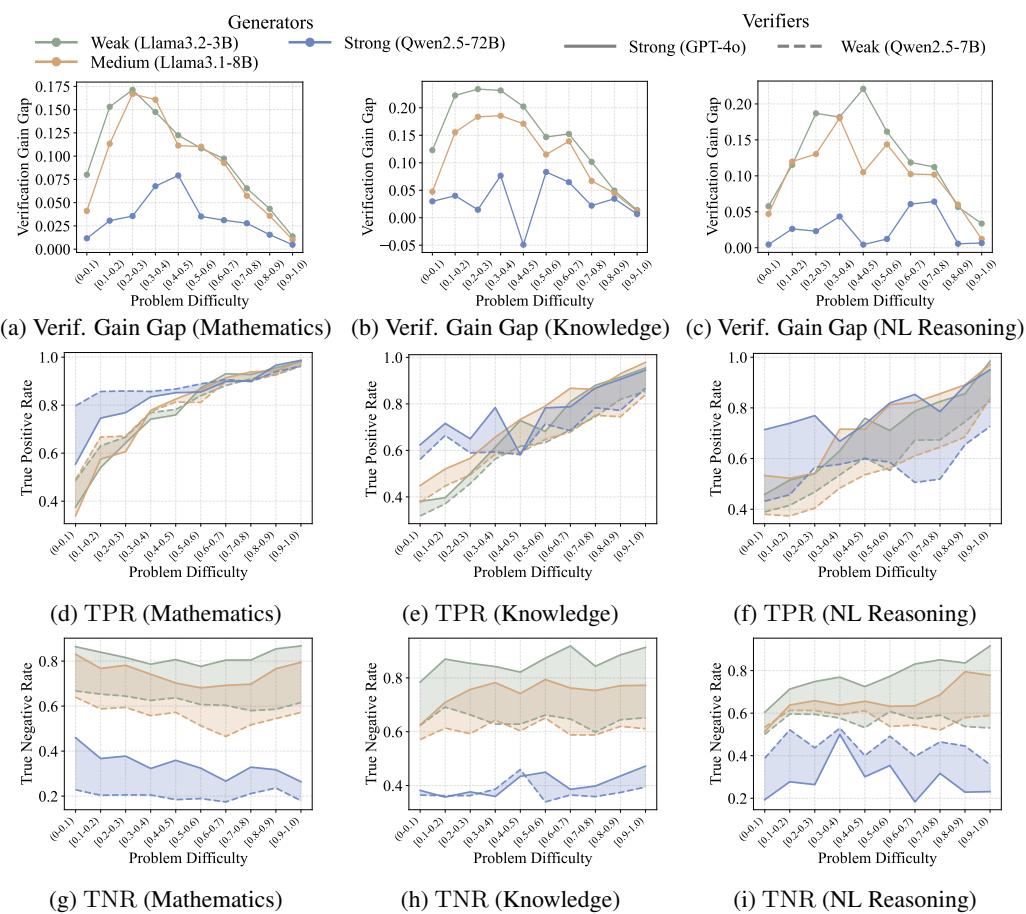


Figure 19: **Percentage of TTS performance gap between weak and strong generators closed by verification.** Each heatmap shows the fraction of the performance gap between a weaker generator ( $x$ -axis) and a stronger generator ( $y$ -axis) that is closed by verification with a fixed verifier GPT-4o. Green cells indicate a larger gap closure, meaning the weaker model approaches the stronger one after verification. A value greater than 100% means that the originally weaker model performs better with verifier augmentation. Purple cells indicate negative values where verification increases the gap.

## 1782 C.6 ADDITIONAL RESULTS OF VERIFIER ANALYSIS IN TEST-TIME SCALING

1784 Here we provide complementary results to the verifier analysis in Section 5.2. Figure 20 presents two  
 1785 other domains' results. Figure 21 presents additional metrics in the same setup for complete analysis,  
 1786 including balanced accuracy and verification gains. We can see that our findings from Mathematics  
 1787 generalize to other domains. Figures 20a to 20c shows that weak verifiers can approximate strong  
 1788 verifier performance in TTS, at the extremes of problem difficulty or responses generated by strong  
 1789 generators. While on these regimes, we show that both verifiers provide limited verification gain in  
 1790 Figures 21a to 21c. Notably, in the NL Reasoning domain (Figure 20i), on the strongest generator, both  
 1791 verifiers' TNR fall below 0.5, and the weak verifier's TNR exceeds that of the strong verifier. Despite  
 1792 this TNR inversion, the overall verification performance gap (shown as balanced accuracy in Figure  
 1793 21f) remains narrow, with the strong verifier maintaining a slight advantage due to its superior TPR.  
 1794



1823 **Figure 20: Analyzing verification gain gaps and TPR/TNR between verifiers under varying**  
 1824 **problem difficulty and generator strength.** The  $x$ -axis shows problem difficulty measured relative  
 1825 to each generator. Shaded regions visualize the difference in metrics between verifiers for each  
 1826 generator. (a-c) Verification gain gap between strong and weak verifiers. (d-f) TPR increases as  
 1827 problems become easier for all generator-verifier combinations. (g-i) TNR decreases as generators  
 1828 become stronger, with TNR gap narrowing.  
 1829  
 1830  
 1831  
 1832  
 1833  
 1834  
 1835

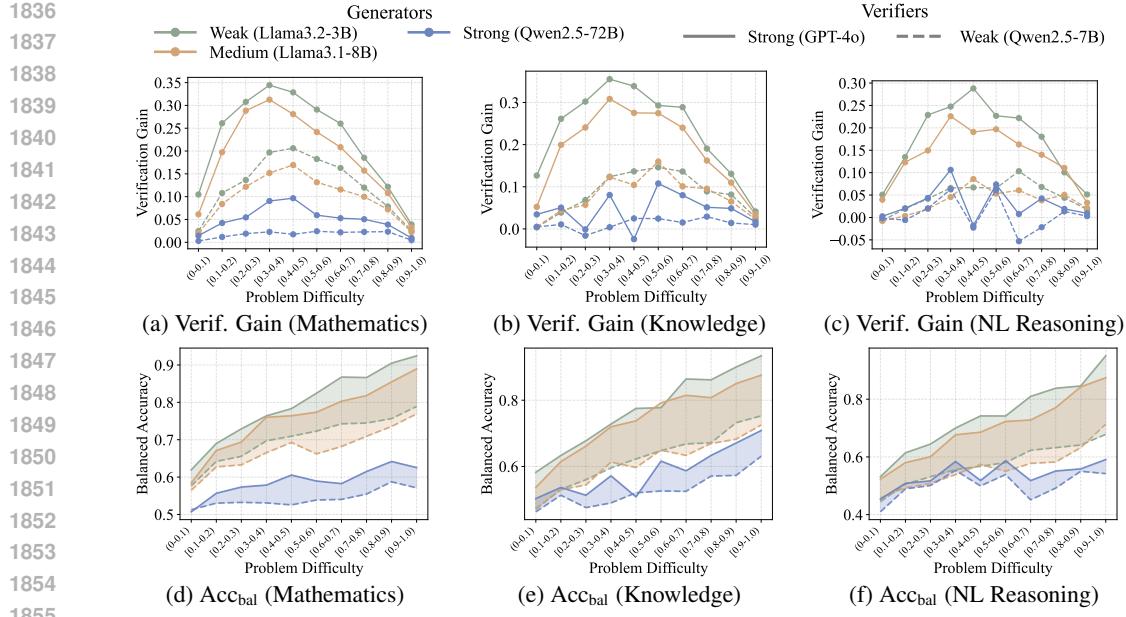


Figure 21: **Analyzing verification gains and  $\text{Acc}_{\text{bal}}$  between verifiers under varying problem difficulty and generator strength.** The  $x$ -axis shows problem difficulty measured relative to each generator. Shaded regions visualize the difference in metrics between verifiers for each generator. (a-c) Verification gain for both strong (solid lines) and weak (dashed lines) verifiers across three generators: weak, medium, and strong. (d-f) Balanced accuracy for the same verifier-generator combinations. The accuracy gap (shaded regions) between verifiers is smallest on the hard problems (left side of the  $x$ -axis).

## D ADDITIONAL RESULTS

### D.1 ESTIMATION OF PROBLEM DIFFICULTY

We adopt the uncertainty-based estimator of problem difficulty from Lee et al. (2025). The idea is that a problem is more difficult when a model produces highly diverse or inconsistent answers across repeated stochastic samples. For a given problem, we sample a generator model  $K$  times and collect the final answers:  $\{a(r_1), a(r_2), \dots, a(r_k)\}$ . Let the set of unique answers be  $\{u_1, \dots, u_M\}$ , where  $M \leq K$ . Let  $n_j$  denote the number of times answer  $u_j$  appears among the  $K$  samples. The corresponding empirical probabilities are:

$$p_j = \frac{n_j}{K}, \quad j = 1, \dots, M. \quad (2)$$

We compute the Shannon entropy of the empirical answer distribution:

$$H = - \sum_{j=1}^M p_j \log p_j \quad (3)$$

The maximum possible entropy occurs when all  $K$  samples produce distinct answers, giving  $H_{\text{max}} = \log K$ . We define the normalized difficulty score as

$$\text{difficulty}(x) = \frac{H}{\log K} \quad (4)$$

This value lies in  $[0, 1]$ , taking 0 when all samples agree (easy problems) and 1 when all samples differ (hard problems), and serves as an uncertainty-based estimate of the intrinsic difficulty of each problem.

For implementation, we draw 8 solutions from Qwen2.5-3B for each question and compute the uncertainty-based difficulty estimator defined above. We use the resulting estimated difficulty values to reproduce the main results of RQ1 (Section 4.1) and RQ3 (Section 4.3). As shown in Figures 22 and 23, the findings remain consistent when using this estimated difficulty measure, demonstrating that our conclusions hold under a practical difficulty-estimation setting.

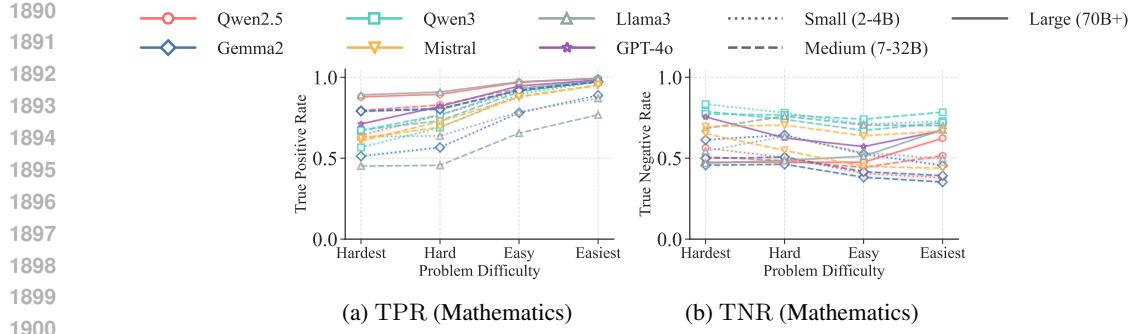


Figure 22: **Ablation study of RQ1 with estimated problem difficulty.** The experimental setup follows that of Section 4.1 and Figure 2, except that we replace the oracle difficulty with our uncertainty-based difficulty estimation.

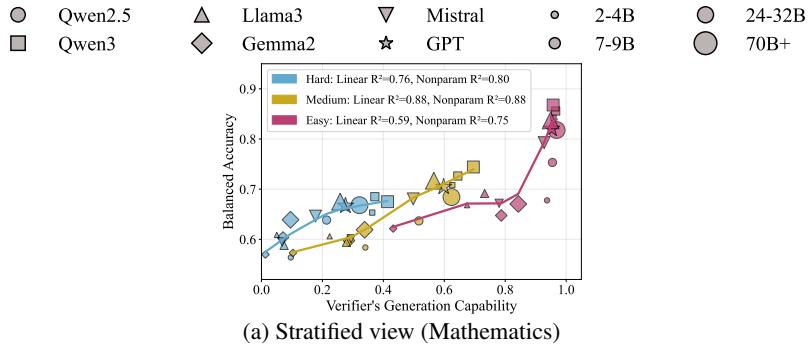


Figure 23: **Ablation study of RQ3 with estimated problem difficulty.** The experimental setup follows that of Section 4.3 and Figure 6, except that we replace the oracle difficulty with uncertainty-based difficulty estimation.

## D.2 EFFICIENCY AND PERFORMANCE ANALYSIS OF MODEL CHOICES IN TTS

This subsection examines how to make cost-effective model choices using our findings from RQ4 and RQ5 (Section 5). From RQ4, we observe that when weaker generators achieve high verification performance, they also get larger TTS verification gains, and can approach the performance of stronger generators. From RQ5, we find that in certain regimes, smaller verifiers can obtain verification performance comparable to larger ones, with similar verification gains. These results motivate us to use a verification metric to guide generator and verifier selection in TTS.

In these experiments, we follow the efficiency metric and difficulty-stratified analysis used in Liu et al. (2025a). For each subset stratified by problem difficulty, we split the problem set into a 50% validation set and a 50% test set. Verification metrics are computed on the validation set. Specifically, for each problem, we sample 8 balanced correct and incorrect responses from the generator, apply the verifier, and compute balanced accuracy (the average of TPR and TNR). We use this metric to select the generator or verifier among the candidates, and then evaluate their efficiency and TTS accuracy on the test set.

For the efficiency metric, we use inference FLOPs, computed as  $2ND_{\text{inference}}$ , where  $N$  represents the model parameters and  $D_{\text{inference}}$  is the total number of tokens generated during inference. For open-source models, we approximate  $N$  using the parameter counts indicated in their model names (e.g., we use  $2 \times 10^9$  parameters for Gemma2-2B). For the proprietary model GPT-4o, we estimate the number of active parameters during inference as 100B, since the largest open-source model in our study is 72B and GPT-4o is widely understood to operate at a larger effective scale. This estimate is used exclusively for relative efficiency comparison and does not affect our experimental findings. We report FLOPs averaged over all problems. When results are reported without verification, only the generator FLOPs are included; when results are reported with verification, the FLOPs of both the generator and the verifier are summed. We use a TTS sample size of 64 for every model combination

1944 and problem. We compute the total FLOPs used per problem and report the average FLOPs over the  
 1945 problem set.

1946  
 1947 For the accuracy metric, we use the pass rate defined in Section 3.1, which measures the expected  
 1948 accuracy obtained by uniformly sampling one response from either the set of model-generated samples  
 1949 or from the verifier-retained samples. To complement the pass rate, we also report majority-vote  
 1950 accuracy.

1951 We consider the following two subproblems, named RQ4.1 and RQ5.1.

1952  
 1953 **D.2.1 RQ4.1: HOW TO CHOOSE GENERATORS GIVEN A FIXED VERIFIER?**

1954 We compare three generator models (Gemma2-2B, 9B, and 27B) under a fixed verifier, and we restrict  
 1955 the comparison to models within the same family and version to avoid confounding effects from  
 1956 differences in pre-training quality. Across three difficulty ranges: easy [0.9, 1.0), medium [0.7, 0.8),  
 1957 and hard [0.3, 0.4), we select the generator with the highest verification performance measured on  
 1958 the validation set. We vary the verifier between Qwen3-32B and GPT-4o.

1959 **Results of RQ4.1** The results are presented in Figure 24 (pass rate, Qwen3-32B as verifier), Figure 25  
 1960 (pass rate, GPT-4o as verifier), Figure 26 (majority vote, Qwen3-32B as verifier), and Figure 27  
 1961 (majority vote, GPT-4o as verifier). Each figure contains three rows: (1) TTS performance vs.  
 1962 efficiency without verification on the test set, (2) the same analysis with verification, and (3) balanced  
 1963 accuracy on the validation set. The model with the highest balanced accuracy is selected as the  
 1964 cost-efficient choice and highlighted in rows 2 and 3.

1965 Without verification, the largest model is usually the preferred choice due to its large accuracy  
 1966 advantage. With verification, there are cases where smaller models become compute-efficient choices  
 1967 because they achieve comparable TTS performance while requiring fewer FLOPs. This is consistent  
 1968 with our earlier observation in Section 5.1 RQ4. Notably, the generator selected by the highest  
 1969 balanced accuracy on the validation set (red circle) consistently matches the compute-efficient choice:  
 1970 it either identifies a smaller model that achieves accuracy close to the larger one or preserves the  
 1971 largest model when it retains a clear performance advantage. This shows that verification-based  
 1972 metrics provide effective guidance for generator selection.

1973

1974

1975

1976

1977

1978

1979

1980

1981

1982

1983

1984

1985

1986

1987

1988

1989

1990

1991

1992

1993

1994

1995

1996

1997

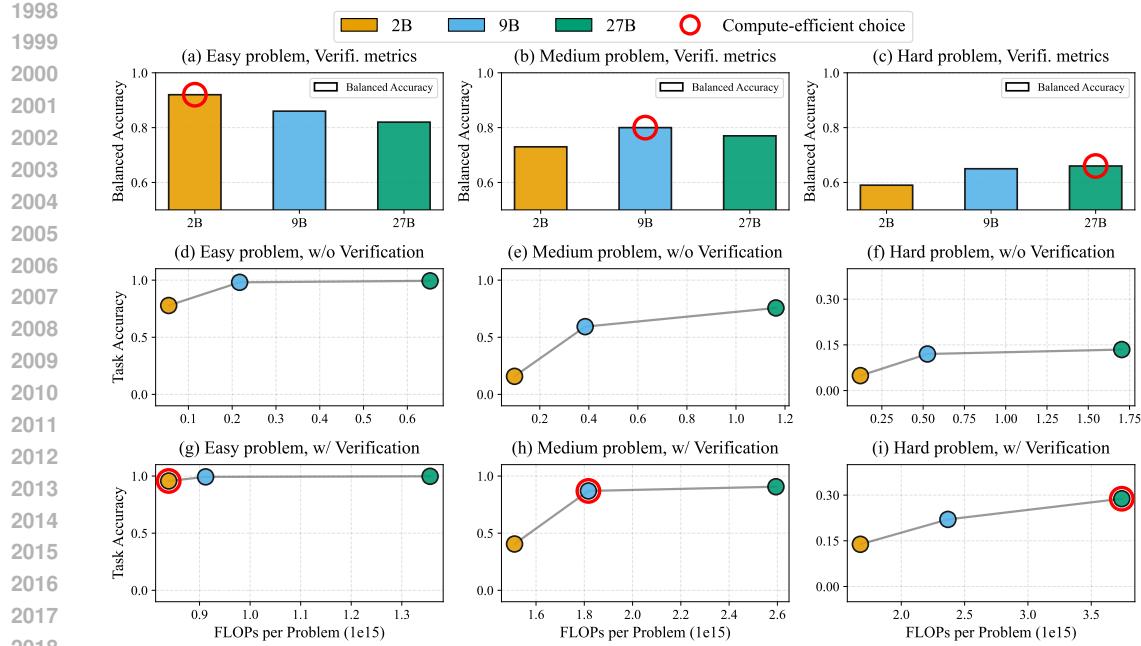


Figure 24: **Using a verification metric to select compute-efficient generators across different problem difficulties.** Qwen3-32B is used as the verifier, and generator candidates are Gemma2-2B, 9B, and 27B. (a–c) Balanced accuracy on the validation set varies across difficulty ranges and identifies different preferred generators: the 2B model on easy problems, 9B on medium problems, and 27B on hard problems. (d–f) TTS accuracy versus compute on the test set without verification. (g–i) TTS accuracy versus compute with verification. The generators selected by balanced accuracy correspond to the compute-efficient choices: on easier problems, smaller generators achieve accuracy close to larger models while requiring fewer FLOPs, whereas on hard problems, the largest generator retains a performance advantage.

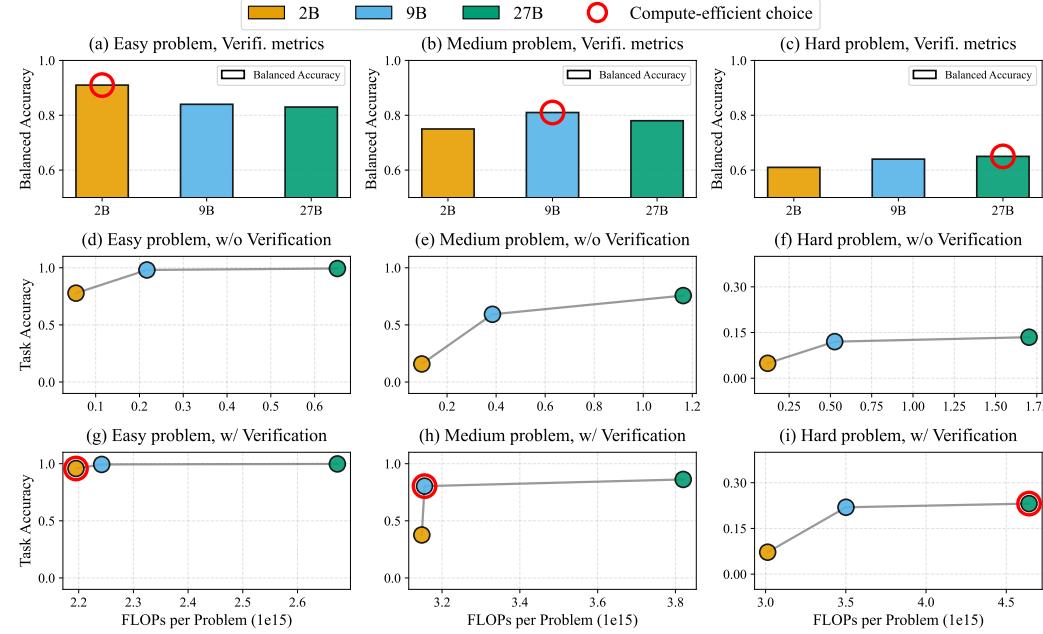


Figure 25: **Using a verification metric to select compute-efficient generators across different problem difficulties.** GPT-4o is used as the verifier; task accuracy is measured by pass rate. The experimental setups and observations are consistent with Figure 24.

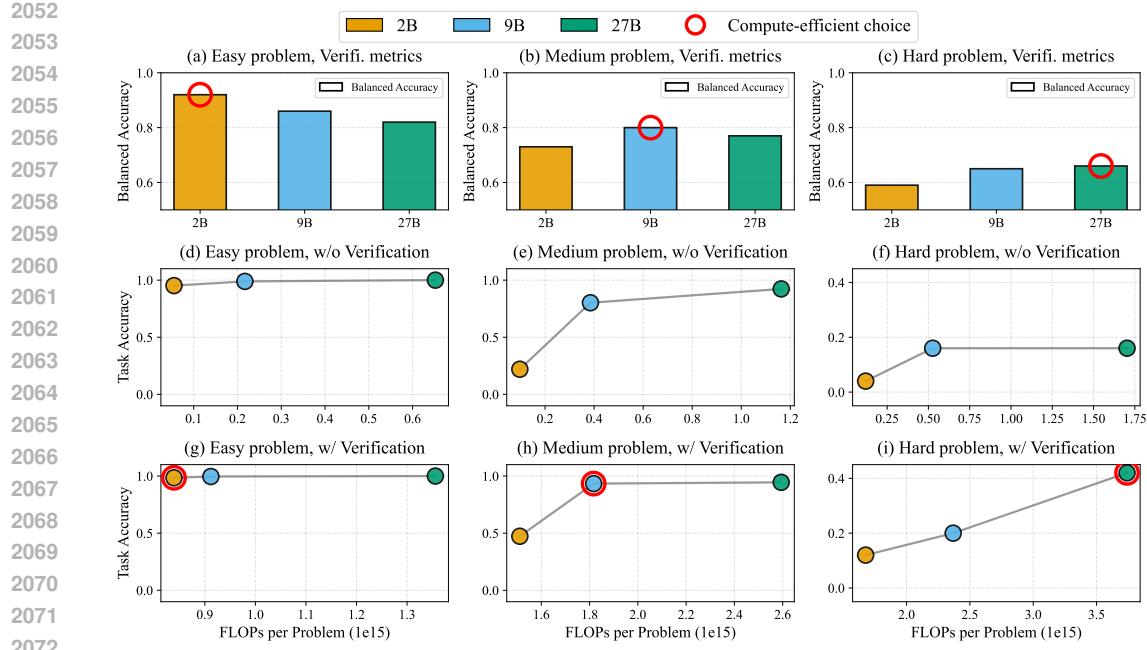


Figure 26: **Using a verification metric to select compute-efficient generators across different problem difficulties. Qwen3-32B is used as the verifier; task accuracy is measured by majority vote.** The experimental setups and observations are consistent with Figure 24.

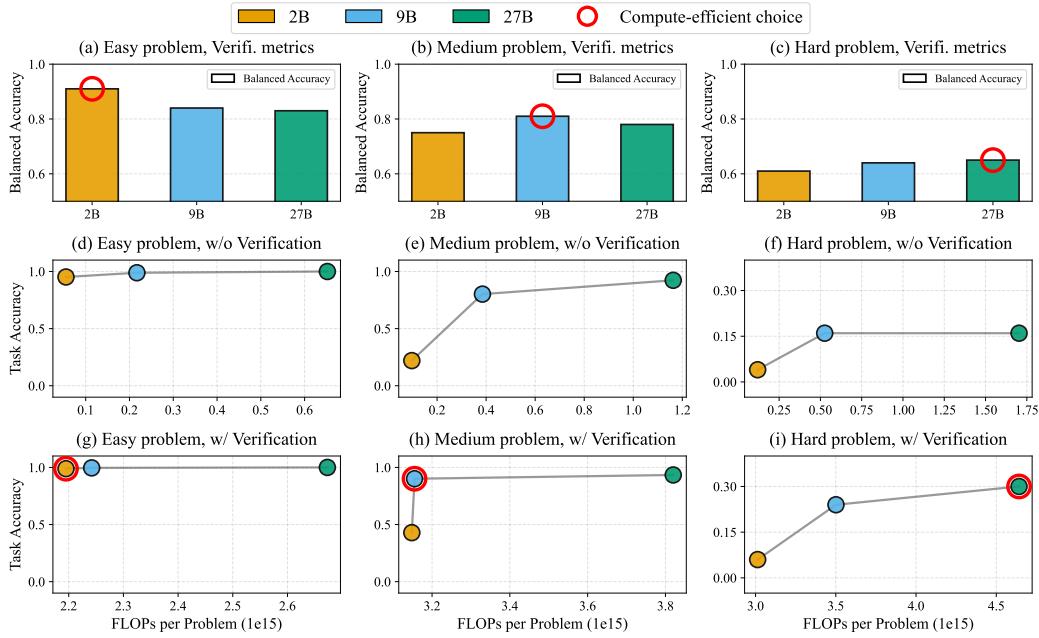


Figure 27: **Using a verification metric to select compute-efficient generators across different problem difficulties. GPT-4o is used as the verifier; task accuracy is measured by majority vote.** The experimental setups and observations are consistent with Figure 24.

### D.2.2 RQ5.1: HOW TO CHOOSE A VERIFIER GIVEN A FIXED GENERATOR?

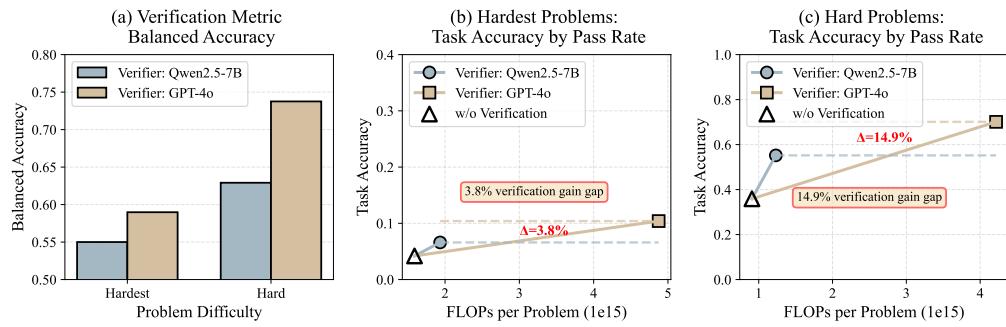
We evaluate whether verification metrics can guide the choice between two verifiers (GPT-4o and Qwen2.5-7B) under different levels of problem difficulty. Following the procedure in Appendix D.2.1,

2106 we compute the verification metric on the validation set and use it to select the verifier. TTS accuracy  
 2107 and computation cost are then reported on the test set.

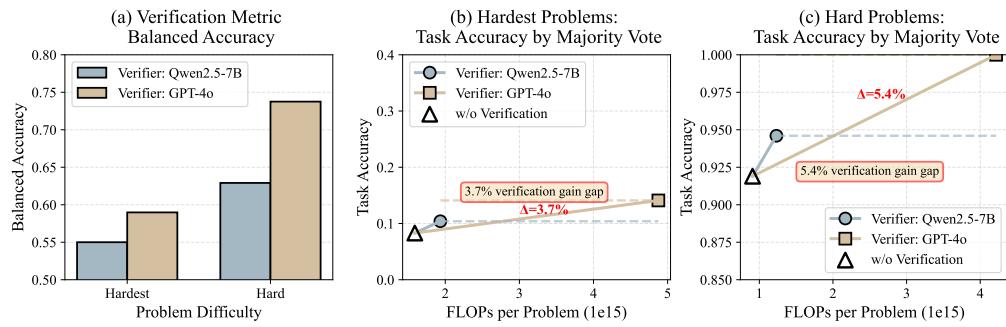
2108 **Results of RQ5.1** The results are presented in Figure 28 (pass rate) and Figure 29 (majority vote).  
 2109 For each figure, the first two subplots report TTS accuracy versus compute when using a small or  
 2110 large verifier. The final subplot reports the verification metrics.

2112 On the hardest problem, GPT-4o yields only a small balanced accuracy improvement over Qwen2.5-  
 2113 7B. The minimal metric difference suggests the smaller verifier. Evaluated on the test set, the accuracy  
 2114 difference between the two verifiers is small. It confirms that the stronger verifier offers limited  
 2115 additional benefit in this regime. On hard problems, GPT-4o shows a clear gap of balanced accuracy  
 2116 over the small model. This difference points to choosing GPT-4o. On the test set, this leads to  
 2117 substantially improved TTS accuracy despite higher compute cost.

2118 These results show that verification accuracy provides actionable guidance. When compute is limited,  
 2119 one can compare the balanced accuracy of the large and small verifiers: if the gap is small, the  
 2120 smaller verifier is the more cost-efficient choice; if the gap is large, the stronger verifier yields more  
 2121 substantial verification gains in TTS.



2134 **Figure 28: Verification-metric-guided verifier selection across problem difficulty; task accuracy**  
 2135 **is measured by pass rate.** (a) Balanced accuracy on the validation set: when the metric gap between  
 2136 GPT-4o and Qwen2.5-7B is small (hardest problems), the smaller verifier is selected; when the gap is  
 2137 large (hard problems), the stronger verifier is selected. (b–c) Corresponding TTS accuracy on the test  
 2138 set: consistent with the metric-based choice, GPT-4o yields only a small improvement on the hardest  
 2139 problems but provides substantially larger gains on the hard problems.

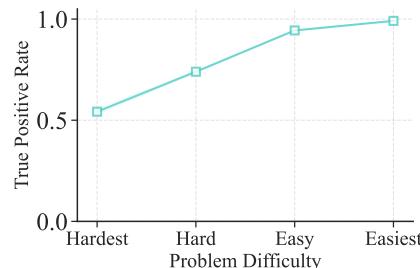


2152 **Figure 29: Verification-metric-guided verifier selection across problem difficulty; task accuracy**  
 2153 **is measured by majority vote.** The experimental setups and observations are the same as Figure 28.

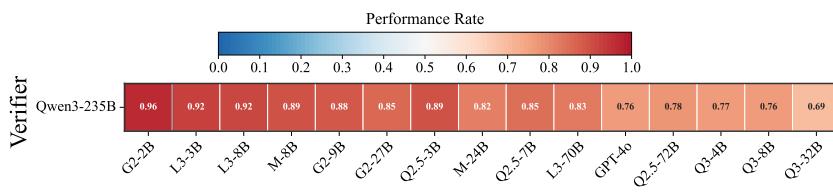
### 2156 D.3 ABLATION STUDY ON LARGE MODELS

2158 We conduct an ablation study using a large LLM with more than 100B parameters, Qwen3-235B  
 2159 (Qwen/Qwen3-235B-A22B-Instruct-2507). We use the recommended sampling hyperparameters:  
 temperature 0.7, top-p 0.8, and top-k 20. We repeat the experiments for RQ1 (Section 4.1) and RQ2

(Section 4.2) on the Mathematics domain and report the results in Figure 30. The results show that our main findings remain consistent when using a larger model.



(a) RQ1 Trend of TPR



(b) RQ2 Trend of TNR

Figure 30: **Ablation study of our main findings with the large model Qwen3-235B.** The experiments follow the same setup as Sections 4.1 and 4.2 and are conducted on the Mathematics domain. (a) The trend that TPR increases as problem difficulty decreases is consistent with our RQ1 finding. (b) The trend that TNR decreases as the generator strength increases is consistent with our RQ2 finding.

#### D.4 ABLATION STUDY ON VERIFICATION PROMPTS

We conduct an ablation study to evaluate the robustness of our findings under different verification prompts. All prompt templates used in this study are listed in Appendix A. In addition to the original “Verification Evaluation Prompt,” we consider two alternatives: a concise “step-by-step” verification prompt from Zhang et al. (2024), and a “solve-then-verify” prompt that encourages the verifier to first solve the problem before evaluating the candidate response, a strategy shown to be effective in prior work Chen et al. (2025d). These two prompts differ substantially from our original design.

The experiments follow the same setup as Sections 4.1 and 4.2 and are conducted on the Mathematics domain. For each prompt, we apply Qwen2.5-72B as the verifier to evaluate responses produced by all 15 generator models. The results in Figure 31 show that our main findings regarding TPR and TNR dynamics remain consistent across all prompts, indicating that the observed verification dynamics are robust to prompt variations.

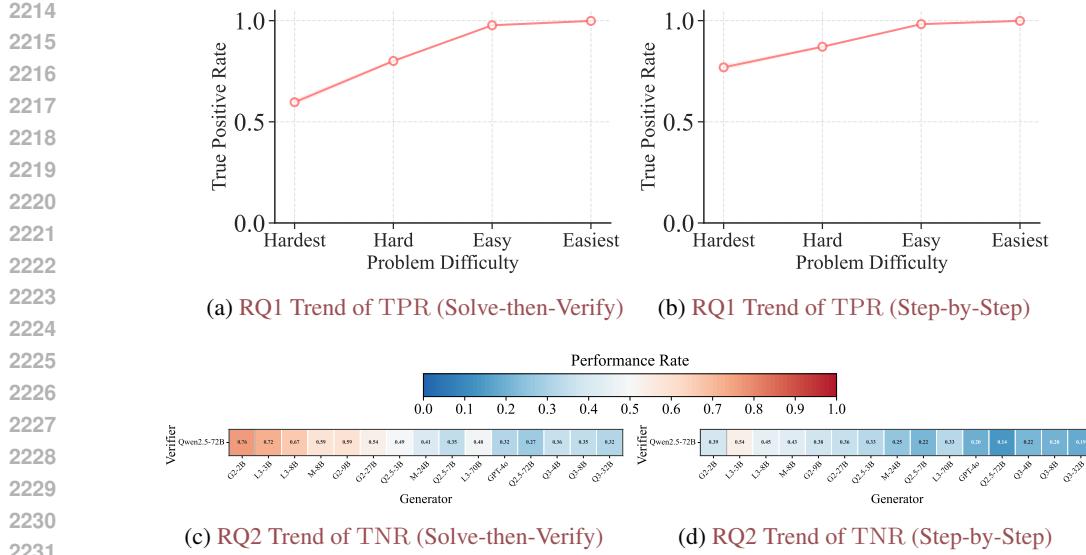


Figure 31: **Ablation study of our findings under different verification prompts.** (a, c) Using “solve-then-verify” prompt for verification. (b, d) Using “step-by-step” prompt for verification. The trend that TPR increases as difficulty decreases (RQ1) and TNR decreases as the generator strength increases (RQ2) still holds under both prompts.

Table 2: Comparison of TTS performance and token cost per problem when using a strong model as the verifier versus as the generator. Token count is reported on average per prompt.

Generator	Verifier	Pass Rate	Avg. GPT-4o Tokens	Avg. Qwen3-4B Tokens
Qwen3-4B	—	0.938	—	551
GPT-4o	—	0.952	482	—
Qwen3-4B	GPT-4o	0.954	193	551

## D.5 COMPARISON ANALYSIS OF STRONG MODEL AS GENERATOR OR VERIFIER

We examine whether it is practical to use a strong model as a verifier instead of using it solely as a generator. We compare two setups: (i) using GPT-4o as the generator, and (ii) using GPT-4o as the verifier together with a smaller generator, Qwen3-4B.

We use the same problem set as in Section 5.1: Mathematics problems with difficulty in the range [0.7, 0.8), consisting of 181 problems. The TTS sampling size is 64. Task accuracy is measured using the pass rate, and efficiency is measured using the average tokens per prompt.

Here are the findings. Both setups achieve the same performance, but using GPT-4o as a verifier consumes 193 tokens on average per prompt, whereas GPT-4o as a generator uses 483 tokens per prompt on average. This represents a  $2.5 \times$  reduction in token usage (and thus cost) per prompt. Thus, for practitioners who host small models but care about maximizing performance per dollar, leveraging the relatively short-form nature of verification with API models represents massive savings.

## E CASE STUDIES

**Problem domain, dataset and difficulty:** Mathematics, MATH-500, Hardest ( $d(x) = 0.075$ )

**Generator:** Qwen2.5-3B

**Verifier:** GPT-4o

**Question:** Bob and Alice each have a bag that contains one ball of each of the colors, blue, green, orange, red, and violet. Alice randomly selects one ball from her bag and puts it into Bob’s bag. Bob then randomly selects one ball from his bag and puts it into Alice’s bag. What is the probability that after this process the contents of the two bags are the same?

**Ground-Truth Solution:** After Alice puts the ball into Bob’s bag, his bag will contain six balls: two of one color and one of each of the other colors. After Bob selects a ball and places it into Alice’s bag, the two bags will have the same contents if and only if Bob

2268 picked one of the two balls in his bag that are the same color. Because there are six balls in the bag when Bob makes his selection, the  
 2269 probability of selecting one of the same colored pair is  $2/6 = \boxed{\frac{1}{3}}$ .  
 2270  
 2271 **Generator Output (Correct):** To determine the probability that the contents of both Bob's and Alice's bags are the same after the  
 2272 process described, we need to analyze each possible step and its impact on the contents of the bags.  
 2273 Initially, each bag contains one ball of each of the colors: blue, green, orange, red, and violet.  
 2274 **Step-by-Step Analysis** [ *Details omitted* ]  
 2275 **Identifying the Conditions for Equality** [ *Details omitted* ]  
 2276 **Calculating the Probability** [ *Details omitted* ]  
 2277 The probability is then calculated as follows:  
 2278 
$$\text{Probability} = \frac{\text{Number of favorable outcomes}}{\text{Total number of outcomes}} = \frac{2}{6} = \frac{1}{3}$$
  
 2279  
 2280 Thus, the probability that after this process the contents of the two bags are the same is  $\boxed{\frac{1}{3}}$ .  
 2281  
 2282 **Verifier Output (False negative):** Explanation: The problem involves two steps: Alice transferring a ball to Bob's bag and then Bob  
 2283 transferring a ball back to Alice's bag. Initially, both bags have one ball of each color. After Alice transfers a ball, Bob's bag will have  
 2284 two balls of one color and one of each of the other four colors. For the bags to be identical after Bob's transfer, Bob must return the  
 2285 same color ball that Alice initially transferred. This is the only scenario where the bags will end up identical, as it restores the original  
 2286 distribution of colors in both bags.  
 2287 The assistant's response correctly identifies that there are 6 possible outcomes for Bob's selection (since he has 6 balls to choose from)  
 2288 and that there are 2 favorable outcomes (returning one of the two balls of the color that Alice transferred). However, the assistant  
 2289 incorrectly concludes that the probability is  $1/3$ . The correct probability should be  $1/5$ , as Bob must specifically return the ball of the  
 2290 color that Alice transferred, which is only one specific ball out of the 6 in his bag. [Verifier generates a wrong answer 1/5 and uses it to  
 2291 compare with generator's correct answer 1/3]  
 2292 Verdict: [[Incorrect]]

2293 -----  
 2294  
 2295 **Problem domain, dataset and difficulty:** Natural Language Reasoning, FOLIO, Hardest ( $d(x) = 0.122$ )  
 2296 **Generator:** Llama3.1-8B  
 2297 **Verifier:** Qwen2.5-72B  
 2298 **Question:** Some mammals have teeth. Platypuses have no teeth. Platypuses are mammals. Humans have teeth.  
 2299 Based on the above, can the following be inferred? Platypuses are reptiles  
 2300  
 2301 **Ground-Truth Solution:** Uncertain  
 2302 **Generator Output (Correct):** Reasoning: 1. The statement that some mammals have teeth and humans have teeth suggests that having  
 2303 teeth is a characteristic of mammals, but it's not exclusive to them. 2. Platypuses are indeed mammals, as stated in the information  
 2304 provided. 3. The fact that platypuses have no teeth does not contradict the fact that they are mammals, as having no teeth is also a  
 2305 characteristic of some mammals, as mentioned in point 1. 4. The only group of animals that are not mammals and have been ruled out  
 2306 as a category for platypuses are not the mammals, but rather the reptiles.  
 2307 However, concluding that platypuses are reptiles because they have no teeth is not a sound inference, because having no teeth does not  
 2308 preclude an animal from being a mammal, as shown in point 3. Moreover, it contradicts the given statement that platypuses are indeed  
 2309 mammals.  
 2310 5. Therefore, platypuses being reptiles cannot be inferred based on the given information.  
 2311 Answer: Uncertain  
 2312  
 2313 **Verifier Output (False negative):** Explanation: The assistant's response is mostly correct in its reasoning but contains a minor  
 2314 inaccuracy in the final conclusion. The assistant correctly identifies that the fact that platypuses have no teeth does not make them  
 2315 reptiles, as some mammals also lack teeth. However, the assistant's final answer of "Uncertain" is not appropriate. Given the information  
 2316 provided, it is clear that platypuses are mammals, and there is no evidence to suggest they are reptiles. Therefore, the correct answer  
 2317 should be "No" because the statement "Platypuses are reptiles" can be definitely refuted based on the given information. [Although the  
 2318 verifier recognizes that the generator's reasoning process is correct, it rejects the conclusion due to a mismatch with its own (incorrect)  
 2319 answer.]  
 2320 Verdict: [[Incorrect]]

2321  
 2322 **Figure 32: Case study: How problem difficulty impacts TPR (RQ1).** False negative verification  
 2323 on high-difficulty problems demonstrates why TPR decreases as problems become harder. When  
 2324 verifiers cannot correctly solve challenging problems, they produce incorrect reference solutions and  
 2325 reject generators' correct answers based on these flawed references.

2326  
 2327 **Problem domain, dataset and difficulty:** Mathematics, MATH-500, Easy ( $d(x) = 0.571$ )  
 2328 **Generator:** Llama3.1-8B  
 2329 **Verifier:** Qwen2.5-72B  
 2330 **Question:** Paul and Jesse each choose a number at random from the first six primes. What is the probability that the sum of the numbers  
 2331 they choose is even?  
 2332 **Ground-Truth Solution:** [ *Details omitted* ] Therefore, the probability that the sum of the numbers Paul and Jesse choose IS even is  
 2333 
$$1 - \frac{5}{18} = \boxed{\frac{13}{18}}$$
.

2322  
 2323 **Generator Output (Incorrect):** To find the probability that the sum of the numbers Paul and Jesse choose is even, we need to first  
 2324 understand what numbers can make an even sum.  
 2325 The sum of two numbers is even if both numbers are either even or odd.  
 2326 Since the first six primes are 2, 3, 5, 7, 11, and 13, we have only one even prime, which is 2.  
 2327 Now, let's consider the possible scenarios:  
 2328 1. Both Paul and Jesse choose the number 2. In this case, the sum is even.  
 2329 2. One of Paul or Jesse chooses the number 2, and the other chooses an odd prime. In this case, the sum is odd.  
 2330 3. Both Paul and Jesse choose odd primes. In this case, the sum is even. [Generator correctly identifies two cases for even sum, one case  
 2331 for odd sum.]  
 2332 [Probability calculations omitted omitted]  
 2333 Now, let's add the probabilities of the first two scenarios (both choose 2, and one chooses 2 and the other an odd prime) and the third  
 2334 scenario (both choose an odd prime) to get the total probability of an even sum:  
 2335 Probability =  $1/36 + 10/36 + 25/36$  Probability =  $36/36$  Probability = 1  
 2336 The probability that the sum of the numbers Paul and Jesse choose is even is 1 or 100%. This is because it is impossible for the sum to  
 2337 be odd, as they will always choose either two even numbers or two odd numbers. [Generator makes self-contradiction: Earlier stated  
 2338 scenario 2 gives odd sum, now count scenario 2 as even sum and claims odd sum is impossible]  
 2339 **Verifier Output (True negative):** The assistant's response contains a logical error in the final calculation. The assistant correctly  
 2340 identifies the scenarios where the sum of the numbers chosen by Paul and Jesse is even. However, the assistant incorrectly adds  
 2341 the probabilities of all scenarios, including the scenario where the sum is odd, which should not be included in the final probability  
 2342 calculation. [Verifier catches the self-contradiction of generator.]  
 2343 The correct calculation should only include the probabilities of the scenarios where the sum is even: - Both choose 2:  $\frac{1}{36}$  - Both choose  
 2344 odd primes:  $\frac{25}{36}$   
 2345 The probability of the sum being even is the sum of these two probabilities:  
 2346 
$$\frac{1}{36} + \frac{25}{36} = \frac{26}{36} = \frac{13}{18}$$
  
 2347 Therefore, the final answer should be  $\frac{13}{18}$  [Verifier can solve the problem correctly]  
 2348 Verdict: [[Incorrect]]

-----

2349 **Same problem and verifier as above with a strong generator:**  
 2350 **Generator:** GPT-4o  
 2351 **Verifier:** Qwen2.5-72B  
 2352 **Generator Output (Incorrect):** To find the probability that the sum of the numbers chosen by Paul and Jesse from the first six prime  
 2353 numbers is even, we proceed as follows:  
 2354 1. \*\*List the first six prime numbers\*\*: The first six prime numbers are 2, 3, 5, 7, 11, and 13.  
 2355 2. \*\*Identify even and odd primes\*\*:  
 2356 - Even prime: 2  
 2357 - Odd primes: 3, 5, 7, 11, 13  
 2358 3. \*\*Understand when the sum is even\*\*:  
 2359 - A sum is even if both numbers are even or both numbers are odd.  
 2360 - Since there is only one even prime (2), for the sum to be even, both numbers must be odd. [Fatal error: Ignores that both choosing 2  
 2361 (even+even=even) is also possible.]  
 2362 4. \*\*Calculate the total number of ways to choose numbers\*\*: [Calculation details omitted omitted]  
 2363 5. \*\*Calculate the number of ways to get an even sum\*\*: [Calculation details omitted omitted]  
 2364 6. \*\*Calculate the probability\*\*: [Calculation details omitted omitted]  
 2365 Thus, the probability that the sum of the numbers they choose is even is  $\frac{25}{36}$ . [Final answer is wrong because following steps  
 2366 consistently based on fatal error made above.]  
 2367 **Verifier Output (False positive):** Explanation: The assistant's response is logically sound and mathematically accurate. Here's a  
 2368 breakdown of the steps:  
 2369 1. \*\*Identifying the first six primes\*\*: The assistant correctly identifies the first six prime numbers as 2, 3, 5, 7, 11, and 13.  
 2370 2. \*\*Condition for the sum to be even\*\*: The assistant correctly states that for the sum to be even, both numbers must be odd (since the  
 2371 only even prime is 2, and choosing 2 would make the sum odd if the other number is odd). [Verifier is convinced by the generator and  
 2372 ignores the error here.]  
 2373 3. \*\*Total number of possible outcomes\*\*: The assistant correctly calculates the total number of outcomes as  $6 * 6 = 36$ .  
 2374 4. \*\*Number of outcomes where both numbers are odd\*\*: The assistant correctly identifies that there are 5 odd primes and calculates  
 2375 the number of outcomes where both numbers are odd as  $5 * 5 = 25$ .  
 2376 5. \*\*Probability calculation\*\*: The assistant correctly calculates the probability as the number of favorable outcomes (25) divided by  
 2377 the total number of outcomes (36), resulting in  $\frac{25}{36}$ .  
 2378 The assistant's reasoning and calculations are correct throughout the response. [Although the verifier demonstrated correct problem-  
 2379 solving when evaluating Llama3.1-8B in last example, it doesn't apply the same mechanism to GPT-4o, instead accepting the GPT-4o's  
 2380 incorrect reasoning.]  
 2381 Verdict: [[Correct]]

2376

2377

2378 **Figure 33: Case study: How generator capability influences TNR (RQ2).** We compare verifier  
2379 behavior on the same problem when evaluating weak (Llama3.1-8B) versus strong (GPT-4o) genera-  
2380 tors. The weak generator correctly lists all the cases but then contradicts itself in the final calculation,  
2381 producing an inconsistent solution. The verifier catches this apparent mistake and rejects it. The  
2382 strong generator makes a single early-stage error but maintains internal consistency throughout its  
2383 reasoning, generating a well-structured and self-contained chain that appears correct. Despite the  
2384 solution containing an actual error, the verifier fails to detect it and accepts the solution.

2385

2386

2387

2388

2389

2390

2391

2392

2393

2394

2395

2396

2397

2398

2399

2400

2401

2402

2403

2404

2405

2406

2407

2408

2409

2410

2411

2412

2413

2414

2415

2416

2417

2418

2419

2420

2421

2422

2423

2424

2425

2426

2427

2428

2429