

JaggedBench: Measuring Reasoning Invariance via Automated Weakness Exploration

Donghao Luo* Chenchen Zhang*[†] Siheng Li Zenan Xu Qi Yi Boyu Qiu
Weikang Wang Guanhua Huang Tao Yang Bo Zhou Alex Chan[†]

Tencent Hunyuan, Shenzhen, China

[†]Corresponding authors: {adamwzhang, alexjchen}@tencent.com

Abstract

Strong large language models (LLMs) often achieve high scores on many benchmarks, yet can fail on closely related—sometimes easier—problems near their apparent capability boundary, a phenomenon known as jagged intelligence that reveals reasoning weaknesses masked by aggregate scores. Existing evaluations of jagged intelligence rely on manually designed benchmark-adjacent rewrites, which are hard to scale and quickly saturate as models improve. To address this challenge, we propose Automated Weakness Exploration (AWE), a fully automated framework for constructing and maintaining jaggedness evaluations. Using AWE, we construct JAGGEDBENCH with 1,142 verified capability-adjacent variants and evaluate 24 state-of-the-art models. On the primary evaluation set AIME 2025, benchmark-adjacent rewrites induce an average OG→RW relative drop of approximately 25% in Avg@k across 24 models, with consistent degradation observed across math, code, science, and knowledge domains. We identify 18 recurring failure modes. Our analysis shows that increasing reasoning intensity and scaling model size improve average performance and reduce failure frequency, yet the underlying failure-mode categories largely persist. Overall these findings indicate that standard benchmark scores can present an overly optimistic view of reliability near the capability boundary. JAGGEDBENCH provides a more faithful and scalable lens for evaluating boundary robustness, which enables systematic diagnosis of boundary failures and it supports longitudinal tracking of persistent weaknesses as models evolve. We release the benchmark, code, and subsequent updates at <https://jaggedbench.github.io/>.

1 Introduction

Strong large language models (LLMs) often achieve high scores on standardized benchmarks

(Art of Problem Solving, 2024; Jain et al., 2024; Rein et al., 2023; Wang et al., 2024), creating an impression of steadily improving reasoning ability (OpenAI, 2024; Gemini Team, Google, 2025). Yet in real-world use, they often fail on problems closely related to those they can already solve on standardized benchmarks—an instance of jagged intelligence, where strong benchmark performance does not translate into reliable behavior near the model’s apparent capability boundary. It suggests that benchmark accuracy alone may not faithfully reflect the reasoning weaknesses that compromise reliability near the capability boundary.

A growing line of work seeks to quantify jagged intelligence and characterize failure modes exposing capability-adjacent weaknesses. A common approach creates benchmark-adjacent rewrites: for each original problem, researchers construct a closely matched rewrite preserving the intended solution path but designed to trigger a particular failure mode. Examples include symbol remapping (Mirzadeh et al., 2024), reframing the scenario (Huang et al., 2025), or making minor formatting changes (Sclar et al., 2023). When a model succeeds on the original but degrades substantially on the rewrite, the gap provides direct evidence of jagged intelligence and helps localize the corresponding reasoning weakness.

Despite their diagnostic value, rewrite-based evaluations are costly to extend and tend to concentrate on a small set of familiar failure modes. More critically, as models improve, popular manual rewrite styles are often absorbed or mitigated by training and post-training pipelines, causing rapid saturation and shrinking gaps. In practice, many widely used rewrite sets now yield substantially smaller drops on the strongest models (Figure 18 in Appendix), limiting their ability to expose current reasoning weaknesses. As a result, by the time a jaggedness benchmark is established, the failure modes it targets may already be partially mitigated,

*Equal contribution.

leaving limited diagnostic traction for identifying capability-adjacent weaknesses.

To address this challenge, we propose Automated Weakness Exploration (AWE), a fully automated framework for constructing and maintaining jaggedness evaluations, in which LLMs generate benchmark-adjacent rewrites to expose their own failure modes and to evaluate jagged intelligence. By replacing manual and intuition-driven rewrites with automated generation, AWE surfaces a broader, more diverse set of failure modes, including those beyond familiar human priors. Further, AWE is naturally updatable: as model capabilities improve, it can be rerun to refresh the evaluation set, reducing rapid saturation and sustaining the discovery of failure modes that remain active in frontier models.

Building on AWE, we introduce JAGGEDBENCH, a dynamic benchmark that operationalizes jagged intelligence as the relative performance drop from original problems (OG) to their benchmark-adjacent rewrites (RW). JAGGEDBENCH comprises OG–RW pairs where each rewrite preserves the intended solution path while triggering a failure mode corresponding to a specific weakness. Because JAGGEDBENCH can be refreshed via AWE, it keeps pace with improving models and mitigates rapid saturation in prior rewrite-based benchmarks. The current version contains 1,142 curated OG–RW pairs spanning 4 domains, each capturing where a model’s apparent competence breaks down.

We evaluate 24 frontier reasoning models on JAGGEDBENCH and draw three insights. 1) Jagged intelligence is widespread. On AIME 2025 (Art of Problem Solving, 2025) OG/RW pairs, benchmark-adjacent rewrites induce an average relative drop of about 25%. 2) We identify 18 recurring failure modes exposing deeper weaknesses, often elicited by conversational rewrites or added distractors common in real interactions but sparse in canonical benchmarks. 3) Increasing reasoning intensity, scaling model size, and scaling test-time compute improves performance and reduces failure frequency, yet underlying failure-mode categories largely persist. Overall, these results suggest standard benchmark scores can overstate reliability near the capability boundary, and JAGGEDBENCH offers a more faithful lens by quantifying OG/RW gaps and enabling continual discovery of active weaknesses.

2 Related Work

Benchmarks and Dynamic Evaluation. Standardized reasoning benchmarks such as AIME, MATH, GPQA, MMLU-Pro, and LiveCodeBench remain central to measuring mathematical, scientific, knowledge-intensive, and coding ability (Art of Problem Solving, 2024, 2025; Hendrycks et al., 2021; Rein et al., 2023; Wang et al., 2024; Jain et al., 2024). However, fast leaderboard saturation and potential benchmark exposure motivate more dynamic evaluation protocols that refresh tasks or compare models under changing conditions (Huang et al., 2025; Xu et al., 2025; Zheng et al., 2023). JAGGEDBENCH complements these efforts by starting from established benchmarks but evaluating their semantics-preserving neighborhoods, where small changes can expose weaknesses hidden by the original test set.

Jagged Intelligence and Robustness Evaluation. Jagged intelligence describes the coexistence of high benchmark scores and failures on closely related problems near a model’s apparent capability boundary. Prior robustness evaluations construct such variants in three main ways: human-designed surface perturbations (Wang and Zhao, 2024; Sclar et al., 2023; Gao et al., 2018; Jin et al., 2020), LLM-executed rewrites based on human-specified recipes (Lunardi et al., 2025; Huang et al., 2025), and template or functional variants for specific task formats (Xu et al., 2025; Mirzadeh et al., 2024; Srivastava et al., 2024). These approaches reveal important brittleness, but their transformation spaces are usually manually specified and often tied to a particular benchmark format. In contrast, AWE autonomously proposes and refines rewrite strategies conditioned on the seed problem’s latent solution logic, targeting reasoning structure rather than only surface form. Complementary to such input-level robustness evaluation, TRACE (Imani et al., 2026) decomposes multimodal STEM problems into Auxiliary Reasoning Sets and localizes errors along reasoning trajectories using consistency and First-Failure-Step metrics. Whereas TRACE diagnoses *where* a failure occurs within a fixed problem, AWE discovers semantics-preserving variants that *trigger* failures and measures the resulting OG–RW gap; the two perspectives are complementary, and TRACE-style trajectory decomposition could further localize the failures surfaced by AWE.

We emphasize that our contribution is not the general observation that LLMs are sensitive to

paraphrases, formatting, distractors, or semantics-preserving transformations, nor the mere combination of automated generation with robustness testing. Rather, AWE differs from prior paradigms—including paraphrase- and perturbation-based robustness testing (Zhu et al., 2023; Wang and Zhao, 2024; Lunardi et al., 2025), mathematically equivalent or template-based transformations (Mirzadeh et al., 2024; Hao et al., 2025; Golikov et al., 2026; Lam et al., 2025), and automated or dynamic benchmark construction (Li et al., 2025; Zhu et al., 2024; Huang et al., 2025)—along two dimensions. First, *automated exploration beyond fixed human priors*: AWE treats the rewrite strategy *itself* as the search object, autonomously proposing seed-conditioned strategies and updating a reusable pool of positive/negative experience from equivalence judgments and Solver outcomes, rather than staying within a fixed human-specified transformation inventory. Second, *renewable, weakness-targeted evaluation*: AWE searches the semantics-preserving neighborhood of a *fixed* seed problem for transformations that expose currently active weaknesses, producing paired OG–RW probes that can be refreshed—and whose modular construction models can be upgraded—as models improve, rather than constructing new tasks. A detailed comparison organized by assumptions/search objects, generation mechanisms, validation criteria, and empirical coverage is provided in Table 22 (Appendix I.8).

LLM-as-Judge and Synthetic Evaluation. Our validation stage also connects to LLM-as-judge methods, which use strong models to compare or score model outputs (Zheng et al., 2023). In AWE, LLM judges are not used to assign open-ended preference scores; they serve as conservative semantic-equivalence filters under a unanimity rule, followed by proof-oriented checks and human review. AWE also relates to LLM-driven data synthesis: Self-Instruct and Evol-Instruct style methods generate or evolve training examples (Wang et al., 2023; Xu et al., 2023; Luo et al., 2025, 2023), while newer methods let models design synthesis strategies themselves (Zeng et al., 2024; Yu et al., 2025). We adapt this idea from training-data generation to evaluation-data generation, filtering the resulting probes for equivalence, solvability, and diagnostic value.

3 Automated Weakness Exploration and JaggedBench Construction

Following previous works (Mirzadeh et al., 2024; Huang et al., 2025), we cast weakness exploration as generating *benchmark-adjacent* variants through *semantics-preserving* rewrites. Given a seed problem q , a rewrite strategy t produces a variant $q' = t(q)$ that preserves the underlying task and solution of q but is more likely to elicit incorrect reasoning. Unlike prior methods relying on manual design, we employ Automated Weakness Exploration (AWE) to autonomously explore a broad strategy space and iteratively discover effective failure-inducing rewrites.

3.1 Automated Weakness Exploration Framework

As shown in Figure 1, AWE is a closed-loop multi-agent pipeline that jointly enforces semantic preservation and searches for failure-inducing rewrites.

Rewriter: Strategy Generation and Variant Production. The *Rewriter* autonomously generates and proposes diverse weakness-exploration strategies ($t \in \mathcal{T}$) for a seed problem q , producing benchmark-adjacent variants q' . Rather than generating variants in a single step, it follows a structured procedure: analyzing the problem and its solution, abstracting a problem-solving template, proposing diverse high-level strategies, selecting a subset for a concrete rewriting plan with explicit equivalence constraints, and executing it with iterative checks to ensure solvability and solution preservation. This approach balances reliability with flexibility, allowing the Rewriter to leverage its reasoning and creativity to generate diverse and challenging variants.

Judgers: Equivalence Verification. After the Rewriter generates candidate variants, *Judgers* verify semantic equivalence to the original problem by inspecting modifications to ensure all constraints, quantities, and targets remain unchanged. We use multiple LLMs from different families and adopt a strict unanimity rule: a variant is *validated* only if all Judges independently confirm equivalence; otherwise, it is rejected (Lu et al., 2025).

Solvers: Failure-Inducing Assessment. For each validated variant, *Solvers* assess its likelihood of eliciting failures from strong reasoning models. Multiple state-of-the-art models are evaluated with several independent responses each. Overall

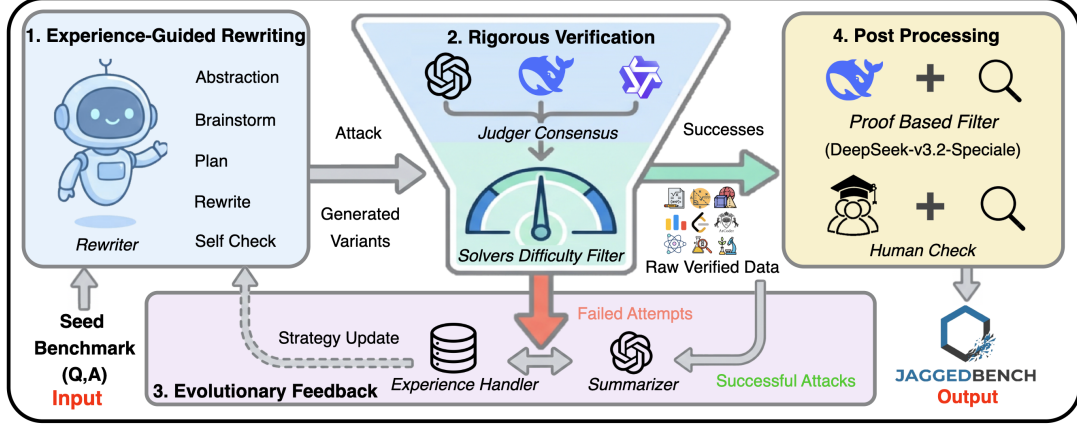


Figure 1: **Overview of the AWE Framework.** We formulate weakness discovery as closed-loop adversarial optimization. (1) **Experience-Guided Rewriting:** A *Rewriter* proposes rewrite strategies, producing candidate variants guided by past experience. (2) **Rigorous Verification:** Variants pass *Judger Consensus* for semantic equivalence and a *Solvers Difficulty Filter* to ensure failure-eliciting yet solvable. (3) **Evolutionary Feedback:** A *Summarizer* and *Experience Updater* extract positive/negative experience from outcomes to update the experience pool and steer subsequent rewriting. (4) **Post Processing:** Validated variants undergo *Proof-based Filter* and *Human Check*, yielding JAGGEDBENCH.

accuracy Acc is defined as the fraction of correct responses; lower Acc indicates stronger failure induction. Variants are retained if $0 < \text{Acc} \leq \tau$ ($\tau = 0.7$), ensuring they are both failure-inducing and solvable.

Summarizer: Outcome-Grounded Analysis and Attribution. The *Summarizer* analyzes the above rewrite–validate–solve trajectory, comparing the *Rewriter*’s intended failure-inducing effects with the outcomes realized by *Judgers* and *Solvers*. It distills these comparisons into concise reports highlighting both effective strategies (semantics-preserving and failure-inducing) and ineffective ones, providing a clear basis for experience extraction.

Experience Updater: Experience Extraction and Pool Management. The *Experience Updater* maintains a structured pool of positive (guidance) and negative (avoidance) experience, dynamically adding, refining, or removing entries based on empirical performance. In the next iteration, the *Rewriter* conditions its proposals on this pool, selectively leveraging empirically supported patterns as high-level cues, autonomously generating new strategies, and constraining those known to produce weak or invalid variants. To prevent convergence to local optima, we apply experience dropout—stochastically withholding the experience pool in a fraction of iterations—which forces the *Rewriter* to explore novel strategies beyond those already

validated. Repeated iterations of this feedback loop progressively refine the strategy space, improving both the yield and efficiency of failure-inducing, semantics-preserving variants.

Implementation Details of AWE Framework.

The *Rewriter*, *Summarizer*, and *Experience Updater* are implemented with DeepSeek-V3.1-Terminus (DeepSeek-AI, 2024). To mitigate single-model blind spots in both semantic equivalence verification and stress testing, we implement *Judgers* and *Solvers* as a cross-family ensemble consisting of DeepSeek-V3.1-Terminus (DeepSeek-AI, 2024), gpt-oss-120b (OpenAI, 2025), and Qwen3-235B-A22B-Thinking-2507 (Qwen Team, 2025). Algorithm 1 (see Appendix A.1) provides the full construction procedure. The prompts for each role are provided in Appendix J. We note that the AWE framework is model-agnostic; the current model choices reflect resource constraints rather than architectural requirements (see Appendix G for discussion).

3.2 JaggedBench Construction and Statistics

Using the AWE framework, we construct JAGGEDBENCH by applying semantics-preserving rewrites to *established* seed benchmarks spanning representative reasoning domains: mathematics (AIME 2024/2025; Math-500 (Art of Problem Solving, 2024, 2025; Hendrycks et al., 2021)), science (GPQA-Diamond (Rein et al., 2023)), general knowledge (MMLU-Pro (Wang et al., 2024)),

and code generation (LiveCodeBench (Jain et al., 2024)). This multi-domain design enables weakness discovery beyond the distribution of any single benchmark.

We rewrite the full AIME 2024 and AIME 2025 sets (5 distinct benchmark-adjacent variants per seed problem) and randomly selected subsets of the remaining seed benchmarks (1–5 variants per problem). In total, JAGGEDBENCH contains 1,142 validated instances. Table 1 summarizes the dataset statistics.

Source Benchmark (Domain)	#Seed (OG)	OG:RW Ratio	#Instances
AIME 2024 (Math)	30/30	1:5	150
AIME 2025 (Math)	30/30	1:5	150
Math-500 (Math)	56/500	1:1–5	59
GPQA-Diamond (Science)	38/198	1:1–5	46
MMLU-Pro (General)	104/12032	1:1–5	150
LiveCodeBench (Code)	131/131	1:1–5	587
Total	389	—	1142

Table 1: Dataset Statistics. JAGGEDBENCH contains 1,142 validated instances synthesized from six source benchmarks, covering mathematics, science, general knowledge, and coding.

3.3 Post-processing and Quality Control

Beyond the closed-loop filters, we apply post-construction quality control for answer correctness, uniqueness, and seed–rewrite equivalence. This combines proof-based verification, domain-specific objective checks (unit-test/judge checks for code; answer-key and extraction checks for multiple-choice or short-answer tasks), and expert review. Human reviewers audit AIME 2024/2025 per instance and conduct stratified spot checks elsewhere; confirmed issues (e.g., ~ 10 ambiguous AIME 2025 variants) are re-synthesized or edited. See Appendix B for details.

4 Experiments

The experiments are designed to demonstrate jagged intelligence using the proposed benchmark. Specifically, the results show a clear performance degradation on AWE-rewritten variants across diverse domains and model families (Section 4.2). This behavior remains consistent under both test-time scaling and parameter scaling (Section 4.3). We further discuss the broader implications of these findings and the necessity of dynamic evaluation in Appendix G, with additional analyses of AWE deferred to the Appendix due to space limitations.

4.1 Experimental Setup

We evaluate 24 LLMs under the original problems (OG) and benchmark-adjacent rewritten variants (RW). Our model set spans (i) multiple families (Qwen2.5/3; DeepSeek; GPT-oss; doubao; kimi and proprietary reasoning models like GPT and Gemini), (ii) a wide range of sizes (from small to flagship), and (iii) diverse training styles (instruction-tuned and reasoning-oriented), providing breadth and recency.

We distinguish between reliability and capability by reporting two metrics: **Avg@k** (average accuracy over k samples) as the primary measure of **reliability** (stability of the reasoning process), and **Pass@k** as an estimate of **potential capability** (feasibility of finding a solution), with $k = 8$ for proprietary models and $k = 16$ for open-source models. For any metric m , we report the absolute change $\Delta = m_{RW} - m_{OG}$ (in pp) and the relative change $\text{Rel } \Delta = \frac{m_{RW} - m_{OG}}{m_{OG}} \times 100\%$. We also define the OG→RW relative drop as $-\text{Rel } \Delta$. Unless otherwise specified, “drop” refers to the relative drop in Avg@k. Response length (Len) denotes the average token count (tokenizer: Qwen3-30B-A3B-Instruct-2507).

4.2 Jagged Intelligence in LLMs: Empirical Evidence and Failure Modes

Widespread Performance Degradation Across Benchmarks. Our experiments reveal jagged intelligence: models often do well on original problems (OG) yet degrade on benchmark-adjacent rewritten variants (RW).

On the primary evaluation set AIME 2025 (Table 2), almost all models exhibit significant OG→RW relative drops in both Avg@k and Pass@k. Even though AIME 2025 is a widely saturated benchmark, these drops indicate that conventional metrics mask substantial nearby weaknesses. It is worth noting that the maximum Pass@k reaches 1 for RW, confirming these problems remain well-posed and solvable. The consistent drops hold across all 24 evaluated models, including GPT-5.2, GPT-5.4, Gemini-3-Pro, Gemini-3.1-Pro, Doubao-Seed-1.8, and Kimi-K2-Thinking, which are entirely held out from the AWE construction pipeline—providing an out-of-pipeline validity check complementing the audits in Section 3.3.

Critically, this phenomenon is not an artifact of a specific dataset. As illustrated in Figure 2, we generalize the AWE framework to cover Math-500,

LiveCodeBench, GPQA, and MMLU-Pro. The OG→RW relative drops persist across these diverse domains—spanning math, code, science, and knowledge reasoning. This consistency confirms that jagged intelligence is a widespread, domain-agnostic vulnerability in current reasoning models, rather than a localized overfitting issue.

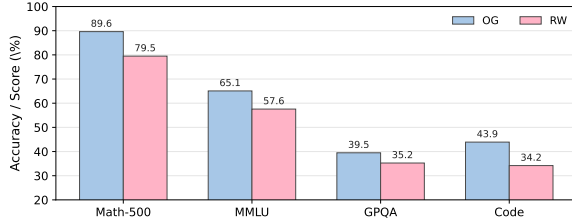


Figure 2: OG→RW relative drop in Avg@k across multiple domains, averaged over models. The drop is consistent across domains, suggesting domain-agnostic weaknesses rather than an AIME-specific artifact. See Tables 6 and 7 in Appendix for detailed results.

Diagnosing Weaknesses Behind Performance Degradation. To understand why OG→RW drops occur, we conduct a fine-grained strategy analysis, focusing on the AIME 2025 subset as a representative case study. We analyze the rewriting strategies discovered by AWE during automated exploration. Concretely, we cluster the final experience pool and distill 18 recurrent rewriting strategies that emerge during exploration (see Table 8 in Appendix for the complete taxonomy with definitions). We then label each rewritten problem with up to three strategy labels, and report the frequency of each label and the mean OG→RW relative drop associated with each strategy (Figure 3).

As shown in Figure 3, conversational style shifts and distractor-oriented rewrites yield the largest mean OG→RW relative drops in Avg@k, making them the two most severe weaknesses for current reasoning models.

We hypothesize that these high-impact strategies are underrepresented in current training data: most reasoning models are trained primarily on clean and formally stated math problems, whereas everyday phrasing and explicit distractor injection are relatively rare. Consequently, models may commit quickly to a plausible reasoning trajectory on well-structured inputs. When distractors are introduced, models can misinterpret redundant or misleading information as genuine constraints or subgoals; once such spurious cues enter intermediate conclusions, subsequent reasoning may become

anchored to an incorrect structure and progressively drift from the correct solution path. This matters in practice because these two weaknesses closely resemble day-to-day user queries, which are often expressed in everyday language and may include abstraction, redundancy, or misleading hints rather than clean, exam-style formulations.

4.3 Effects of Reasoning and Scaling on Jagged Intelligence

We analyze how reasoning and scaling affect jagged intelligence along three complementary dimensions: absolute performance on OG and RW (Avg@k), reliability under benchmark-adjacent stress testing as measured by the OG→RW relative drop, and the structure of failures as captured by failure-mode distributions.

To quantify changes in failure structure across adjacent reasoning or scaling settings, we further compute the Jensen–Shannon (JS) divergence between the corresponding failure-mode distributions. Each distribution is estimated from the top-10 RW instances with the largest OG→RW relative drops for each model. JS is computed with base-2 logarithm and normalized to $[0, 1]$, and we categorize divergence values into Small (0–0.06), Medium (0.06–0.12), and Large (≥ 0.12) regimes for interpretability.

4.3.1 Reasoning Strength and Failure Stability

Impact of Reasoning Strength. We analyze the effect of reasoning strength by comparing model variants with different levels of thinking intensity (e.g., DeepSeek Non-Thinking vs. Thinking, GPT-OSS Low vs. High). As shown in Table 3, increasing reasoning effort yields coordinated gains in both capability and reliability across most evaluated families. Stronger reasoning models not only achieve higher absolute accuracy on both OG and RW, but also consistently exhibit reduced OG→RW relative drops. For instance, DeepSeek-V3.2 reduces its relative drop from 30.43% to 17.13% when thinking is enabled, and GPT-OSS-120b shows a similar improvement (31.90% to 17.37%) as reasoning scales from low to high. For GPT-OSS-120b, the corresponding 14.53 pp reduction in the absolute OG→RW drop is statistically significant (paired-bootstrap 95% CI $[6.87, 22.26]$, $p < 10^{-4}$). Gemini-3-Pro is an exception: it exhibits high reliability even at low intensity (relative drop $\approx 8\%$), leaving limited headroom for fur-

Model	OG Avg@k (%)	RW Avg@k (%)	Δ Avg@k (pp)	Rel Δ Avg@k (%)	OG Pass@k (%)	RW Pass@k (%)	Δ Pass@k (pp)	Rel Δ Pass@k (%)	OG Len (tok)	RW Len (tok)
Open-source instruct model										
Qwen2.5-14B-Instruct (2024)	11.79	5.62	-6.17	-52.33	31.33	27.33	-4.00	-12.77	991.6	999.5
Qwen2.5-32B-Instruct (2024)	14.25	7.08	-7.17	-50.32	34.00	24.00	-10.00	-29.41	885.3	914.2
Qwen3-4B-Instruct-2507 (2025)	46.08	32.12	-13.96	-30.30	75.33	61.33	-14.00	-18.58	6928.4	7530.5
Qwen3-30B-A3B-Instruct-2507 (2025)	61.62	45.17	-16.45	-26.70	84.00	71.33	-12.67	-15.08	6648.7	7970.5
Qwen3-235B-A22B-Instruct-2507 (2025)	68.71	52.29	-16.42	-23.90	82.67	76.67	-6.00	-7.26	7004.1	8662.5
Open-source reasoning model										
QwQ-32B (2025)	69.50	45.38	-24.12	-34.71	85.33	71.33	-14.00	-16.41	16928.1	18284.6
DeepSeek-V3.1-Terminus (Thinking) (2024)	88.79	59.27	-29.52	-33.25	96.55	91.28	-5.27	-5.46	17834.8	27194.2
Qwen3-14B (2025)	71.54	47.79	-23.75	-33.20	85.33	71.33	-14.00	-16.41	17803.7	18979.2
Qwen3-32B (2025)	72.54	48.67	-23.87	-32.91	89.33	77.33	-12.00	-13.43	17747.7	18981.9
Qwen3-4B-Thinking-2507 (2025)	80.33	55.67	-24.66	-30.70	90.67	82.67	-8.00	-8.82	22599.3	25795.6
DeepSeek-R1-Distill-Qwen-32B (2025a)	53.79	37.67	-16.12	-29.97	80.67	66.00	-14.67	-18.19	13400.9	14626.3
Qwen3-8B (2025)	67.17	47.54	-19.63	-29.22	84.00	74.67	-9.33	-11.11	20041.4	20931.7
Qwen3-30B-A3B-Thinking-2507 (2025)	85.46	61.75	-23.71	-27.74	94.00	89.33	-4.67	-4.97	18850.7	23089.3
Qwen3-235B-A22B-Thinking-2507 (2025)	92.03	66.53	-25.50	-27.71	96.55	97.99	+1.44	+1.49	22968.8	30896.1
DeepSeek-R1-Distill-Qwen-14B (2025a)	48.67	35.50	-13.17	-27.06	74.67	66.00	-8.67	-11.61	13779.7	14161.2
gpt-oss-120b (High) (2025)	94.61	78.36	-16.25	-17.18	100.00	96.64	-3.36	-3.36	18690.3	23814.9
DeepSeek-V3.2 (Thinking) (2025b)	92.92	77.00	-15.92	-17.13	100.00	99.33	-0.67	-0.67	17077.4	24614.1
Kimi-K2-Thinking (2025)	94.17	80.42	-13.75	-14.60	100.00	98.00	-2.00	-2.00	28195.0	36607.6
DeepSeek-V3.2-Special (Thinking) (2025b)	93.97	86.62	-7.35	-7.82	100.00	100.00	+0.00	+0.00	24009.2	33014.7
Proprietary reasoning model										
Doubao-Seed-1.8 (High) (2025)	83.75	66.17	-17.58	-20.99	93.33	87.33	-6.00	-6.43	20382.9	24584.7
Gemini-3.1-Pro (High) (2026)	97.42	88.38	-9.04	-9.28	100.00	96.67	-3.33	-3.33	11379.4	15592.6
GPT-5.2 (High) (2025)	99.17	90.13	-9.04	-9.12	100.00	97.33	-2.67	-2.67	10728.7	15244.9
Gemini-3-Pro (High) (2025)	95.00	86.49	-8.51	-8.96	100.00	96.67	-3.33	-3.33	14420.9	16534.8
GPT-5.4 (xHigh) (2026)	99.42	94.97	-4.45	-4.48	100.00	100.00	+0.00	+0.00	10127.2	13659.6

Table 2: AIME 2025 performance on Original (OG) vs. benchmark-adjacent rewritten variants (RW), sorted by drop in Avg@k. We report Avg@k and Pass@k, as well as absolute (Δ , in pp) and relative (Rel Δ , in %) changes from OG to RW, and response length (Len, in tokens). See Section 4.1 for metric definitions and the evaluation protocol.

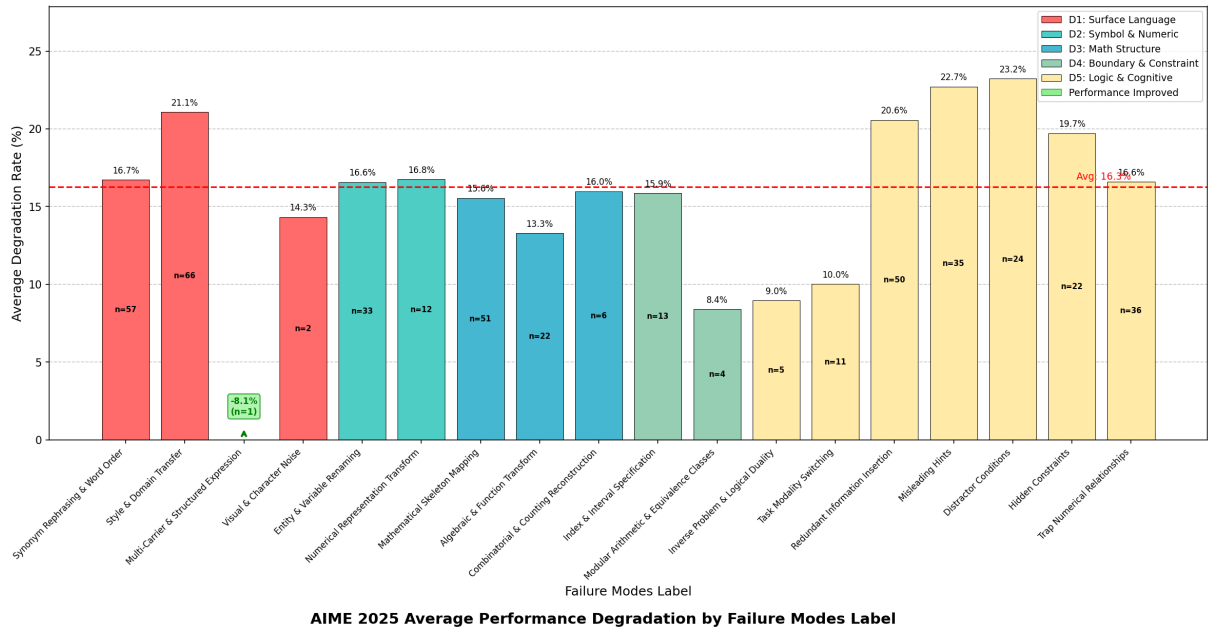


Figure 3: Bars report the average OG→RW relative drop in Avg@k for each AIME 2025 rewriting strategy, aggregated over problems labeled with that strategy (higher bars indicate larger average degradation). Bar annotations show the number of labeled instances (n); colors indicate the strategy family; the dashed line marks the overall average across strategies that degrade performance on average, and strategies that improve performance on average are indicated with a green callout and arrow.

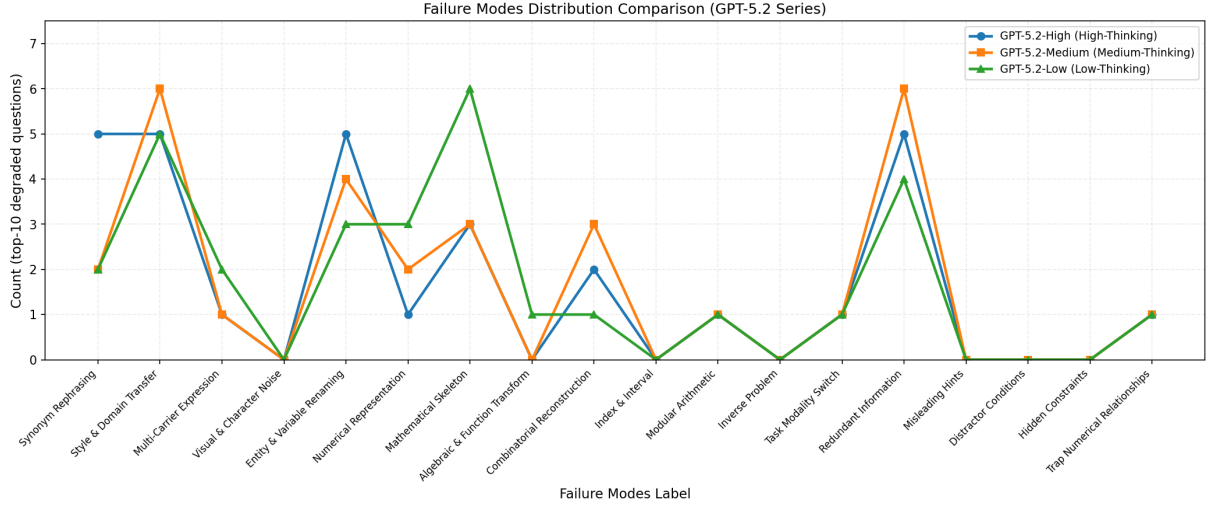


Figure 4: Failure mode distribution comparison of different reasoning level in GPT-5.2. Despite the reduction in total errors with higher reasoning levels (High-Thinking), the relative distribution of failure modes remains structurally invariant compared to Low-Thinking, as quantified by the low JS divergence.

ther gains from increased reasoning. These within-model controls are directly actionable: they indicate that selecting a higher reasoning mode, and (as shown next) allocating a larger sampling budget, can measurably improve robustness for reliability-sensitive deployments.

Model/Type	K	OG Avg@K	RW Avg@K	Rel. Drop (%)↓	JS (adj.)
DeepSeek-V3.1-Terminus					
Non-Thinking	16	53.96	33.58	37.77	0.0687
Thinking	16	89.17	59.54	33.24	
DeepSeek-V3.2					
Non-Thinking	16	57.08	39.71	30.43	0.1303
Thinking	16	92.92	77.00	17.13	
GPT-OSS-120b					
Low-Thinking	16	49.38	33.62	31.90	Low-Med: 0.0600 Med-High: 0.0250
Medium-Thinking	16	75.42	52.50	30.39	
High-Thinking	16	94.99	78.49	17.37	
GPT-5.2					
Low-Thinking	8	87.92	71.25	18.96	Low-Med: 0.0557 Med-High: 0.0260
Medium-Thinking	8	96.67	84.67	12.41	
High-Thinking	8	99.17	90.13	9.12	
Gemini-3-Pro					
Low-Thinking	8	84.10	77.31	8.07	0.1132
High-Thinking	8	95.00	86.49	8.96	

Table 3: OG vs. RW Avg@k and OG→RW relative drop for different reasoning settings. JS (adj.) reports the JS divergence between failure-mode distributions of adjacent thinking levels within each model; rows are ordered by increasing thinking intensity. Detailed visual comparisons of failure mode distributions can be found in Appendix E.2.

Structural Invariance of Failure Modes. Critically, despite these reliability gains, the failure-mode structure remains largely invariant. The JS divergence between failure distributions of adjacent reasoning levels is consistently small (mostly < 0.07 , with DeepSeek-V3.2 showing moderate values around 0.13). This indicates that scaling reasoning strength primarily improves reliability by suppressing the *frequency* of errors rather than fundamentally altering the underlying weakness structure. The models become better at avoiding known pitfalls, but the types of problems that trip them up remain structurally similar. This structural invariance is visualized in Figure 4 using GPT-5.2 as a representative case study. As shown, while the Low-Thinking and High-Thinking variants exhibit different total failure counts, the relative distribution of failure modes (the "shape" of the histogram) remains strikingly similar across reasoning levels.

4.3.2 Model Size Scaling

We compare Qwen models across scales (4B, 30B, 235B) in Figure 5. While scaling consistently improves absolute performance and reduces OG→RW drops, it does not fundamentally remove underlying weaknesses. The JS divergence be-

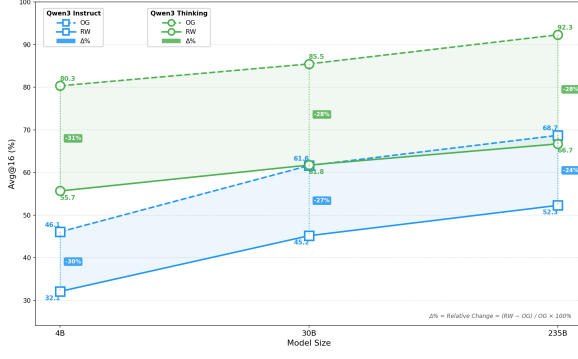


Figure 5: Avg@ k on OG and RW, and the drop, across model sizes in the Qwen Instruct and Qwen Thinking families.

Setting	235B vs. 30B	30B vs. 4B
Thinking (Failure Mode JS)	0.1087	0.0901
Instruct (Failure Mode JS)	0.0790	0.0224

Table 4: JS divergence of failure-mode distribution across model sizes. Detailed visual comparisons of failure mode distributions can be found in Appendix E.3.

tween failure-mode distributions across sizes remains small-to-medium (0.022–0.109, Table 4), implying that large models retain similar weakness profiles to smaller ones. Scaling primarily reduces error trigger frequency rather than eliminating the root vulnerabilities.

4.3.3 Parallel Inference Scaling

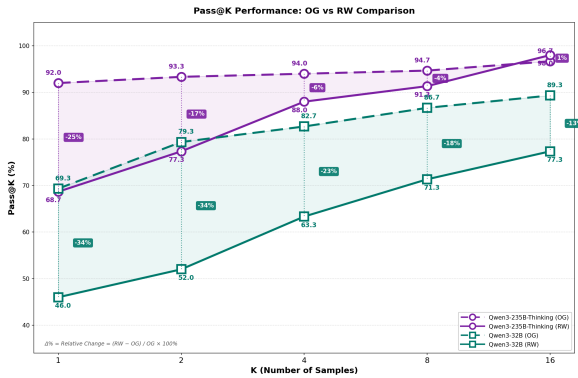


Figure 6: Pass@ k on OG and RW as a function of sampling attempts k , together with the corresponding drop in Pass@ k . For readability, we plot two representative models here and include results for all models in Appendix.

We analyze parallel sampling inference by measuring Pass@ k with increasing sampling attempts k (Figure 6). As k increases, the OG→RW relative drop consistently narrows. This suggests that

jagged intelligence often manifests as a probabilistic instability ("soft" failure) rather than a fundamental lack of capability ("hard" failure). Specifically, the rewrites introduce distractors that increase the likelihood of the model deviating from the correct solution path, thus requiring more sampling attempts to successfully recover the valid trajectory. However, the drop does not vanish even at high k , indicating that some failures stem from persistent grounding errors that compute alone cannot resolve.

This finding implies that the core challenge is often *stability* rather than *competence*. Consequently, future post-training should prioritize reasoning invariance alignment—using AWE-style adversarial variants to penalize semantic sensitivity during Reinforcement Learning—to convert these latent but unstable capabilities into reliable performance.

5 Conclusion

This work characterizes *jagged intelligence*, where state-of-the-art reasoning models fail on semantics-preserving variations of problems they can solve. We introduce Automated Weakness Exploration (AWE), a framework that treats weakness discovery as automated adversarial optimization, and construct JAGGEDBENCH, a multi-domain benchmark that induces an average OG→RW relative drop of approximately 25% on AIME 2025 and surfaces 18 recurring failure modes across frontier models. Our analysis shows that scaling and enhanced reasoning reduce error frequency but do not rectify underlying failure types, which remain structurally invariant across model sizes. Notably, reasoning intensity improvements within a model yield stable suppression of weaknesses, whereas switching between models induces volatile shifts in failure composition. These results underscore that standard benchmark scores often mask reasoning brittleness near capability boundaries.

Building on these findings, AWE and JAGGEDBENCH offer auditable, refreshable probes for future post-training and evaluation. Because each probe remains tied to a source problem and an observable OG/RW gap, the benchmark can support targeted invariance training and longitudinal audits as models evolve.

Limitations

This work evaluates reasoning robustness through benchmark-adjacent rewrites of tasks with objectively verifiable answers. We deliberately focus the current empirical study on this setting because it enables controlled checks of answer correctness, uniqueness, and seed–rewrite equivalence, providing a reliable first setting in which to instantiate and evaluate our two contributions: AWE, an automated weakness-exploration framework that can be rerun and refreshed as models improve, and JAGGEDBENCH, which operationalizes jagged intelligence through the gap between original problems and their semantics-preserving rewrites. As a consequence, our conclusions may not directly transfer to open-ended generation, multilingual or multimodal interaction, or domains where semantic equivalence is inherently subjective. Extending AWE to such subjective reasoning settings is an important direction for future work, but it will require task-specific verifiers and calibrated human or LLM judgments in place of the objective oracles used here. For the same reason, we do not claim that the present instantiation has been validated on complex agentic tasks; extending AWE to agentic settings is a promising direction, most naturally in environments with objective feedback, where goal and constraint preservation can be checked by task-specific executors or environment-based verifiers, while more open-ended agentic tasks would additionally require calibrated human or LLM judgments.

JAGGEDBENCH is constructed with a specific AWE configuration and therefore reflects the Rewriter, Judger, Solver, verifier, and sampling choices used in this study. We mitigate obvious defects with cross-family verification, proof-oriented checks, and human review, but future work should expand independent human auditing and study how the discovered rewrite strategies vary under different construction models and thresholds.

Finally, JAGGEDBENCH should be viewed as a renewable diagnostic benchmark rather than a permanent leaderboard. Public release can eventually reduce novelty through benchmark contamination, and repeated benchmark refreshes incur inference cost; both issues motivate versioned releases and transparent reporting of construction settings.

Ethical Considerations

Intended use and societal impact. The intended use of AWE and JAGGEDBENCH is diagnostic: to reveal reasoning brittleness that can be hidden by high benchmark scores and to support more reliable evaluation before deployment in coding, education, and scientific-assistance workflows. Results should be interpreted as aggregate robustness trends rather than isolated failure anecdotes.

Release and misuse safeguards. AWE-style rewriting could be misused to cherry-pick failures or create misleading model comparisons, and public benchmark variants may eventually be absorbed into training data. To support auditability and reduce this risk, releases should be versioned and accompanied by the original seeds, verification metadata, model versions, and filtering criteria, with future held-out splits kept separate from exposed instances. The tasks are derived from public reasoning benchmarks and contain no personal or sensitive user data; code-generation items should follow the licenses of their source benchmarks. Human review is limited to correctness, uniqueness, and seed–rewrite equivalence, with separated review roles and group adjudication for ambiguous cases.

References

- Art of Problem Solving. 2024. 2024 AIME I and II. Art of Problem Solving Wiki. https://artofproblemsolving.com/wiki/index.php/2024_AIME_I and https://artofproblemsolving.com/wiki/index.php/2024_AIME_II.
- Art of Problem Solving. 2025. 2025 AIME I and II. Art of Problem Solving Wiki. https://artofproblemsolving.com/wiki/index.php/2025_AIME_I and https://artofproblemsolving.com/wiki/index.php/2025_AIME_II.
- ByteDance Seed. 2025. Seed1.5-Thinking: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*.
- DeepSeek-AI. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.
- DeepSeek-AI. 2025a. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- DeepSeek-AI. 2025b. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

- Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 50–56. IEEE.
- Gemini Team, Google. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.
- Pavel Golikov, Evgenii Opryshko, Gennady Pekhimenko, and Mark C. Jeffrey. 2026. Robust reasoning benchmark. *arXiv preprint arXiv:2604.08571*.
- Google DeepMind. 2026. Gemini 3.1 Pro model card. <https://deepmind.google/models/model-cards/gemini-3-1-pro/>.
- Yuren Hao, Xiang Wan, and ChengXiang Zhai. 2025. An investigation of robustness of llms in mathematical reasoning: Benchmarking with mathematically-equivalent transformation of advanced mathematical problems. *arXiv preprint arXiv:2508.08833*.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.
- Shulin Huang, Linyi Yang, Yan Song, Shuang Chen, Leyang Cui, Ziyu Wan, Qingcheng Zeng, Ying Wen, Kun Shao, Weinan Zhang, Jun Wang, and Yue Zhang. 2025. Thinkbench: Dynamic out-of-distribution evaluation for robust llm reasoning. *arXiv preprint arXiv:2502.16268*.
- Shima Imani, Seungwhan Moon, Lambert Mathias, Lu Zhang, and Babak Damavandi. 2026. TRACE: A framework for analyzing and enhancing stepwise reasoning in vision-language models. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3611–3625, Rabat, Morocco. Association for Computational Linguistics.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Live-codebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.
- Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8018–8025.
- Kimi Team. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.
- Man Ho Lam, Chaozheng Wang, Jen-tse Huang, and Michael R. Lyu. 2025. Codecrash: Exposing llm fragility to misleading natural language in code reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Xiang Lisa Li, Farzaan Kaiyom, Evan Zheran Liu, Yifan Mai, Percy Liang, and Tatsunori Hashimoto. 2025. Autobench: Towards declarative benchmark construction. In *International Conference on Learning Representations (ICLR)*.
- Llama Team, AI @ Meta. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Jack Lu, Ryan Teehan, Jinran Jin, and Mengye Ren. 2025. When does verification pay off? a closer look at llms as solution verifiers. *arXiv preprint arXiv:2512.02304*.
- Riccardo Lunardi, Vincenzo Della Mea, Stefano Mizzaro, and Kevin Roitero. 2025. On robustness and reliability of benchmark-based evaluation of llms. *arXiv preprint arXiv:2509.04013*.
- Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, Yansong Tang, and Dongmei Zhang. 2025. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.
- Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.
- Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*.
- OpenAI. 2024. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- OpenAI. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.
- OpenAI. 2025. Update to GPT-5 system card: GPT-5.2. <https://openai.com/index/gpt-5-system-card-update-gpt-5-2/>.
- OpenAI. 2026. GPT-5.4 thinking system card. <https://openai.com/index/introducing-gpt-5-4/>.
- Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. 2025. A new era of intelligence with Gemini 3. <https://blog.google/products-and-platforms/products/gemini/gemini-3/>.
- Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

- Qwen Team, Alibaba Group. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Di-rani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*.
- Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2023. Quantifying language models’ sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. *arXiv preprint arXiv:2310.11324*.
- Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. 2025. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*.
- Saurabh Srivastava, Annarose M B, Anto P V, Shashank Menon, Ajay Sukumar, Adwaith Samod T, Alan Phil-
pose, Stevin Prince, and Sooraj Thomas. 2024. Func-
tional benchmarks for robust evaluation of reasoning
performance, and the reasoning gap. *arXiv preprint
arXiv:2402.19450*.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa
Liu, Noah A Smith, Daniel Khashabi, and Hannaneh
Hajishirzi. 2023. Self-instruct: Aligning language
models with self-generated instructions. In *Proceed-
ings of the 61st annual meeting of the association for
computational linguistics (volume 1: long papers)*,
pages 13484–13508.
- Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni,
Abhranil Chandra, Shiguang Guo, Weiming Ren,
Aaran Arulraj, Xuan He, Ziyang Jiang, Tianle Li, Max
Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue,
and Wenhui Chen. 2024. Mmlu-pro: A more robust
and challenging multi-task language understanding
benchmark. *arXiv preprint arXiv:2406.01574*.
- Yuqing Wang and Yun Zhao. 2024. Rupbench: Bench-
marking reasoning under perturbations for robustness
evaluation in large language models. *arXiv preprint
arXiv:2406.11020*.
- Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng,
Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin
Jiang. 2023. Wizardlm: Empowering large lan-
guage models to follow complex instructions. *arXiv
preprint arXiv:2304.12244*.
- Xin Xu, Jiaxin Zhang, Tianhao Chen, Zitong Chao,
Jishan Hu, and Can Yang. 2025. Ugmathbench: A
diverse and dynamic benchmark for undergraduate-
level mathematical reasoning with large language
models. *arXiv preprint arXiv:2501.13766*.
- Ping Yu, Jack Lanchantin, Tianlu Wang, Weizhe Yuan,
Olga Golovneva, Ilia Kulikov, Sainbayar Sukhbaatar,
Jason Weston, and Jing Xu. 2025. Cot-self-instruct:
Building high-quality synthetic prompts for rea-
soning and non-reasoning tasks. *arXiv preprint
arXiv:2507.23751*.
- Weihao Zeng, Can Xu, Yingxiu Zhao, Jian-Guang Lou,
and Weizhu Chen. 2024. Automatic instruction
evolving for large language models. *arXiv preprint
arXiv:2406.00770*.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan
Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin,
Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang,
Joseph E. Gonzalez, and Ion Stoica. 2023. Judging
llm-as-a-judge with mt-bench and chatbot arena. *Ad-
vances in Neural Information Processing Systems*,
36:46595–46623.
- Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang
Gong, Diyi Yang, and Xing Xie. 2024. Dyval: Dy-
namic evaluation of large language models for reason-
ing tasks. In *International Conference on Learning
Representations (ICLR)*.
- Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang,
Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Yue
Zhang, Neil Zhenqiang Gong, and Xing Xie. 2023.
Promptrobust: Towards evaluating the robustness of
large language models on adversarial prompts. *arXiv
preprint arXiv:2306.04528*.

A AWE Framework Details

This appendix provides comprehensive documentation of the AWE (Automated Weakness Exploration) framework. The AWE framework details are organized as follows:

1. **Section A.1:** Detailed algorithm for dataset construction
2. **Section H:** Qualitative case studies with complete problem statements
3. **Section J:** Complete prompt templates for all AWE components

For readers interested in the AWE framework implementation, we recommend reading these three sections in order: Algorithm $\rightarrow$ Case Studies $\rightarrow$ Prompt Templates.

A.1 Detailed Algorithm

B Details of Post-processing and Quality Control

The closed-loop pipeline described in Section 3.1 already enforces our core selection criteria: *semantic equivalence* and *failure-inducing yet solvable* difficulty. To further strengthen reliability, we apply additional quality checks that target two requirements crucial for benchmark-adjacent evaluation: (i) the released answer is correct and uniquely determined by the problem statement, and (ii) the rewritten variant remains semantics-preserving with respect to its seed problem.

Construction funnel. To make the filtering process transparent, Table 5 reports the AIME 2025 construction funnel by the number of candidate rewrites surviving each stage. *Parse pass* counts generated outputs that can be parsed into valid rewrite instances; *Judger pass* counts candidates for which all three cross-family Judges confirm equivalence; *Difficulty-filter pass* counts candidates with $\text{Acc} \leq \tau$ ($\tau = 0.7$, still including zero-accuracy candidates); *Solvable stress-test pass* counts candidates further satisfying $0 < \text{Acc} \leq \tau$, excluding those no internal Solver can solve; and *Final release* counts candidates passing proof-based and human verification, followed by balanced selection to retain five variants per seed. Of the 5,988 generated candidates, 648 pass both equivalence verification and the difficulty threshold; after excluding zero-accuracy candidates, 442

satisfy the solvable stress-test criterion, and 150 are released after post-verification and balanced selection across all 30 AIME 2025 seeds.

Construction stage	Count	Retention
Generated	5,988	—
Parse pass	5,376	89.8%
Judger pass	3,925	73.0%
Difficulty-filter pass ($\text{Acc} \leq 0.7$)	648	16.5%
Solvable stress-test pass ($0 < \text{Acc} \leq 0.7$)	442	68.2%
Final release	150	33.9%

Table 5: **AIME 2025 construction funnel.** Number of candidate rewrites surviving each filtering stage. “Retention” is the fraction retained from the previous stage; the final release applies balanced selection to keep five variants per seed across all 30 seeds.

Cross-domain filtering. The core AWE filtering pipeline is *shared* across all domains: every candidate undergoes the same parsing procedure, the same unanimous cross-family semantic-equivalence verification, and the same Solver difficulty criterion $0 < \text{Acc} \leq 0.7$. Only the final objective verification oracle is domain-specific: mathematics uses proof-based checks, code uses executable unit tests or benchmark judges, and multiple-choice or short-answer tasks use answer-key and extraction-consistency checks. Consistent with this, AIME receives instance-level human review, while the other domains receive stratified expert spot-checks.

Proof-based automated verification. We further strengthen reliability with a proof-driven verification signal that targets subtle issues beyond standard model-based gating. We employ deepseek-math-v2 (Shao et al., 2025) and deepseek-v3.2-speciale (DeepSeek-AI, 2025b) as verifiers due to their demonstrated strength in rigorous, long-form mathematical reasoning and proof verification. In particular, these models are trained to identify logical gaps, inconsistencies, and unverified steps rather than merely validating final answers, and can sustain iterative self-checking during the proof process. We leverage this capability to stress-test instances for hard-to-detect defects, such as latent answer non-uniqueness or subtle semantic drift from the seed problem. Concretely, we cast verification as two proof tasks: (1) proving that the released answer is both correct and *unique* for the problem, and (2) proving that the rewritten variant is *semantically equivalent* to its seed problem.

Algorithm 1: Final dataset construction via rewrite–validate–collect with local experience

Require: Seed set $\mathcal{D} = \{(q, a, id)\}$; rewrite trials N_r ; solver trials N_s ; thresholds $(\tau, r_{\text{majority}})$
Ensure: Final dataset $\mathcal{F}$

```

1:  $\mathcal{C} \leftarrow \emptyset$  {accepted candidates across the entire run}
2:  $stop \leftarrow \text{false}$ 
3: while not  $stop$  do
4:   for each  $(q, a, id) \in \mathcal{D}$  do
5:     Load/init experience pool  $E_{id}$ ; sample whether to provide  $E_{id}$  as conditioning context to REWRITER
6:      $\{(\tilde{q}_k, r_k)\}_{k=1}^{N_r} \leftarrow \text{REWRITER}(q, a, E_{id})$ 
7:     for  $k = 1$  to  $N_r$  do
8:        $\{(s_{k,j}, \text{rev}_{k,j})\}_j \leftarrow \text{JUDGERGROUP}(q, \tilde{q}_k, r_k)$ 
9:        $\text{pass}_k^{eq} \leftarrow [\forall j, s_{k,j} = 1]$ 
10:      if  $\text{pass}_k^{eq}$  then
11:         $\alpha_k \leftarrow \text{EVALUATEACCURACY}(\text{SOLVER}(\tilde{q}_k, a, N_s))$ 
12:        if  $\alpha_k \leq \tau$  then
13:           $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\tilde{q}_k, r_k, \{(s_{k,j})_j, \alpha_k\})\}$ 
14:        end if
15:      end if
16:    end for
17:     $(q^{eq}, q^{diff}) \leftarrow \text{SUMMARIZER}(\{r_k\}_k, \{(s_{k,j}, \text{rev}_{k,j})\}_{k,j}, \{\alpha_k\}_k)$ 
18:     $E_{id} \leftarrow \text{PROBLEMEXPRIENCEUPDATER}(id, E_{id}, q^{eq}, q^{diff})$ 
19:  end for
20:   $stop \leftarrow \text{CHECKSTOP SIGNAL}()$ 
21: end while
22: // Final post-filtering for the released dataset
23:  $\mathcal{F} \leftarrow \{x \in \mathcal{C} \mid 0 < \alpha_x \leq \tau\}$ 
24:  $\text{SAVEFINALDATASET}(\mathcal{F})$ 

```

For each instance and each task, we perform 8 independent proof attempts and record the verifier’s self-reported confidence. An instance is discarded if it is flagged as problematic with *high confidence* in more than two attempts for *either* task.

Domain-specific objective checks. Beyond LLM and proof-based verification, we apply deterministic or answer-key-based checks whenever the source benchmark provides an objective oracle. For code-generation tasks, each rewritten prompt is required to preserve the original input–output contract and expected interface, and variants that fail the corresponding unit-test suite or benchmark judge are rejected. For multiple-choice and short-answer science, knowledge, and math benchmarks, we preserve the original answer key or final answer and run answer-format/extraction consistency checks before release. These checks are not substitutes for semantic review, but they catch many non-equivalence errors that would otherwise survive text-only judgment.

Human consistency verification. Human review audits remaining semantic risks not covered by objective oracles. For the main analysis benchmarks (AIME 2024/2025), we perform per-instance human verification in addition to automatic checks. For the remaining seed benchmarks, we conduct stratified random spot-checking across domains and rewrite types with an expert review team (graduate-level or above; backgrounds in computer science, mathematics, or AI). The human review is split into two tasks and assigned to disjoint review-

ers to reduce leakage: one group verifies answer correctness and uniqueness, while another independently assesses semantic equivalence between the seed problem and the rewritten variant.

During this process, reviewers identified a small number of problematic instances. For example, in AIME 2025 we found roughly ten variants with potential ambiguity in the stated conditions. Such cases are escalated to group discussion to confirm whether an issue truly exists. Confirmed issues are handled according to severity: severely flawed instances are discarded and re-synthesized, while minor issues are corrected with targeted human edits to restore unambiguous semantics and verifiable correctness.

C Detailed Generalization Results

LiveCodeBench evaluation notes (v6, weeks 2502–2506). We report two aggregated views: **AT** (AtCoder) and **LC** (LeetCode). In the main paper (Figure 2), we visualize *AtCoder only* because AtCoder problems typically provide ≥ 10 private tests per instance, making the evaluation stricter and the OG→RW degradation more faithful. In contrast, the LeetCode split in LiveCodeBench v6 has limited or no private tests for many problems, which can make the judge signal overly permissive; as a result, LC often shows little (or even negative) “drop” that is primarily attributable to evaluation looseness rather than genuine robustness.

Model	Math-500		MMLU		GPQA-Diamond	
	OG	RW	OG	RW	OG	RW
Qwen3-30B-A3B-Thinking-2507	97.67	80.77	70.83	69.92	44.84	39.13
Qwen3-30B-A3B-Instruct-2507	95.76	69.54	50.17	45.92	29.62	26.09
Qwen3-4B-Thinking-2507	96.82	80.40	45.33	41.92	36.68	34.51
Qwen3-4B-Instruct-2507	90.89	74.95	46.17	38.33	32.61	26.63
Qwen3-32B	96.77	84.69	65.92	59.25	42.39	33.70
Qwen3-14B	96.13	88.29	68.33	62.00	42.12	39.67
Qwen3-8B	96.29	81.73	60.00	46.67	39.40	36.68
QwQ-32B	94.97	82.79	75.08	67.58	45.38	40.22
DeepSeek-R1-Distill-Qwen-32B	92.16	79.13	76.83	74.17	45.92	42.66
DeepSeek-R1-Distill-Qwen-14B	92.48	80.77	73.83	70.50	44.29	38.84
DeepSeek-R1-Distill-Llama-8B	83.05	71.24	64.83	63.17	36.68	33.70
Qwen2.5-32B-Instruct	79.18	68.38	74.83	68.83	41.58	40.22
Qwen2.5-14B-Instruct	77.12	66.74	71.33	47.42	38.59	33.15
Llama-3.1-8B-Instruct	42.53	36.86	67.67	50.25	32.61	29.08
Average	89.62	79.48	65.08	57.57	39.48	35.25

Table 6: Prevalence of jagged intelligence in Science, Knowledge and more Math domains. OG→RW relative drop in Avg@k across Math-500, MMLU, and GPQA-Diamond. Models are listed in the first column and scores follow.

Model	AT		LC	
	OG	RW	OG	RW
DeepSeek-R1-Distill-Llama-8B	28.05%	20.86%	22.45%	28.00%
DeepSeek-R1-Distill-Qwen-32B	34.15%	30.48%	28.57%	35.00%
Llama-3.1-8B-Instruct	18.29%	8.02%	8.16%	7.50%
QwQ-32B	51.22%	42.78%	51.02%	54.00%
Qwen2.5-14B-Instruct	29.27%	11.23%	18.37%	18.25%
Qwen2.5-32B-Instruct	30.49%	16.04%	32.65%	28.50%
Qwen2.5-72B-Instruct	31.71%	17.11%	26.53%	26.75%
Qwen2.5-Coder-32B-Instruct	31.71%	8.56%	20.41%	27.00%
Qwen3-14B	53.66%	49.73%	61.22%	54.50%
Qwen3-235B-A22B	54.88%	50.80%	63.27%	63.00%
Qwen3-235B-A22B-Instruct-2507	57.32%	49.73%	55.10%	65.25%
Qwen3-235B-A22B-Thinking-2507	67.07%	58.29%	77.55%	78.50%
Qwen3-30B-A3B	57.32%	48.13%	57.14%	56.50%
Qwen3-32B	56.10%	48.66%	65.31%	62.75%
Qwen3-4B-Thinking-2507	56.10%	46.52%	53.06%	50.00%
Qwen3-8B	45.12%	40.11%	44.90%	41.75%
Average	43.90	34.19	42.85	43.57

Table 7: Prevalence of jagged intelligence in Code domain. OG→RW relative drop in Avg@k across two aggregated views (AT and LC) in Livecodebench. Models are listed in the first column and scores follow.

D Taxonomy of 18 Rewriting Strategies

Table 8 provides the complete taxonomy of 18 rewriting strategies discovered by AWE during automated exploration on AIME 2025. Strategies are grouped into five families based on the type of transformation they apply.

E Full Results for Analysis in AIME 2025 OG-RW Pairs

E.1 Full Pass@k Results

We provide the full Pass@k results on AIME 2025 OG and RW in Figure 7 and 8.

E.2 Full Failure Mode Distribution Comparison of Different Reasoning Level

We provide the full results of failure mode distribution comparison of different reasoning level across multiple models in Figure 9, 10, 11, and 12. (Note: The comparison for GPT-5.2 is presented in the main text as Figure 4).

E.3 Full Failure Mode Distribution Comparison of Different Model Size

We provide the full results of failure mode distribution comparison of different model size across Qwen Instruct and Qwen Thinking families in Figure 13 and 14.

F Full Results for Analysis in AIME 2024 OG-RW Pairs

We provide the AIME 2024 performance comparison in Table 9 and the analysis of reasoning ability effects in Table 10. And the Pass@k results on AIME 2024 OG and RW are provided in Figure 16 and 17.

G Discussion

The prevalence of jagged intelligence across state-of-the-art models highlights a critical gap between benchmark performance and robust reasoning. Our findings via Automated Weakness Exploration (AWE) suggest a necessary paradigm shift in how we evaluate and improve reasoning models.

G.1 From Static Saturation to Dynamic Diagnosis

Current proprietary models (e.g., GPT-5.2) exhibit smaller gaps between original and rewritten problems compared to open-source models, likely due

to extensive post-training on diverse data. However, as model development becomes increasingly benchmark-aware, static benchmarks suffer from rapid saturation and “Goodhart’s Law” effects—losing their diagnostic power. Our results show that AWE restores this diagnostic capability. By continually generating benchmark-adjacent variants that evolve alongside model capabilities, AWE transforms evaluation from a static “scorecard” into a dynamic “stress test,” preventing the illusion of reliability created by memorization or overfitting to fixed test sets. While our current instantiation targets tasks with objectively verifiable answers, the framework itself is task-agnostic: extending it to more subjective reasoning settings—by pairing the closed-loop pipeline with task-specific verifiers and calibrated human or LLM judgments—is a natural next step for broadening this dynamic-diagnosis paradigm.

G.2 Beyond Human Priors: Uncovering the “Unknown Unknowns”

Prior stress tests typically rely on human-curated perturbations (e.g., variable renaming), which only probe known failure modes (“known unknowns”). In contrast, AWE autonomously discovers high-impact strategies that humans often overlook, such as conversational style shifts and distractor injection (Section 4.2). These strategies are particularly critical because they mirror real-world user interactions—imperfect, redundant, and informal—rather than the sanitized logic of exam problems. By surfacing these “unknown unknowns,” AWE ensures that models are tested against the complexity of deployment environments, not just the priors of researchers.

G.3 The Scaling Illusion: Suppression is Not Resolution

Perhaps our most sobering finding is the resilience of failure modes against scaling. While increasing model size and reasoning intensity (Thinking) significantly reduces error *frequency* (Table 3), it barely alters the *distribution* of failure types (Table 4). This suggests that scaling acts as a suppressor rather than a resolver: it helps models avoid known pitfalls more often but does not fundamentally patch the underlying reasoning holes. To truly resolve these weaknesses, we must move beyond passive scaling. We envision frameworks like AWE being integrated directly into the training loop—creating an “Adversarial Curriculum” for

Family	Strategy	Description
D1: Surface Language	Conversational & Story Framing Style & Format Change	Rewrites the problem in casual, narrative, or dialogue style Alters presentation format (e.g., bullet points, verbose phrasing)
D2: Symbol & Numeric	Notation & Symbol Substitution	Replaces standard notation with uncommon or domain-shifted symbols
	General & Mathematical Reformulation	Restates the problem using alternative mathematical formalisms
	Extreme & Non-standard Parametrization Encoding Representation Change	Uses boundary or unusual parameter values Switches numeric base, coordinate system, or encoding
D3: Math Structure	Computational & Answer Format Shift	Changes how the answer should be computed or expressed
	Multi-step Composition & Iterative Reasoning	Decomposes into chained sub-problems requiring sequential reasoning
	Abstract & Structural Transformation	Maps to an isomorphic but abstractly framed structure
	Modular & Intermediate Verification Representation & Operations Change	Requires verifying intermediate results before proceeding Alters the mathematical operations or representations used
D4: Boundary & Constraint	Precision & Edge Case Emphasis	Highlights boundary conditions or precision requirements
	Strict Definition & Formal Language	Adds rigorous formal definitions that obscure the core task
	Near-boundary Perturbation	Modifies parameters to values near critical thresholds
	Multi-constraint Composition Distractor & Complexity Injection	Combines multiple constraints that interact non-trivially Introduces irrelevant information or misleading cues
D5: Logic & Cognitive	Hidden Complexity	Embeds additional reasoning steps not apparent from surface reading
	Well-defined Completeness	Ensures all edge cases are explicitly specified (can simplify)

Table 8: Taxonomy of 18 rewriting strategies discovered by AWE, grouped into five families. For each strategy, we provide a brief description of the transformation it applies.

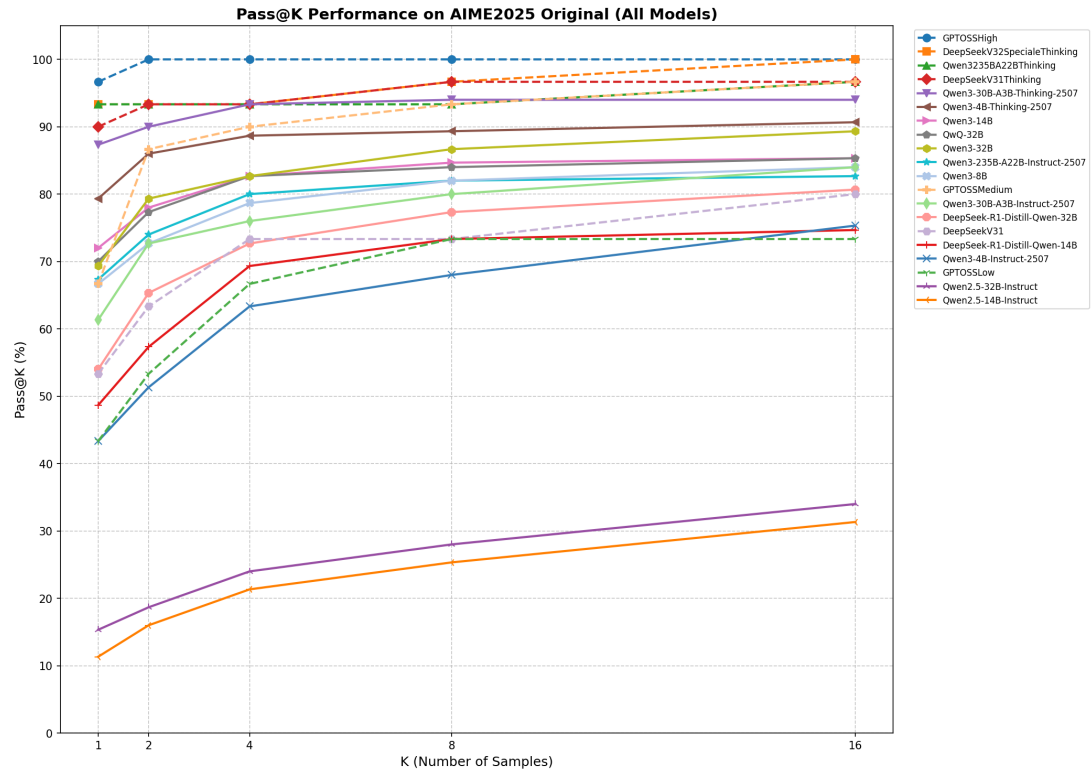


Figure 7: Full Pass@k performance in AIME 2025 OG.

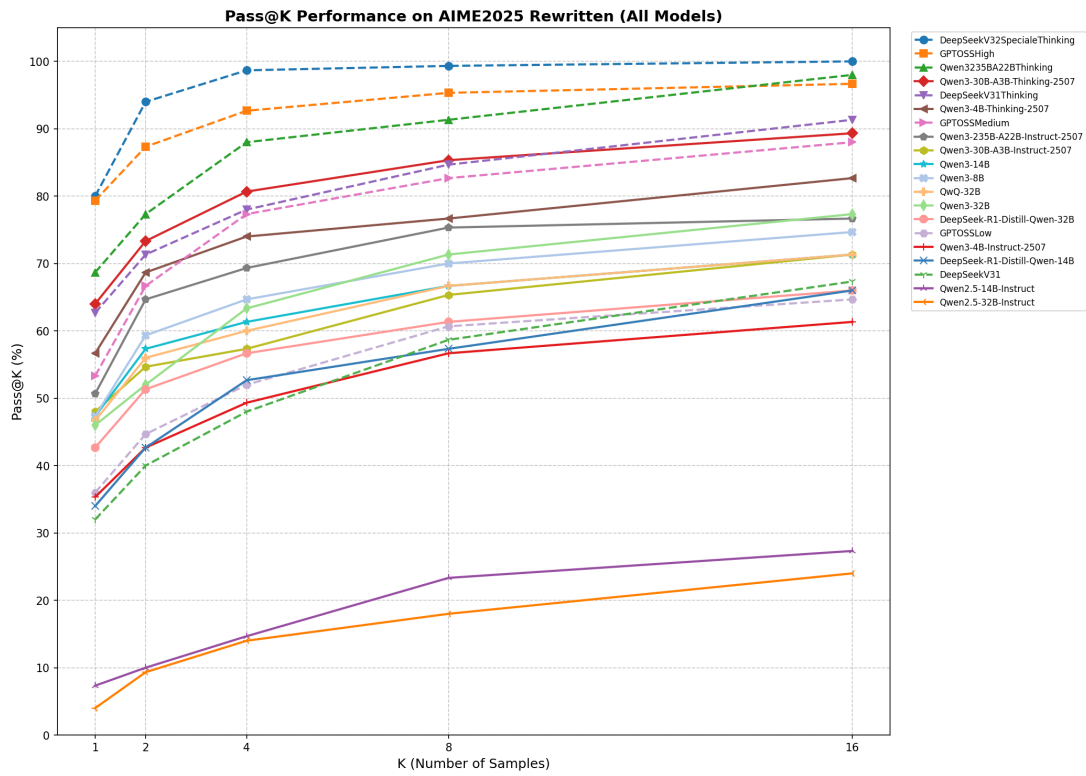


Figure 8: Full Pass@k performance in AIME 2025 RW.

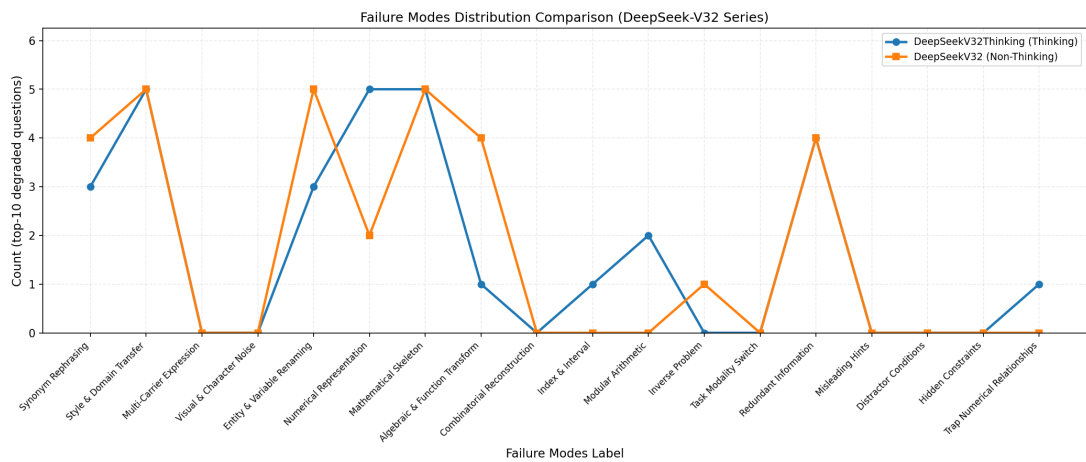


Figure 9: Failure mode distribution comparison of different reasoning level in Deepseek-V3.2. **Takeaway:** stronger reasoning reduces the overall failure frequency while largely preserving the shape of the failure-mode distribution, as discussed in Section 4.3.1.

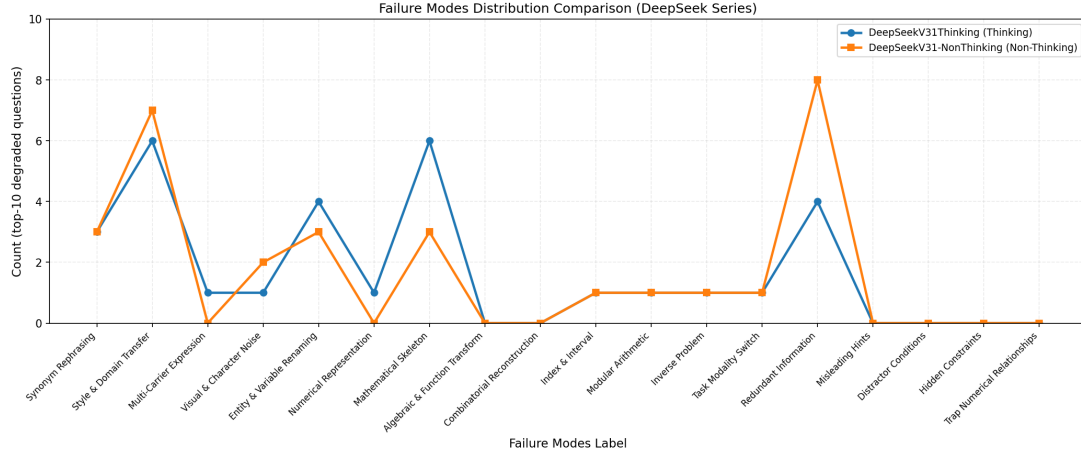


Figure 10: Failure mode distribution comparison of different reasoning level in Deepseek-V3.1-Terminus. **Takeaway:** stronger reasoning reduces the overall failure frequency while largely preserving the shape of the failure-mode distribution, as discussed in Section 4.3.1.

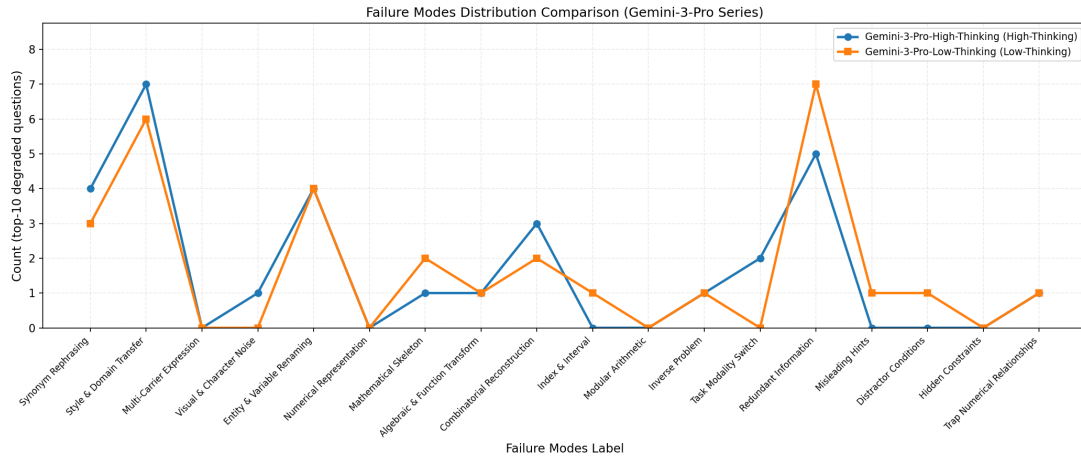


Figure 11: Failure mode distribution comparison of different reasoning level in Gemini-3-Pro. **Takeaway:** stronger reasoning reduces the overall failure frequency while largely preserving the shape of the failure-mode distribution, as discussed in Section 4.3.1.

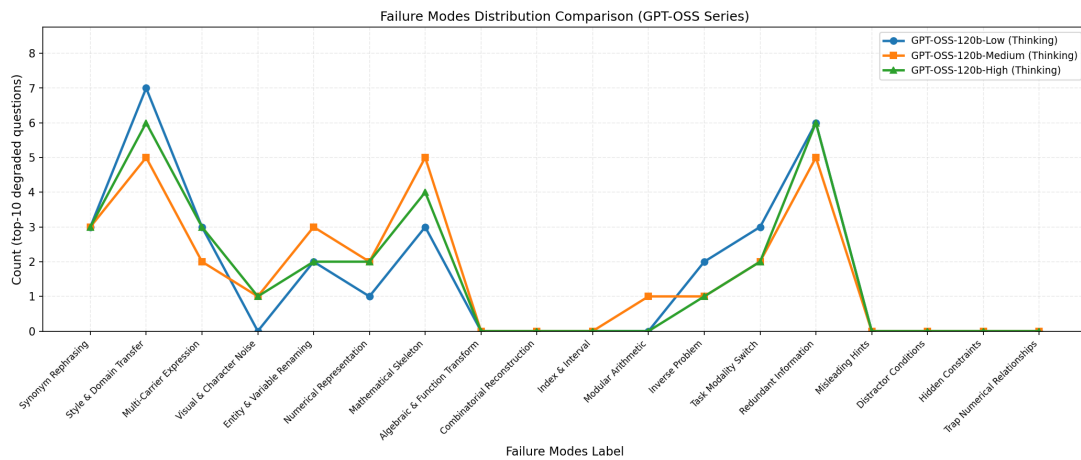


Figure 12: Failure mode distribution comparison of different reasoning level in GPT-OSS-120B. **Takeaway:** stronger reasoning reduces the overall failure frequency while largely preserving the shape of the failure-mode distribution, as discussed in Section 4.3.1.

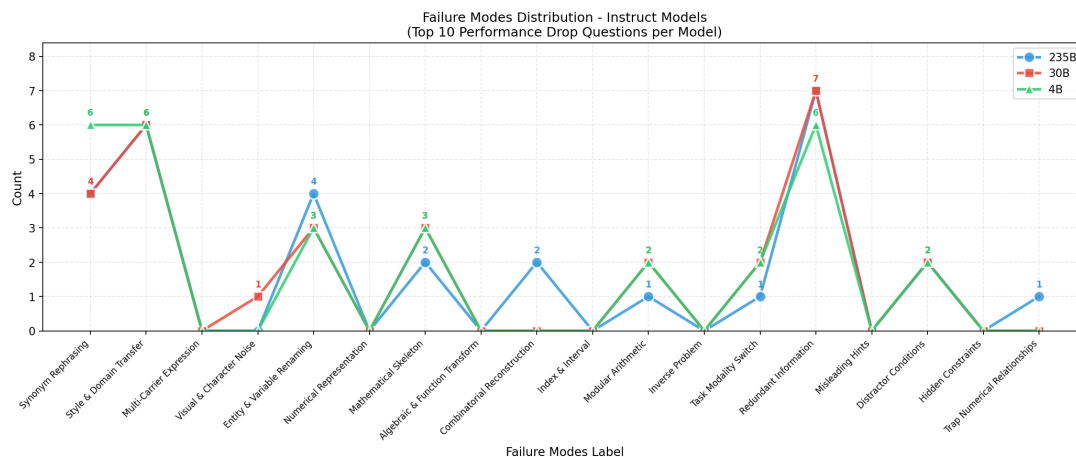


Figure 13: Failure mode distribution comparison of different model size in Qwen Instruct family. **Takeaway:** increasing model size improves absolute performance while similar weakness profiles persist across sizes, as discussed in Section 4.3.

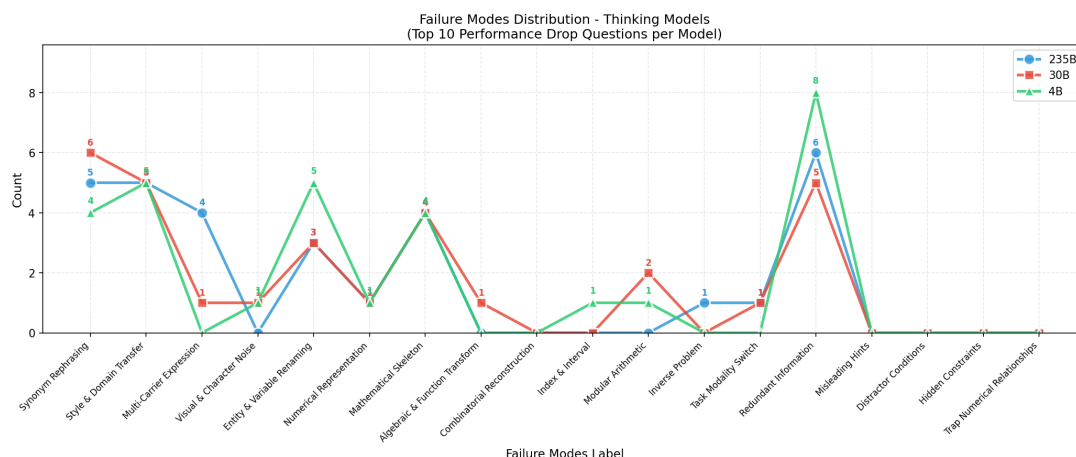


Figure 14: Failure mode distribution comparison of different model size in Qwen Thinking family. **Takeaway:** increasing model size improves absolute performance while similar weakness profiles persist across sizes, as discussed in Section 4.3.

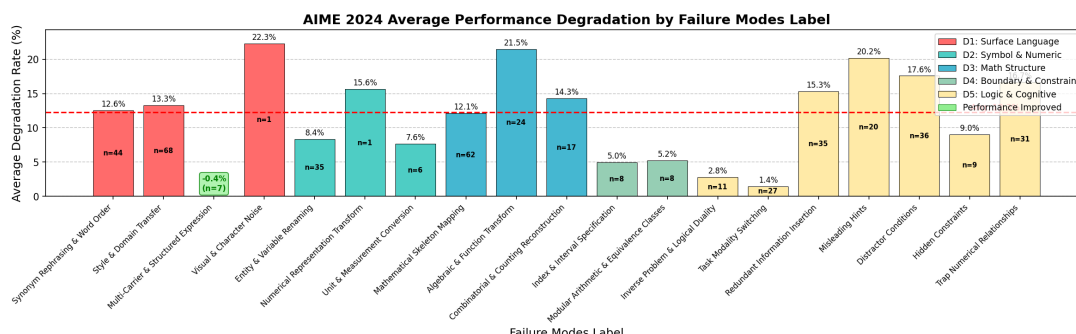


Figure 15: Bars report the average OG→RW relative drop in Avg@k for each AIME 2024 rewriting strategy, aggregated over problems labeled with that strategy (higher bars indicate larger average degradation). Bar annotations show the number of labeled instances (n); colors indicate the strategy family; the dashed line marks the overall average across strategies that degrade performance on average, and strategies that improve performance on average are indicated with a green callout and arrow.

Model	OG Avg@k (%)	RW Avg@k (%)	Δ Avg@k (pp)	Rel Δ Avg@k (%)	OG Pass@k (%)	RW Pass@k (%)	Δ Pass@k (pp)	Rel Δ Pass@k (%)	OG Len (tok)	RW Len (tok)
Open-source instruct model										
Llama-3.1-8B-Instruct (2024)	4.88	2.75	-2.13	-43.65	25.33	14.67	-10.66	-42.08	7246.5	9326.0
Qwen3-4B-Instruct-2507 (2025)	61.21	40.38	-20.83	-34.03	83.33	74.00	-9.33	-11.20	7079.9	8073.8
Qwen3-30B-A3B-Instruct-2507 (2025)	73.79	57.38	-16.41	-22.24	90.00	79.33	-10.67	-11.86	6169.2	7574.0
Qwen3-235B-A22B-Instruct-2507 (2025)	81.12	68.92	-12.20	-15.04	94.67	88.00	-6.67	-7.05	6493.2	7571.5
Qwen2.5-14B-Instruct (2024)	13.54	12.21	-1.33	-9.82	36.00	28.00	-8.00	-22.22	1145.4	1183.5
Qwen2.5-32B-Instruct (2024)	16.46	16.08	-0.38	-2.31	39.33	36.00	-3.33	-8.47	962.3	1038.5
Open-source reasoning model										
DeepSeek-R1-Distill-Llama-8B (2025a)	48.50	31.79	-16.71	-34.45	82.00	68.67	-13.33	-16.26	15133.9	15602.5
DeepSeek-R1-Distill-Qwen-14B (2025a)	68.96	49.08	-19.88	-28.83	86.67	80.00	-6.67	-7.70	11397.1	13452.0
DeepSeek-V3.1-Terminus (Thinking) (2024)	93.54	70.33	-23.21	-24.81	100.00	93.33	-6.67	-6.67	14106.6	20530.4
QwQ-32B (2025)	78.42	59.33	-19.09	-24.34	92.67	80.67	-12.00	-12.95	13943.6	16142.7
Qwen3-8B (2025)	75.62	57.21	-18.41	-24.35	89.33	79.33	-10.00	-11.19	15979.5	18410.4
DeepSeek-R1-Distill-Qwen-32B (2025a)	70.96	53.96	-17.00	-23.96	88.67	80.00	-8.67	-9.78	10931.2	12913.2
Qwen3-4B-Thinking-2507 (2025)	83.75	64.42	-19.33	-23.08	92.67	84.67	-8.00	-8.63	20162.9	23590.5
Qwen3-30B-A3B-Thinking-2507 (2025)	90.21	70.83	-19.38	-21.48	94.00	88.67	-5.33	-5.67	16299.8	20690.7
Qwen3-14B (2025)	79.50	62.54	-16.96	-21.33	93.33	82.00	-11.33	-12.14	13911.2	15643.4
Qwen3-32B (2025)	80.79	64.38	-16.41	-20.31	93.33	86.67	-6.66	-7.14	13539.6	16034.7
gpt-oss-120b (High) (2025)	95.62	81.75	-13.87	-14.51	100.00	98.00	-2.00	-2.00	16204.0	18614.2
Qwen3-235B-A22B-Thinking-2507 (2025)	94.17	81.50	-12.67	-13.45	96.67	95.33	-1.34	-1.39	19306.9	25284.4
DeepSeek-V3.2 (Thinking) (2025b)	94.58	82.62	-11.96	-12.65	96.67	94.67	-2.00	-2.07	12443.5	18614.9
Kimi-K2-Thinking (2025)	94.58	84.33	-10.25	-10.84	96.67	98.00	+1.33	+1.38	23150.4	28231.2
DeepSeek-V3.2-Special (Thinking) (2025b)	96.46	89.12	-7.34	-7.61	96.67	98.00	+1.33	+1.38	18315.0	24950.8
Proprietary reasoning model										
GPT-5.2 (High) (2025)	94.58	82.75	-11.83	-12.51	96.67	94.00	-2.67	-2.76	13017.46	18589.65
Gemini-3-Pro (High) (2025)	90.83	82.00	-8.83	-9.72	96.67	94.00	-2.67	-2.76	13159.14	14606.43

Table 9: AIME 2024 performance on Original (OG) vs. benchmark-adjacent rewritten variants (RW), sorted by drop in Avg@k. We report Avg@k and Pass@k, as well as absolute (Δ , in pp) and relative (Rel Δ , in %) changes from OG to RW, and response length (Len, in tokens). See Section 4.1 for metric definitions and the evaluation protocol.

Model	Type	OG Avg	RW Avg	Rel. Drop (%)↓
DeepSeek-V3.1-Terminus				
DeepSeek-V3.1	non_thinking	66.04	53.46	19.05
DeepSeek-V3.1-Thinking	thinking	93.54	70.33	24.81
DeepSeek-V3.2				
DeepSeek-V3.2	non_thinking	74.17	54.00	27.19
DeepSeek-V3.2-Thinking	thinking	94.58	82.62	12.65
GPT-OSS-120b				
GPT-OSS-120b-Low	low_thinking	55.00	41.21	25.07
GPT-OSS-120b-Medium	medium_thinking	78.33	59.54	23.99
GPT-OSS-120b-High	high_thinking	95.62	81.75	14.51
GPT-5.2				
GPT-5.2-Low	low_thinking	86.67	75.75	12.59
GPT-5.2-Medium	medium_thinking	94.58	82.75	12.51
GPT-5.2-High	high_thinking	98.31	90.32	8.13
Gemini-3-Pro				
Gemini-3-Pro-Low	low_thinking	90.83	82.00	9.71
Gemini-3-Pro-High	high_thinking	97.08	91.77	5.47

Table 10: OG vs. RW Avg@k and OG→RW relative drop for different reasoning settings. JS (adj.) reports the JS divergence between failure-mode distributions of adjacent thinking levels within each model; rows are ordered by increasing thinking intensity.

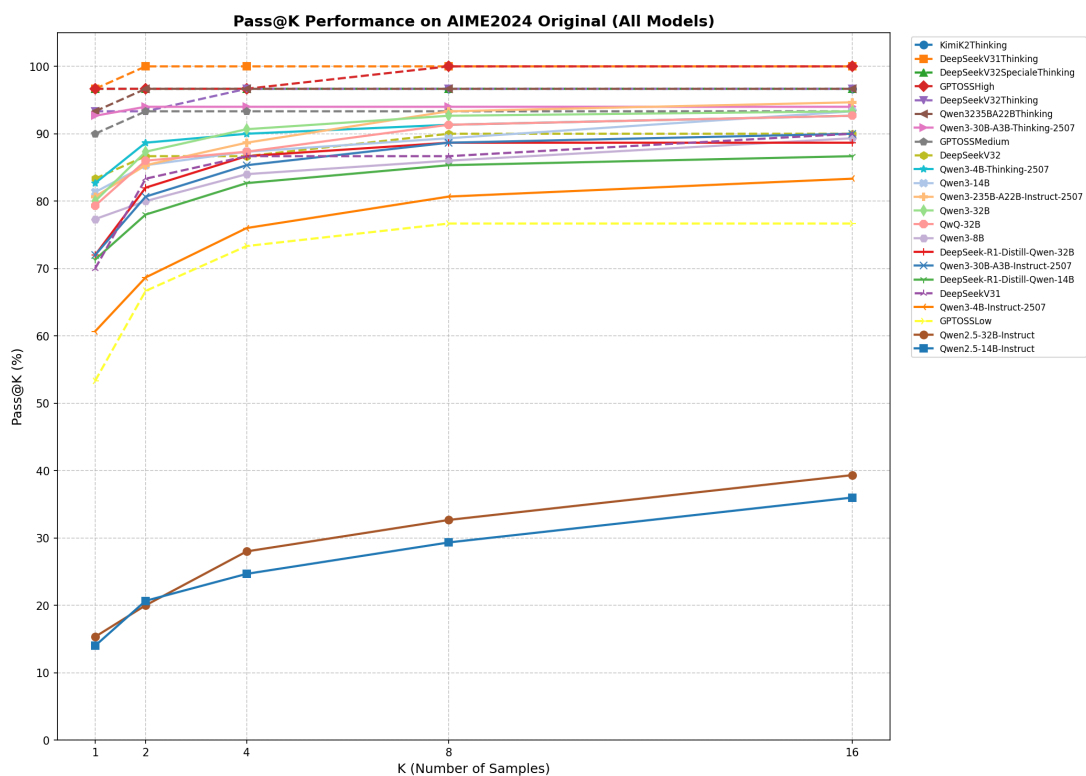


Figure 16: Full Pass@k performance in AIME 2024 OG.

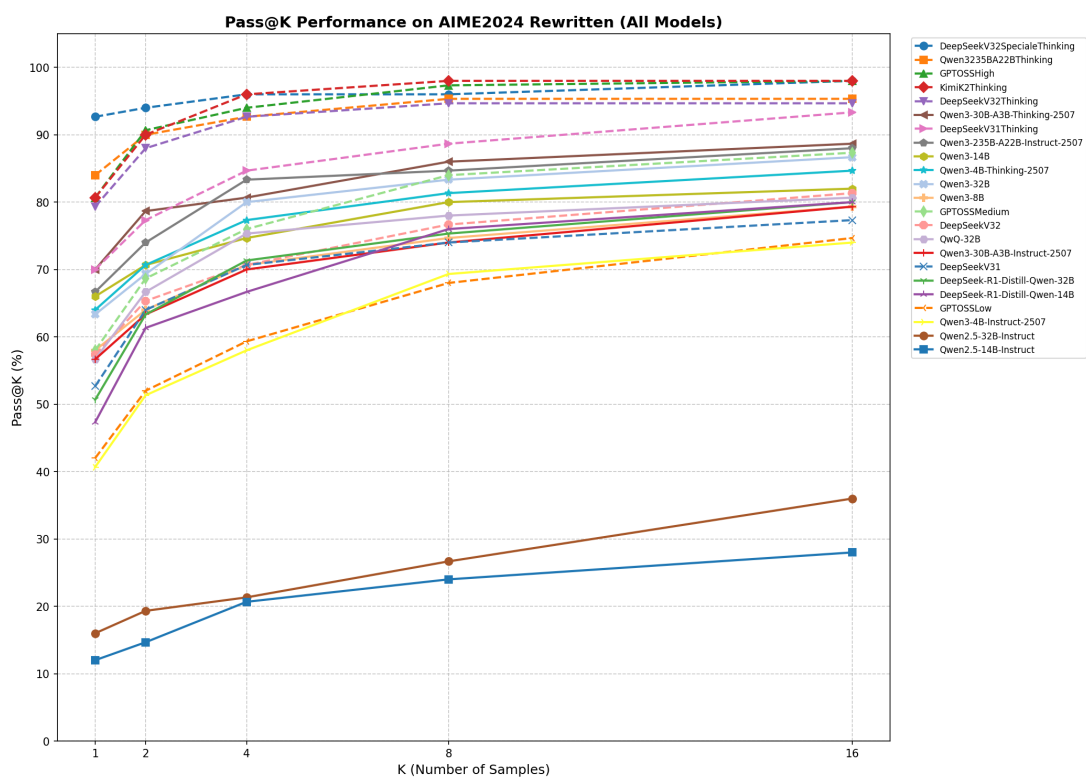


Figure 17: Full Pass@k performance in AIME 2024 RW.

Reinforcement Learning (e.g., RLVR)—to explicitly penalize semantic instability and force models to generalize rather than merely memorize. Concretely, AWE can support two complementary training pathways: (1) using the verified OG–RW pairs as targeted invariance data for SFT, RLVR, or other post-training objectives that penalize semantic sensitivity; and (2) using the discovered failure-mode taxonomy (Section 4.2) to guide targeted data collection and synthesis for continued pre-training or subsequent model generations, motivated by our observation that several high-impact failure modes appear underrepresented in current training data. Empirically validating these AWE-guided training recipes is an important direction for future work.

G.4 Model Choice in the AWE Pipeline

A natural concern is whether the specific models used in the AWE pipeline introduce systematic bias. First, the Solver ensemble (DeepSeek-V3.1-Terminus, gpt-oss-120b, Qwen3-235B-A22B-Thinking) partially overlaps with the evaluated model families, potentially biasing difficulty filtering toward weaknesses these solvers can detect. Second, employing DeepSeek as the Rewriter may shape the distribution of discovered failure modes toward patterns that DeepSeek-family models are predisposed to generate. We acknowledge these as limitations of resource-constrained construction: the AWE framework is model-agnostic by design, and the pipeline models can be replaced with stronger or more diverse alternatives as they become available. We expect that substituting more capable or heterogeneous models into the Rewriter and Solver roles would yield a broader and higher-quality set of failure-inducing variants, further strengthening JAGGEDBENCH’s diagnostic power.

H Qualitative Case Study

This section provides detailed case studies demonstrating how AWE discovers model weaknesses through automated problem rewriting. Each case shows the complete original problem, the rewritten variant, and analysis of why specific models fail on the rewritten version.

CASE STUDY: THE "GRAPH THEORY" ABSTRACTION TRAP

Targeting Geometric Probability Heuristics via Domain Transfer

Complete Problem Statements

Original Seed Problem (Geometry)

Alex divides a disk into four quadrants with two perpendicular diameters intersecting at the center of the disk. He draws 25 more line segments through the disk, drawing each segment by selecting two points at random on the perimeter of the disk in **different quadrants** and connecting those two points. Find the expected number of regions into which these 27 line segments divide the disk.

Answer: 204

AWE Rewritten Variant (Graph Theory)

Consider a circular disk with a **planar graph** constructed as follows: Begin with two perpendicular diameters intersecting at the center, dividing the disk into four congruent sectors. These diameters form the initial **edges** of the graph, with their intersection point and the four boundary points as **vertices**.

Now, add 25 additional edges to this graph. Each new edge is constructed by selecting two distinct points uniformly at random on the circular boundary, with the constraint that the two points lie in **different sectors** determined by the diameters. Each such edge is drawn as the chord connecting these points.

After all edges are added, the graph divides the disk into several connected regions (**faces**). Determine the expected number of faces in this planar graph.

Answer: 204 (same as original)

Key Transformation Analysis

Element	Original	Rewritten
Terminology	"line segments", "regions"	"edges", "faces", "planar graph"
Constraint	"different quadrants"	"different sectors determined by diameters"
Structure	Single-step description	Multi-stage construction
Domain	Geometric probability	Graph theory + probability

1. Phase 1: Rewriter's Attack Synthesis (Experience-Guided)

The Rewriter synthesized the attack by combining strategies, guided by the experience pool (referencing Steps 2–7):

- **Step 2 (Decomposition):** Identified the "Standard Random Chord" heuristic ($p = 1/3$) as the primary attack surface. Noted that the "quadrant constraint" is the key differentiator from standard problems.
- **Step 3 (Brainstorming):** Generated candidate strategies:
 - *Idea 1 (Selected): Graph Theory Reformulation.* Map geometric entities to abstract graph terms (*Line* → *Edge*, *Region* → *Face*) to force domain transfer.

- *Idea 2 (Selected): Probability Obscuration.* Describe the quadrant constraint implicitly using "sectors determined by diameters" to hide the symmetry breaking.
- *Idea 3 (Selected): Multi-Step Construction.* Break the construction into stages to increase cognitive load.

- **Step 4 (Strategy Selection):** The chosen strategy aims to trigger a "Heuristic Collapse". By masking the geometry, the Rewriter predicts models will fail to derive the specific intersection probability ($p = 17/36$) and revert to the standard random chord prior ($p = 1/3$).
- **Step 7 (Self-Correction):** Refined the "sector" terminology to ensure it maps bijectively to "quadrants", mitigating potential ambiguity while preserving difficulty.

2. Phase 2: Verification Loop (Solver & Judger)

The internal verification process confirmed both the difficulty trap and semantic equivalence:

- **Judger Consensus:** Score [1.0, 1.0, 1.0]. Judgers verified that "sectors determined by perpendicular diameters" is mathematically isomorphic to "quadrants", ensuring strict semantic equivalence.
- **Solver's Difficulty Filter (The "1/3 vs 17/36" Trap):** Internal solvers revealed the core trap. Weak reasoning paths defaulted to the standard heuristic $p = 1/3$ (ignoring constraints), while robust paths derived $p = 17/36$ via case analysis:
 - *Standard Heuristic* ($P = 1/3$): Random chords without sector constraints → $P = 1/3$.
 - *Required Derivation* ($P = 17/36$): "Different sector" constraint breaks symmetry. Aggregating weighted probabilities: **Same Adj** (4 cases, $p = 1/2$); **Same Opp** (2 cases, $p = 1/2$); **Disjoint/Sharing** (30 cases). Total weighted sum $2 + 1 + 14 = 17 \Rightarrow P = 17/36$.

3. Phase 3: Summarizer's Reflection

"The strategy succeeded via three mechanisms: (1) **Graph Abstraction** disrupted pattern recognition; (2) **Multi-stage Construction** increased cognitive load; (3) **Probability Obscuration** acted as the critical trap, hiding the non-standard 17/36 probability behind 'sector' terminology."

4. Phase 4: JaggedBench Evaluation (Post-Generation)

When deployed on the final benchmark, the problem devastated heuristic-based models:

- **[Failure] GPT-5.2-Low (Succumbed to Trap):**

"...Mapping back to geometry, this is equivalent to random chords. Recall that the intersection probability for random chords is 1/3. Using Euler's formula... we get $E \approx 129$."

Analysis: The model recognized the geometry but defaulted to the training prior (1/3), missing the constraint.

- **[Success] Gemini-3-Pro (Rigorous Reasoning):**

"...Vertices must lie in different sectors. Breaks symmetry. Case analysis: Same Adj ($4 \times 1/2$), Same Opp ($2 \times 1/2$), Disjoint Opp (2×1)... Summing weights gives 17/36. Final Answer: 204."

Case Study Summary: *This example demonstrates how AWE automatically discovers a "blind spot" in model reasoning. While models are proficient at standard geometric probability ($p = 1/3$), they fail to generalize to a topologically equivalent but linguistically abstract formulation ($p = 17/36$). The framework successfully exploits this by retrieving a "Probability Obscuration" strategy, verifying the semantic equivalence via consensus, and crystallizing the failure mode into actionable experience for future iterations.*

I Extended Analysis on AWE Designs

This section provides the detailed experimental setup, data, and analysis for the design choices and validation checks of the AWE framework.

I.1 Ablation: Experience Mechanism

We ablate the **experience mechanism** to study whether accumulated experience enables AWE to progressively focus its search toward more productive regions of the rewrite strategy space $\mathcal{T}$, while avoiding repeatedly making the same unproductive mistakes and wasting tokens.

Table 11 shows that enabling the experience mechanism increases the success rate of validated and failure-inducing rewrites from 8.7% to 12.0% (52→72) under the same trial budget, while halving equivalence check failures (30→15). This suggests that experience guidance steers rewriting toward more productive strategies in $\mathcal{T}$, reducing wasted trials and improving overall search efficiency.

To further isolate what experience contributes, we compare construction iterations in which the experience pool is provided against iterations in which it is randomly dropped (experience dropout). We report the *yield* (percentage of iterations producing at least one accepted rewrite) and the *mean Solver accuracy* on accepted rewrites (lower indicates stronger stress). As shown in Table 12, experience raises yield by 5.10 pp while leaving the Solver accuracy of accepted rewrites nearly unchanged, indicating that experience primarily improves *search efficiency* rather than increasing the intrinsic difficulty of retained variants.

I.2 Judger Number and Cross-Family Diversity

The pipeline uses three Judges from distinct model families—DeepSeek, Qwen, and GPT-OSS—and accepts a candidate only when all three confirm

equivalence. To quantify the value of this cross-family unanimity, we counterfactually replay each individual Judger’s recorded vote as the *sole* acceptance criterion and count how many candidates would pass equivalence verification under each. As shown in Table 13, any single Judger would admit 261–513 additional candidates that are questioned by at least one other family. Requiring three-family agreement is therefore a more conservative safeguard against semantic-equivalence errors than relying on any single Judger.

We additionally analyze the Solver-threshold τ , which controls a smooth coverage–difficulty trade-off in the retained pool; the full sensitivity sweep is reported in Appendix I.9 (Table 23). Fully replacing the Rewriter or introducing new Judger families would require rerunning and re-auditing the entire construction pipeline; we therefore leave these higher-cost studies to future versioned releases, consistent with the configuration-specific limitation acknowledged in Appendix G. Overall, these analyses clarify the role of each component: experience improves exploration efficiency, the Solver threshold controls the coverage–difficulty trade-off, and multiple cross-family Judges provide a conservative semantic-verification safeguard.

I.3 Synthesis Cost Analysis

Across the 600 rewrite trials reported in Table 11, the end-to-end pipeline consumed ~270M tokens across three models and cost **219.81 USD** in total. It produced 72 retained and verified variants, which yields a synthesis cost of **3.05 USD** per usable instance. This cost profile suggests that AWE is a highly cost-effective solution for systematically exploring model weaknesses and for synthesizing failure-inducing and verified data at scale. Detailed breakdowns and per-token unit prices are provided in Table 14 and 15.

I.4 Comparison with Human-Curated Stress Tests

Table 16 compares AWE-generated variants with two widely used human-curated rewrite paradigms—an LLM-based rewriting baseline following human-specified strategies (ThinkBench-style (Huang et al., 2025)) and a rule-based perturbation baseline (Sclar et al., 2023)—on the same AIME 2024 subset. Across all evaluated models, AWE induces a substantial and consistent performance drop (mean relative drop of 24.4%), whereas human-designed rewrite sets lead to only marginal

Metric	With Experience	Without Experience
Total Rewrite Trials	600	600
Success Rate (# Valid Attacks / Trials)	12.0% (72/600)	8.7% (52/600)
Equivalence Check Failures (#)	15	30

Table 11: **Ablation on the experience mechanism.** Experience-guided rewriting yields more *valid* adversarial rewrites and substantially reduces equivalence check failures.

Condition	Yield	Mean Solver Acc.
Experience enabled	32.58%	0.2549
Random experience dropout	27.48%	0.2584

Table 12: **Effect of experience availability.** Experience increases yield while the difficulty of accepted rewrites stays nearly constant, showing that its main benefit is exploration efficiency.

Equivalence criterion	Judger pass count
DeepSeek only	4,438
Qwen only	4,259
GPT-OSS only	4,186
Three-family unanimity (ours)	3,925

Table 13: **Judger cross-family diversity.** Counterfactual pass counts when each single Judger is used as the sole acceptance criterion, versus the three-family unanimity rule adopted by AWE. Unanimity rejects 261–513 candidates that a single Judger would have admitted.

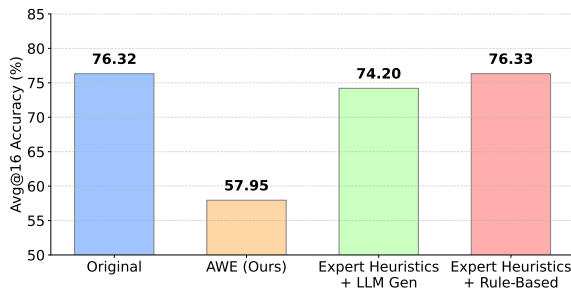


Figure 18: Comparison of AWE with human-designed strategies on AIME 2024 (Avg@16). While human-designed strategies (implemented via LLMs or rules) largely preserve original performance levels, AWE automatically discovers semantically equivalent variants that significantly stress-test the model. Detailed numerical results are provided in Table 16.

degradation (2.8%) or effectively no drop at all. The results suggest that many existing rewrite strategies have largely saturated and no longer provide a meaningful stress test for frontier LLMs. By contrast, allowing models to autonomously propose and diversify rewrite strategies enables the discovery of higher-impact, capability-adjacent failures that are not emphasized by fixed human priors, highlighting the necessity of automated weakness exploration for robust evaluation of modern LLMs.

Comparison protocol: matched-seed evaluation.

For all baseline comparisons in this subsection, we follow a *matched-seed* protocol rather than comparing leaderboard scores across independently constructed benchmarks, which would be confounded by different seed problems, task distributions, and evaluation settings. Concretely, we apply each rewriting mechanism to the *same* original problems and measure the induced degradation under an identical evaluation setting. Table 16 instantiates this protocol against the two fixed-strategy paradigms above; we next extend it to a fixed perturbation suite.

Comparison with fixed perturbation suites (RUPBench).

We further compare AWE with RUPBench (Wang and Zhao, 2024), a fixed perturbation suite covering lexical, syntactic, and semantic perturbations across nine predefined transformation types. Following the matched-seed protocol, we apply both mechanisms to the *same* MMLU-Pro seed questions and evaluate the resulting variants under identical model and metric settings (Avg@8). All decreases are reported as absolute Avg@8 reductions (pp) relative to the shared matched-original score of 43.75%.

As shown in Table 17, on the shared MMLU-Pro seeds AWE variants induce a 9.05 pp absolute drop, substantially larger than any RUPBench perturbation family (0.54–2.41 pp) and their combination (1.29 pp). This indicates that fixed, human-specified perturbation spaces largely fail to stress current models, whereas AWE’s feedback-

Model	Input Tokens	Output Tokens	Total Tokens	Total Cost (USD)
DeepSeek-V3.1	9,622,077	79,619,072	89,241,149	36.13
Qwen3-235B-A22B	6,299,585	98,630,871	104,930,456	140.29
GPT-OSS-High	6,299,585	70,169,224	76,468,809	43.39
Total	–	–	270,640,414	219.81

Table 14: **Token usage and synthesis cost (600 trials).** Total cost sums over model-specific input/output pricing.

Model	Input (\$/1M tok)	Output (\$/1M tok)
DeepSeek-V3.1	0.28	0.42
Qwen3-235B-A22B	0.35	1.40
GPT-OSS-High	0.15	0.60

Table 15: Pricing of different models (converted to USD at 1 RMB = 0.14 USD).

driven strategy search exposes markedly stronger capability-adjacent weaknesses. Finally, we do not perform a direct numerical comparison with AutoBench (Li et al., 2025): it searches for new benchmark datasets satisfying benchmark-level objectives, whereas AWE searches the semantics-preserving neighborhood of existing seeds and produces paired OG–RW probes, so their generated objects and evaluation units are fundamentally different (see Table 22).

I.5 Ranking Consistency and Benchmark Validity

A potential concern with rewritten benchmarks is whether they introduce bias that unfairly penalizes specific model families. To address this, we analyze the Spearman rank correlation between model performance on the original seed problems and our rewritten variants. As illustrated in Figure 19, the model rankings remain remarkably stable across the transition from standard seed problems to our adversarially rewritten variants. The strongest models on the original benchmarks (e.g., Qwen3-235B-Thinking, DeepSeek-V3.1) retain their top positions on JAGGEDBENCH, despite the universal performance drop.

This **rank preservation** does not prove the absence of bias, but it provides evidence against a gross family-specific artifact:

1. **Broad Difficulty Increase:** The generated variants substantially increase difficulty across many model families, rather than only targeting a single construction-time model.
2. **Capability-Order Preservation:** JAGGEDBENCH largely preserves the capability order-

ing induced by the seed benchmarks while moving evaluation into a more challenging, benchmark-adjacent region of the input space.

I.6 Ruling Out Superficial Confounds: Length and Distractor Analyses

A natural concern is whether the AWE rewrite pipeline inadvertently alters the intrinsic *difficulty* of a problem rather than exposing genuine jaggedness. In particular, distractor-oriented rewrites could intuitively conflate the overall problem complexity, causing the OG→RW gap to reflect an increase in task difficulty rather than a reliability failure near the capability boundary.

AWE preserves the underlying task. By construction, AWE preserves the *underlying mathematical task*: the target, the operative constraints, the final answer, the core mathematical relationships, and the intended solution path all remain unchanged. Distractor-family rewrites add only irrelevant information, or facts already implied by the original conditions; they do not introduce new operative constraints or additional mathematical subproblems. This is enforced through the Rewriter’s explicit equivalence constraints and self-solving (Section 3.1), unanimous cross-family Judger verification (Section 3.1), proof-based checks, and per-instance human review for AIME (Section 3.3 and Appendix B). Beyond this qualitative argument, we conduct two complementary quantitative analyses to rule out superficial confounds: (1) controlling for rewrite length, and (2) comparing distractor and non-distractor rewrites.

Effect of rewrite length. For each of the 150 AIME 2025 OG–RW pairs, we compute the RW

Model	AWE (ours)			Human-designed strategy & LLM generate variants (Huang et al., 2025)			Human-designed strategy & rule-based generate variants (Sclar et al., 2023)		
	OG	RW	Rel. Drop	OG	RW	Rel. Drop	OG	RW	Rel. Drop
Qwen3-8B	75.62	57.21	-24.3%	75.62	70.54	-6.7%	75.62	75.66	+0.1%
Qwen3-14B	79.50	62.54	-21.3%	79.50	79.80	+0.4%	79.50	79.49	-0.0%
Qwen3-32B	80.79	64.38	-20.3%	80.79	80.69	-0.1%	80.79	80.85	+0.1%
Qwen3-4B-Thinking-2507	83.75	64.42	-23.1%	83.75	82.92	-1.0%	83.75	83.80	+0.1%
Qwen3-30B-A3B-Thinking-2507	90.21	70.83	-21.5%	90.21	85.21	-5.5%	90.21	90.24	+0.0%
Qwen3-4B-Instruct-2507	61.21	40.38	-34.0%	61.21	59.77	-2.4%	61.21	61.19	-0.0%
Qwen3-30B-A3B-Instruct-2507	73.79	57.38	-22.2%	73.79	70.48	-4.5%	73.79	73.75	-0.1%
DeepSeek-R1-Distill-Qwen-32B	70.96	53.96	-24.0%	70.96	69.08	-2.6%	70.96	70.98	+0.0%
DeepSeek-R1-Distill-Qwen-14B	68.96	49.08	-28.8%	68.96	67.69	-1.8%	68.96	68.93	-0.0%
QwQ-32B	78.42	59.33	-24.3%	78.42	75.84	-3.3%	78.42	78.37	-0.1%
AVG	76.32	57.95	-24.1%	76.32	74.20	-2.8%	76.32	76.33	0.0%
Mean Relative Drop		-24.4%			-2.8%			0.0%	

Table 16: Automated (AWE) vs. human-curated rewrite variants on the same AIME 2024 subset (Avg@16). Larger drops indicate a stronger stress test that exposes capability-adjacent reasoning failures.

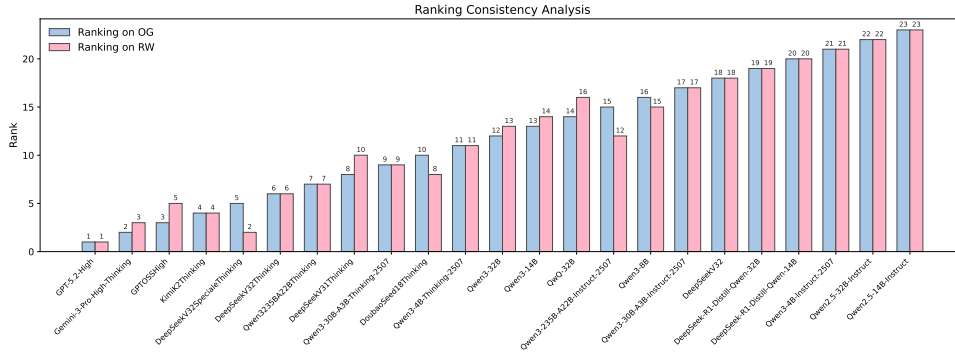


Figure 19: **Ranking Consistency Analysis.** We compare model rankings on the Original (OG) vs. Rewritten (RW) datasets. The rankings remain highly consistent (Kendall’s $\tau = 0.91$, Spearman’s $\rho = 0.98$), suggesting that JAGGEDBENCH largely preserves the relative capability hierarchy of models while exposing additional benchmark-adjacent reasoning failures.

Variant set	Avg@8	Decrease (pp)
Matched original (OG)	43.75%	—
RUPBench lexical perturbations	41.34%	2.41
RUPBench syntactic perturbations	42.84%	0.91
RUPBench semantic perturbations	43.21%	0.54
RUPBench all nine perturbations	42.46%	1.29
AWE variants	34.70%	9.05

Table 17: **Matched-seed comparison with RUPBench on shared MMLU-Pro seeds (Avg@8).** “Decrease” is the absolute Avg@8 reduction (pp) relative to the shared matched-original score of 43.75%. AWE-selected variants induce a substantially larger degradation than all fixed RUPBench perturbation families.

token length (number of tokens in the rewritten problem under the Qwen3-4B tokenizer) and the *drop* (RW minus OG accuracy, averaged across 10 evaluated models with 16 samples each; more negative values indicate larger degradation), and measure their correlation. As shown in Table 18, rewrite length explains only about 1% of the variance in performance degradation and is not statistically significant, indicating that the observed OG→RW drop is not primarily caused by longer rewritten inputs.

Statistic	Value
Pearson correlation	$r = -0.127, p = 0.120$
Spearman correlation	$\rho = -0.093, p = 0.257$
OLS: drop $\sim$ RW token length	$\beta = -0.044 \text{ pp/token}, p = 0.120$
Adjusted R^2	0.010

Table 18: **Correlation between rewrite length and performance drop** on the 150 AIME 2025 OG–RW pairs. Rewrite length is not a significant predictor of degradation.

We further fit nested regressions to test whether the rewriting-strategy content provides explanatory power beyond length (Table 19). Adding the

rewrite-strategy labels raises the adjusted R^2 from 0.010 to 0.204; the partial R^2 is 0.064 for length versus 0.267 for strategy labels, indicating that the rewriting strategy itself—rather than surface length or distractor presence—is the dominant factor associated with degradation.

Model	Predictors	Adj. R^2	Model sig.
M1	RW token length	0.010	$p = 0.120$
M2	+ distractor indicator	0.004	$p = 0.263$
M3	+ rewrite strategy labels	0.204	$p = 1.4 \times 10^{-5}$

Table 19: **Nested regressions for OG→RW drop.** Rewrite-strategy labels explain substantially more variance than length or a distractor indicator.

Distractor-removal analysis. To empirically test whether distractors alone explain the observed degradation, we conduct a distractor-removal analysis on all 150 independently generated and evaluated AIME 2025 OG–RW pairs. A pair is classified as *distractor-related* when its top-three strategy labels include **Redundant Information Insertion**, **Misleading Hints**, or **Distractor Conditions** (fine-grained labels within the *Distractor & Complexity Injection* strategy family; see Table 8). We then split the pairs into distractor-family and non-distractor groups and compare the mean OG→RW decrease. Note that here we report the *absolute* decrease in accuracy (in percentage points, pp), which is distinct from the relative drop reported in the main text.

Group	Mean OG→RW decrease (pp)	95% CI
Distractor-family	25.69	[21.41, 29.96]
Non-distractor	20.23	[14.36, 26.10]
Difference	5.46	—

Table 20: **Distractor-removal analysis on the 150 AIME 2025 OG–RW pairs.** Distractor-family variants show a numerically larger absolute decrease (pp), but the between-group difference is not statistically significant, and non-distractor variants alone still produce a substantial degradation.

Distractors are one failure mode, not the whole story. As shown in Table 20, although distractor-family variants exhibit a numerically larger decrease, the between-group difference is *not* statistically significant (Welch’s t -test $p = 0.136$; Mann–Whitney U test $p = 0.122$). More importantly, after excluding *all* distractor-family pairs, the remaining non-distractor variants still produce

a substantial **20.23 pp** decrease, with a 95% confidence interval that excludes zero. Thus, distractors constitute one high-impact failure mode, but they do not explain the observed jaggedness as a whole.

Identifying non-operative information is also deployment-relevant: as discussed in Section 4.2 and Appendix G, real user inputs are often informal, redundant, or contain misleading cues. This observation is consistent with GSM-NoOp (Mirzadeh et al., 2024), which adds seemingly relevant but operationally inconsequential statements without changing the reasoning required to reach the answer.

1.7 Out-of-Pipeline Validity: Held-out Model Evaluation under Unified K

A further concern is potential *circularity bias*: because AWE uses a fixed set of construction models (Rewriter, Judges, and Solvers), the discovered weaknesses might be specific to those models or their families, rather than reflecting genuine, transferable reasoning failures. We address this concern from two complementary angles.

AWE is model-agnostic. The current JAGGED-BENCH instantiation uses a fixed set of construction models and may therefore inherit configuration-specific bias, as discussed in Appendix G. It is important to distinguish two notions here. By *model-agnostic* we mean that the Rewriter, Judges, and Solvers are *modular and replaceable* at the framework level; we do *not* claim that a particular benchmark release is independent of its construction configuration. The released JAGGED-BENCH is one concrete instantiation and naturally reflects its construction-model choices, whereas the AWE framework itself imposes no architectural requirement on those choices (Section 3.1 and Appendix G). Because AWE is modular, developers can replace its components—for example, adopting a stronger proprietary Rewriter and *including their own target model in the Solver ensemble*—to surface model-specific weaknesses and guide targeted data collection and future model improvement. The framework thus admits a clean separation between construction and evaluation model families whenever such resources are available.

Held-out model evaluation. To directly test transferability, we evaluate six models that were *entirely held out* from AWE construction (Section 4.2). Gemini, Doubao/Seed, and Kimi belong to model families that are *absent* from the

pipeline, while GPT-5.2/5.4 also never participated in rewriting, judging, or solver difficulty filtering. To remove any confound arising from differing sampling budgets, we additionally standardize all results to a unified $K = 16$. We note that the resulting values therefore differ from the main-text table, where proprietary models are reported at $K = 8$ (Table 2); the two are consistent in trend but computed under different sampling budgets.

Held-out model	OG Avg@16	RW Avg@16	Decrease (pp)	Rel. decrease
Gemini-3-Pro	93.12	82.92	10.21	10.96%
GPT-5.2	99.38	84.50	14.88	14.97%
Doubao-Seed-1.8-Thinking	86.04	67.04	19.00	22.08%
Gemini-3.1-Pro	97.42	88.38	9.04	9.28%
GPT-5.4	99.42	94.50	4.92	4.95%
Kimi-K2-Thinking	94.17	80.42	13.75	14.60%

Table 21: **Held-out model evaluation under unified $K = 16$ on AIME 2025.** All six models are entirely held out from the AWE construction pipeline; Gemini, Doubao/Seed, and Kimi additionally belong to families absent from the pipeline. Values are standardized to $K = 16$ and thus differ from the main-text table (proprietary models at $K = 8$).

Weaknesses are not confined to construction-time models. As shown in Table 21, all held-out models exhibit consistent OG→RW degradation, including frontier models with near-saturated OG performance (e.g., GPT-5.4 at 99.42% OG). This provides direct evidence that the discovered weaknesses are not confined to the construction-time models or families, arguing against a gross circularity artifact.

Human-in-the-loop safeguards. Finally, the main analysis benchmarks (AIME 2024/2025) already undergo per-instance human verification, with disjoint reviewer groups checking answer correctness/uniqueness and semantic equivalence, followed by group adjudication and correction or re-synthesis of problematic cases (Appendix B). Together, the model-agnostic design, the held-out evaluation, and the human-in-the-loop review jointly mitigate potential model bias arising from the construction pipeline.

Actionable deployment evaluation: distinguishing models hidden by benchmark saturation. The held-out evaluation also yields a concrete, actionable use case for pre-deployment auditing and model-version selection. Under the unified $K = 16$ setting (Table 21), GPT-5.2 and GPT-5.4 are *nearly indistinguishable* on the original benchmark (OG Avg@16 of 99.38% vs. 99.42%, a gap

of only 0.04 pp), yet they differ substantially under AWE stress testing (RW Avg@16 of 84.50% vs. 94.50%, a gap of 10.00 pp). The corresponding OG→RW decreases—14.88 pp for GPT-5.2 (BCa 95% CI [11.29, 19.17]) and 4.92 pp for GPT-5.4 (BCa 95% CI [2.92, 7.79])—are both statistically significant ($p < 10^{-4}$). Because neither model participated in any AWE construction role, this shows that AWE can expose robustness differences that are entirely masked by saturated original benchmarks, providing an actionable signal for model-version selection and pre-deployment auditing that standard scores cannot.

I.8 Detailed Comparison with Prior Evaluation Paradigms

We do not claim to be the first to observe that LLMs are sensitive to paraphrases, formatting changes, distractors, or semantics-preserving perturbations, nor the first to build automated or dynamic benchmarks. To sharpen the positioning summarized in Section 2, Table 22 contrasts AWE with representative prior paradigms along four dimensions: the underlying *assumption and search object*, the *generation mechanism*, the *validation criteria and empirical coverage*, and the *key distinction* from AWE. The central methodological difference is that AWE treats the rewrite *strategy itself* as the search object and optimizes it under joint equivalence and failure-induction constraints, then turns the discovered transformations into refreshable, verified OG→RW probes; prior paradigms instead operate within fixed human-specified or template-defined transformation spaces, or construct new tasks rather than paired, seed-equivalent weakness probes.

I.9 Selection-Induced Difficulty: Solver-Threshold Sensitivity

Because the Solver filter (Section 3.1) explicitly retains variants that induce solver failures, the observed OG→RW drop is *partly selected by construction*. JAGGEDBENCH should therefore be interpreted as a *targeted stress test* rather than an unbiased estimate over all possible semantics-preserving rewrites. To make this explicit, we separate two conceptually distinct stages: (1) *semantics-preserving candidate discovery*, which generates and verifies seed-equivalent rewrites; and (2) *stress-test selection*, which retains candidates satisfying $0 < \text{Acc} \leq \tau$. The main paper fixes $\tau = 0.7$; here we quantify the effect of this selection criterion.

We define the relevant quantities as follows. τ

Work / paradigm	Assumption & search object	Generation mechanism	Validation & empirical coverage	Key distinction from AWE
PromptRobust; Sclar et al. (Zhu et al., 2023; Sclar et al., 2023)	Robustness to changes in prompts or few-shot formatting	Predefined character/word/sentence/semantic attacks, or search within a user-defined grammar of equivalent formats	Semantic-integrity or grammar-based equivalence; broad NLP tasks and 53 classification tasks	Operate on prompt instructions/formats within predefined spaces, rather than discovering seed-problem rewrite strategies from Solver feedback
RUPBench; Lunardi et al. (Wang and Zhao, 2024; Lunardi et al., 2025)	Robustness of benchmark questions to linguistic perturbations	Nine predefined lexical/syntactic/semantic perturbations, or GPT-generated paraphrases under preservation constraints	Fifteen reasoning datasets and six multiple-choice benchmarks	Evaluate fixed perturbation or paraphrase families without iterative, outcome-grounded strategy updating
GSM-Symbolic (Mirzadeh et al., 2024)	Controlled instantiations of GSM8K-style symbolic templates	Numerical resampling, clause-count variation, and No-Op insertion	Template-derived answers; grade-school mathematics	Provides controlled but fixed, math-specific transformations rather than model-feedback-driven strategy discovery
Mathematically equivalent transformations (Hao et al., 2025)	Robustness of advanced mathematics under equivalence-preserving operators	Surface renaming and kernel/parametric transformations	Multi-round model self-review and human spot-checking; 1,051 Putnam problems and 6,306 items	Provides rigorous fixed transformation families, but does not refine the strategy space from Judger/Solver outcomes
AutoBench (Li et al., 2025)	Search over new dataset descriptions that optimize declared desiderata	Iterative description proposal using candidate-model feedback, privileged information, and final reranking	Grounding through retrieval/tools plus human correctness and salience checks; math, knowledge, multilinguality, and safety	Adaptively constructs new QA datasets; AWE instead preserves a fixed seed task and produces paired OG–RW probes
DyVal (Zhu et al., 2024)	Dynamic generation of new reasoning tasks with controllable complexity	DAG-based composition under task and complexity constraints	Programmatically determined ground truth; mathematics, logic, and algorithms	Generates new tasks of controlled complexity rather than searching seed-equivalent variants that expose active weaknesses
ThinkBench (Huang et al., 2025)	OOD variants of existing questions for reducing contamination effects	Scenario-level rephrasing plus three predefined attack types, with Rephrasing and Verifier agents	Multi-agent verification and sampled human evaluation; AIME and GPQA	Preserves seed content, but uses predefined generation modes without iterative strategy learning from Solver outcomes
CodeCrash (Lam et al., 2025)	Code-reasoning robustness under structural noise and misleading natural language	Predefined structural and textual-distraction perturbations	Source-benchmark oracles; CRUXEval and LiveCodeBench	A domain-specific fixed perturbation suite rather than autonomous, cross-domain weakness exploration
Robust Reasoning Benchmark (Golikov et al., 2026)	Mathematical robustness under reversible structural transformations	Deterministic textual, syntactic, and visual encodings	Transformations preserve values, logic, and answers by construction; AIME 2024/2025	Uses a fixed deterministic transformation inventory rather than model-adaptive strategy discovery and filtering
AWE (ours)	Semantics-preserving neighborhood of a fixed seed, targeting weaknesses active in current models	Autonomous seed-conditioned strategy proposal with iterative Judger/Solver feedback and experience updating	Unanimous equivalence judgment, solvability/failure filtering, and proof-based, objective, and human verification; math, code, science, and knowledge	Closed-loop exploration beyond fixed human priors and renewable evaluation as models evolve

Table 22: **Comparison of AWE with representative prior evaluation paradigms**, organized by assumption/search object, generation mechanism, validation/coverage, and key distinction. AWE’s central novelty is optimizing the transformation *strategy* under joint equivalence and failure-induction constraints, then producing refreshable, verified OG–RW probes.

is the maximum internal Solver accuracy allowed for retention (a lower value selects harder candidates). *Retained candidates* are the Judger-passing variants satisfying $0 < \text{Acc} \leq \tau$, counted before final post-processing and release selection. *Seed coverage* is the number of the 30 AIME 2025 seeds that retain at least one candidate. *Mean Solver accuracy* is the average fraction of correct internal Solver attempts over retained candidates, where lower values indicate stronger stress.

Solver threshold τ	Retained cand.	Seed cov.	Mean Solver Acc.
0.3	188	29/30	0.160
0.5	290	30/30	0.247
0.7 (ours)	442	30/30	0.373
0.9	778	30/30	0.563

Table 23: **Solver-threshold (τ) sensitivity on AIME 2025.** The retained pool size and mean Solver accuracy vary smoothly and monotonically with τ , exposing an explicit coverage–difficulty trade-off. We choose $\tau = 0.7$ as it maintains complete seed coverage while retaining a substantially stronger stress level than $\tau = 0.9$.

As shown in Table 23, both the retained pool size and the mean Solver accuracy vary smoothly and monotonically with τ , making the intended coverage–difficulty trade-off explicit. We choose $\tau = 0.7$ because it maintains complete seed coverage (30/30) while retaining a substantially stronger stress level than $\tau = 0.9$.

I.10 Statistical Significance and Reproducibility Details

To strengthen interpretability, we standardize the main cross-model comparison to a unified $K = 16$, add uncertainty estimates for the main results and scaling analyses, and report complete inference configurations. For a consistent budget, all five proprietary models are additionally evaluated with 16 responses per problem; their unified $K = 16$ scores are reported in Appendix I.7 (Table 21), and are used in the bootstrap analysis below.

Bootstrap significance for all 24 main models.

For all 24 models in the main AIME 2025 evaluation, we bootstrap the absolute Avg@16 drop (OG – RW, in pp) over the 150 OG–RW pairs. We report percentile 95% CIs (and BCa intervals where available) and a test against zero degradation. The five proprietary models use the unified $K = 16$ results; the other 19 use the existing $K = 16$ results. As shown in Table 24, the percentile 95% CI excludes zero for *all 24 models* ($p \leq 2 \times 10^{-4}$), so

the OG→RW degradation is statistically significant across the complete main evaluation set rather than being driven by a small subset of models.

Failure-mode significance. As an additional robustness check, we test whether the degradation persists across the 18 rewriting modes for DeepSeek-V3.2 (Thinking), a representative strong model evaluated with $K = 16$. Each pair may receive up to three mode labels, so the mode subsets may overlap; for each mode we run a paired t -test on the pair-level OG–RW accuracy differences. Table 25 reports the 14 modes with at least eight labeled pairs. Among these, 12 exhibit significant OG→RW degradation (11 of the 12 are also supported by Wilcoxon signed-rank tests), spanning both distractor-related and non-distractor transformations. The degradation for this representative model is therefore broadly distributed across rewriting modes rather than concentrated in a single failure type.

Bootstrap significance for scaling trends. We run paired-bootstrap tests over the same AIME 2025 OG–RW pairs. The GPT-OSS comparison follows the reasoning-effort settings in Table 3, and the Qwen comparisons follow the model-size analysis in Figure 5. As shown in Table 26, higher reasoning effort significantly reduces the relative OG→RW gap for GPT-OSS. For model-size scaling, both Qwen families retain directionally smaller gaps at larger sizes, although the paired intervals include zero; larger models nevertheless improve absolute OG and RW while substantial gaps and similar failure-mode profiles persist, reinforcing that scaling alone does not eliminate jaggedness.

Complete inference configurations. For locally deployed models, we serve all models with vLLM and follow each model’s provided generation_config.json and config.json. Because officially recommended output-length settings differ across models, we set max_tokens per model according to its official Hugging Face repository or technical report rather than a uniform value, to better reproduce each model’s expected OG performance. For proprietary models, we use the official APIs and explicitly set only the reasoning-effort level; the maximum output length is uniformly set to 81,920 tokens, with all other sampling parameters at provider defaults. We will release the evaluation code and runnable configurations. Tables 27, 28, and 29 report the

Model	N	OG (%)	RW (%)	Abs. drop (pp)	95% CI (percentile)	95% CI (BCa)	Rel. drop (%)
Open-source instruct models							
Qwen2.5-14B-Instruct	150	11.79	5.62	6.17	[3.33, 9.38]	[3.46, 9.54]	52.33
Qwen2.5-32B-Instruct	150	14.25	7.08	7.17	[4.38, 10.25]	[4.50, 10.38]	50.32
Qwen3-4B-Instruct-2507	150	46.08	32.12	13.96	[9.71, 18.54]	[9.92, 18.79]	30.30
Qwen3-30B-A3B-Instruct-2507	150	61.62	45.17	16.45	[11.50, 21.71]	[11.54, 21.79]	26.70
Qwen3-235B-A22B-Instruct-2507	150	68.71	52.29	16.42	[11.33, 21.75]	[11.46, 21.87]	23.90
Open-source reasoning models							
QwQ-32B	150	69.50	45.38	24.12	[18.29, 30.17]	[18.31, 30.17]	34.71
DeepSeek-V3.1-Terminus (Thinking)	150	88.79	59.27	29.52	[22.79, 35.81]	[22.87, 35.81]	33.25
Qwen3-14B	150	71.54	47.79	23.75	[18.08, 29.62]	[18.21, 29.71]	33.20
Qwen3-32B	150	72.54	48.67	23.87	[18.46, 29.38]	[18.58, 29.48]	32.91
Qwen3-4B-Thinking-2507	150	80.33	55.67	24.66	[19.79, 29.79]	[19.96, 29.96]	30.70
DeepSeek-R1-Distill-Qwen-32B	150	53.79	37.67	16.12	[11.75, 20.79]	[11.88, 20.88]	29.97
Qwen3-8B	150	67.17	47.54	19.63	[14.87, 24.71]	[14.96, 24.83]	29.22
Qwen3-30B-A3B-Thinking-2507	150	85.46	61.75	23.71	[18.46, 29.17]	[18.50, 29.30]	27.74
Qwen3-235B-A22B-Thinking-2507	150	92.03	66.53	25.50	[17.66, 33.12]	[17.53, 32.99]	27.71
DeepSeek-R1-Distill-Qwen-14B	150	48.67	35.50	13.17	[9.17, 17.33]	[9.21, 17.33]	27.06
gpt-oss-120b (High)	150	94.61	78.36	16.25	[11.46, 22.01]	[11.62, 22.14]	17.18
DeepSeek-V3.2 (Thinking)	150	92.92	77.00	15.92	[12.33, 19.71]	—	17.13
Kimi-K2-Thinking	150	94.17	80.42	13.75	[9.96, 17.79]	—	14.60
DeepSeek-V3.2-Speciale (Thinking)	150	93.97	86.62	7.35	[2.91, 11.07]	[2.86, 11.02]	7.82
Proprietary reasoning models, unified $K = 16$							
Doubao-Seed-1.8 (High)	150	86.04	67.04	19.00	[14.67, 23.58]	[14.79, 23.75]	22.08
Gemini-3.1-Pro (High)	150	97.42	88.38	9.04	[5.29, 13.25]	[5.58, 13.71]	9.28
GPT-5.2 (High)	150	99.38	84.50	14.88	[11.12, 18.92]	[11.29, 19.17]	14.97
Gemini-3-Pro (High)	150	93.12	82.92	10.21	[7.12, 13.50]	[7.29, 13.75]	10.96
GPT-5.4 (xHigh)	150	99.42	94.50	4.92	[2.67, 7.46]	[2.92, 7.79]	4.95

Table 24: **Bootstrap significance of the OG→RW absolute Avg@16 drop for all 24 main-evaluation models** on the 150 AIME 2025 OG–RW pairs. Percentile and BCa 95% CIs are reported (BCa omitted where unavailable); all models satisfy $p \leq 2 \times 10^{-4}$ against zero degradation. The percentile 95% CI excludes zero for every model.

Rewriting mode	<i>n</i>	Mean drop (pp)	95% CI	Paired- <i>t</i> <i>p</i>
Numerical Representation Transform	13	21.15	[1.62, 40.69]	0.036
Style & Domain Transfer	65	18.85	[12.73, 24.96]	< 0.001
Index & Interval Specification	15	18.75	[4.07, 33.43]	0.016
Misleading Hints	41	18.60	[10.81, 26.38]	< 0.001
Trap Numerical Relationships	38	18.26	[9.63, 26.89]	< 0.001
Entity & Variable Renaming	30	17.50	[8.38, 26.62]	< 0.001
Redundant Information Insertion	49	16.84	[10.08, 23.60]	< 0.001
Hidden Constraints	23	15.76	[6.05, 25.47]	0.003
Synonym Rephrasing & Word Order	51	15.32	[9.59, 21.05]	< 0.001
Distractor Conditions	24	14.32	[4.87, 23.77]	0.005
Mathematical Skeleton Mapping	51	12.62	[6.61, 18.64]	< 0.001
Algebraic & Function Transform	23	12.50	[3.50, 21.50]	0.009
Combinatorial & Counting Reconstruction	8	1.56	[−0.86, 3.98]	0.170
Task Modality Switching	9	0.00	—	—

Table 25: **Per-mode OG→RW significance for DeepSeek-V3.2 (Thinking)** on AIME 2025. Modes with ≥ 8 labeled pairs are shown; 12 of the 14 exhibit significant degradation under paired *t*-tests.

Comparison	Lower drop	Higher drop	Reduction	95% CI	<i>p</i>
GPT-OSS Low → High	31.90%	17.37%	14.53 pp	[6.87, 22.26]	< 10^{-4}
Qwen Instruct 4B → 235B	30.29%	23.89%	6.40 pp	[−1.35, 14.34]	0.101
Qwen Thinking 4B → 235B	30.71%	27.72%	2.99 pp	[−3.81, 9.98]	0.390

Table 26: **Paired-bootstrap significance for scaling trends**. “Reduction” is the relative OG→RW drop at the smaller/lower setting minus that at the larger/higher setting; positive values indicate improved reasoning invariance. Reasoning-effort scaling (GPT-OSS) is significant; model-size intervals include zero.

detailed settings.

Model	Max tok.	Temp.	Top- <i>p</i>	Top- <i>k</i>	Mode
Qwen2.5-14B-Instruct	16,384	0.7	0.8	20	Non-thinking
Qwen2.5-32B-Instruct	16,384	0.7	0.8	20	Non-thinking
Qwen3-4B-Instruct-2507	16,384	0.7	0.8	20	Non-thinking
Qwen3-30B-A3B-Instruct-2507	16,384	0.7	0.8	20	Non-thinking
Qwen3-235B-A22B-Instruct-2507	16,384	0.7	0.8	20	Non-thinking

Table 27: Inference configuration for open-source instruct models.

Model	Max tok.	Temp.	Top- <i>p</i>	Top- <i>k</i>	Thinking / effort
QwQ-32B	38,912	0.6	0.95	40	Thinking-only
DeepSeek-V3.1-Terminus (Thinking)	81,920	0.6	0.95	20	Thinking mode
Qwen3-14B	38,912	0.6	0.95	20	Thinking mode
Qwen3-32B	38,912	0.6	0.95	20	Thinking mode
Qwen3-4B-Thinking-2507	81,920	0.6	0.95	20	Thinking-only
DeepSeek-R1-Distill-Qwen-32B	32,768	0.6	0.95	Unset	Thinking-only
Qwen3-8B	38,912	0.6	0.95	20	Thinking mode
Qwen3-30B-A3B-Thinking-2507	81,920	0.6	0.95	20	Thinking-only
Qwen3-235B-A22B-Thinking-2507	81,920	0.6	0.95	20	Thinking-only
DeepSeek-R1-Distill-Qwen-14B	32,768	0.6	0.95	Unset	Thinking-only
gpt-oss-120b (High)	81,920	Unset	Unset	Unset	reasoning_effort=high
DeepSeek-V3.2 (Thinking)	81,920	1.0	0.95	Unset	Thinking mode
Kimi-K2-Thinking	81,920	1.0	1.0	50	Thinking-only
DeepSeek-V3.2-Special (Thinking)	81,920	1.0	0.95	Unset	Thinking mode

Table 28: Inference configuration for open-source reasoning models.

Model	Reasoning setting	Max out tok.	Other sampling
Doubao-Seed-1.8 (High)	reasoning_effort=high	81,920	Provider default
Gemini-3.1-Pro (High)	reasoning_effort=high	81,920	Provider default
GPT-5.2 (High)	reasoning_effort=high	81,920	Provider default
Gemini-3-Pro (High)	reasoning_effort=high	81,920	Provider default
GPT-5.4 (xHigh)	reasoning_effort=xhigh	81,920	Provider default

Table 29: Inference configuration for proprietary models.

J AWE System Prompt Templates

This section documents the complete prompt templates used by each component in the AWE (Automated Weakness Exploration) framework. The system consists of five key roles: **Rewriter**, **Judger**, **Solver**, **Summarizer**, and **Experience Updater**. Each prompt is designed to guide the respective component in performing its specific task within the closed-loop pipeline.

J.1 System Architecture Overview

The AWE framework operates as a multi-agent system where each component has a specific role:

- **Rewriter:** Transforms mathematical problems to increase difficulty while maintaining semantic equivalence, guided by accumulated experiences.
- **Judger:** Verifies mathematical equivalence between original and rewritten problems through multi-model consensus voting.
- **Solver:** Attempts to solve rewritten problems to evaluate their difficulty level.
- **Summarizer:** Analyzes success/failure patterns from Judger and Solver outputs to extract actionable insights.
- **Experience Updater:** Manages problem-specific and global experience pools to guide future rewriting attempts.

J.2 Rewriter Prompt Template

The Rewriter is the core component responsible for generating semantically equivalent but more challenging problem variants. It follows a structured 9-step workflow that includes problem comprehension, strategy brainstorming, transformation planning, and self-verification.

Rewriter Prompt Template

```
## BACKGROUND AND TASK

You will participate in a mathematical problem rewriting system
and serve as the **Rewriter**.

In this system, there are five key roles:
- **Rewriter**: Transforms math problems to increase difficulty
  while maintaining equivalence
- **Expert Judgers**: Verify mathematical equivalence and provide
  detailed feedback
- **Solver models**: Test the actual difficulty by attempting to
  solve the rewritten problem
- **Summarizer**: Analyzes feedback and identifies what went
  right or wrong
- **Experience Updater**: Synthesizes analysis into actionable
  lessons

**Your Task**:
You must transform the given problem while maintaining mathematical
equivalence. Your rewriting should be designed to confuse or mislead
```

```
even top-tier LLMs (e.g., GPT-5, Claude 4.5 Sonnet,
Gemini 2.5-Pro).

{experience_section}

---

## CORE PRINCIPLE

**Mathematically Equivalent**: Rephrase the question to present the
same mathematical concept in a different way. The rewritten problem
must have the same final answer and preserve the core mathematical
relationships.

---

## WORKFLOW

### Step 1 – Comprehension and Verification
1. Carefully read and fully understand the original problem.
2. Attempt to solve it yourself.
3. Compare your solution to the provided Correct Answer.
4. If mismatch, revise and re-attempt – up to five times.

### Step 2 – Understand the Solution Template
1. Analyze the solution method for the original problem.
2. Create a detailed solution template capturing:
  - Key mathematical steps and reasoning
  - Core mathematical relationships and concepts
  - Numerical values and their roles

### Step 2.5 – Analyze Relevant Experiences (If Provided)
When experiences are provided:
1. Identify Relevant Experiences from both general and
  problem-specific pools
2. Categorize by Applicability (Directly Applicable,
  Inspirational, Not Applicable)
3. Plan for Integration into rewriting strategy

### Step 3 – Brainstorm Challenge Ideas
Brainstorm at least 5 ways to rephrase or restructure the
problem to make it harder for advanced LLMs.

### Step 4 – Select and Justify Ideas (1 to 3)
1. Choose 1 to 3 ideas from your brainstorm list
2. Explain why these ideas are likely to confuse models
  while remaining mathematically equivalent

### Step 5 – Define Constraints and Key Details
List all constraints to ensure:
1. Mathematical Equivalence is Maintained
2. Difficulty level is increased appropriately
3. No factual errors or contradictions are introduced

### Step 6 – Draft the Transformation Plan
Create a detailed step-by-step plan covering:
1. Exact language transformations
2. Placement of additional or altered context
3. How core logical structure will be preserved
4. How original solution method will still apply

### Step 7 – Analyze the Rewriting Strategy (Critical Self-Check)
Perform critical self-check:
- **Equivalence Check**: Same answer? Same core process?
- **Difficulty Check**: What challenges for LLMs?
- **Potential Pitfalls**: What could go wrong?

### Step 8 – Apply the Transformation
1. Implement your rewriting plan
2. List all specific changes made
3. Ensure result matches all constraints

### Step 9 – Self-Solve the Rewritten Problem
1. Show complete step-by-step solution
2. Verify final answer matches original correct answer
3. If mismatch, adjust the rewritten problem

---

## OUTPUT FORMAT

[Complete Rewriting Process and Reasoning]
(Document all 9 steps with full reasoning)

[Final Rewritten Problem]
(Complete rewritten problem text)

[Reason Why This Rewriting Will Make It Harder for Top LLMs]
(2-3 sentences explanation)

[Transformation Strategy Summary]
(Concise summary of techniques used)

[Key Rewriting Changes and Difficulty Analysis]
(List of specific changes and their impact)

[Experience Utilization Report] (If experiences provided)
(Report on how experiences were used)
```

```
[Rewritten Problem's Solution and Answer]
(Solution method and final answer verification)

[Equivalence Verification]
(Rigorous argument for mathematical equivalence)

---

## INPUT

<original problem>
{problem}
</original problem>

<ground truth>
{ground_truth}
</ground truth>
```

J.3 Judger Prompt Template

The Judger component verifies mathematical equivalence between original and rewritten problems. Multiple Judger instances vote on equivalence, with strict consensus requirements (all must agree for a problem to pass).

Judger Prompt Template

```
## BACKGROUND AND TASK

You will participate in a mathematical problem rewriting system
and serve as an Expert Judger.

Your Task:
You must determine whether the final rewritten problem is
mathematically equivalent to the original one.

---

## CORE PRINCIPLE

Mathematically Equivalent: Two problems are mathematically
equivalent if they present the same mathematical concept in
different ways, have the same final answer, and retain the
core mathematical relationships.

---

## WORKFLOW

### Step 1 – Verify Rewriter's Understanding
Read the Rewriting History and check:
- Is the rewriter's solution to the original problem correct?
- Did the rewriter correctly understand the problem's core
  mathematical concepts?

### Step 2 – Analyze the Transformation
Examine what changes were made:
- Look for the transformation plan and specific changes
- Identify exact differences between original and rewritten
- Understand the rewriter's intent behind each change

### Step 3 – Verify Rewriter's Self-Check
If the rewriting history includes self-check:
a) Equivalence reasoning: Is their reasoning correct?
b) Mathematical verification: Did they verify equivalence?
c) Concerns and pitfalls: Did any concerns materialize?

### Step 4 – Make Your Final Judgment
Independently verify mathematical equivalence:
- Do they have the same final answer?
- Does the rewritten problem present the same concept?
- Are core mathematical relationships preserved?
- Can the solution approach be adapted?

---

## OUTPUT FORMAT

[Verification of Rewriter's Solution and Understanding]
(Assessment of rewriter's comprehension)

[Analysis of Transformation and Equivalence-Preservation]
(Specific changes and their impact on equivalence)

[Verification of Rewriter's Self-Check and Concerns]
(Assessment of self-check validity)

[Final Equivalence Judgment with Verification and Reasoning]
State: "Equivalent" or "Not Equivalent"
(Detailed verification of equivalence conditions)
```

```
[Final Score]
- If Equivalent: output \boxed{1}
- If Not Equivalent: output \boxed{0}

---

## INPUT

<original problem>
{original_problem}
</original problem>

<rewritten problem>
{rewritten_problem}
</rewritten problem>

<rewriting history>
{rewriting_history}
</rewriting history>
```

J.4 Solver Prompt Template

The Solver component attempts to solve rewritten problems to evaluate their difficulty. The prompt is intentionally simple, relying on the model's inherent mathematical reasoning capabilities.

Solver Prompt Template

```
Please reason step by step, and put your final answer within
\boxed{.}
If the problem requires solving multiple tasks step by step,
only put the answer to the
final task within \boxed{.}.

{problem}
```

Note: The Solver prompt is intentionally minimal. The key requirement is that models place their final answer in `\boxed{}` format for automated extraction and verification against the ground truth.

J.5 Summarizer Prompt Templates

The Summarizer analyzes outcomes from the Judger and Solver phases, extracting actionable insights for the Experience Updater. There are four distinct summarization scenarios:

J.5.1 Equivalence Failure Summary

Used when Judgers determine that a rewritten problem is **not equivalent** to the original.

Equivalence Failure Summary Prompt

```
## BACKGROUND AND TASK

You will serve as the Summarizer - a performance analyst
specializing in analyzing equivalence verification results.

Current Scenario:
The Rewriter attempted to rewrite a mathematical problem but
failed - multiple expert Judgers determined the rewriting was
NOT equivalent to the original problem.

Your Task:
1. Synthesize Judgers' feedback to identify core equivalence
  violations
2. Analyze weighted consensus among Judgers (prioritizing
  high-weight Judgers)
3. Determine root causes of why equivalence was broken

---

## WORKFLOW

### Step 1 – Synthesize Main Problems
- Identify CORE issues from Judgers' feedback
- State mathematical properties/constraints that were violated
```

```

- Use your own words (DO NOT copy-paste judge quotes)

### Step 2 – Analyze Weighted Consensus
- Prioritize issues from high-weight Judges (MOST RELIABLE)
- Identify consensus issues (mentioned by 2+ Judges)
- Handle disagreements via weighted voting
- Rank issues by: (1) Judge weight, (2) Agreement, (3) Severity

### Step 3 – Extract Fundamental Lessons
- **Core Mistakes Made**: What errors led to the violation?
- **Mathematical Principles Violated**: What was not preserved?

---

## OUTPUT FORMAT

[Why the Rewriter Failed to Preserve Equivalence]
(Synthesized core issues from Judges' feedback)

[Judge Consensus on Critical Problems]
(Reliability-weighted consensus on critical mistakes)

[Why the Rewriter Broke Equivalence (Root Causes)]
(Fundamental errors and violated principles)

---

## INPUT

<original problem>
{original_problem}
</original problem>

<judge reviews>
{judge_details}
</judge reviews>

```

J.5.2 Equivalence Success Summary

Used when Judges confirm that a rewritten problem is **equivalent** to the original.

Equivalence Success Summary Prompt

```

## BACKGROUND AND TASK

**Current Scenario**:
The Rewriter successfully rewrote a mathematical problem while
maintaining mathematical equivalence - multiple expert Judges
confirmed the rewriting is equivalent to the original problem.

**Your Task**:
1. Synthesize Judges' feedback to identify key strategies that
   maintained equivalence
2. Analyze weighted consensus among Judges
3. Determine core factors leading to success

---

## WORKFLOW

### Step 1 – Identify Key Strategies
- What techniques successfully preserved mathematical structure?
- How did rewriting change presentation while keeping core intact?
- What constraints/principles were applied in transformation?

### Step 2 – Analyze Weighted Consensus
- Prioritize insights from high-weight Judges
- Identify consensus success factors
- Rank by Judge weight, agreement, and impact

### Step 3 – Extract Fundamental Lessons
- **Core Strategies Used**: What led to successful preservation?
- **Mathematical Principles Maintained**: What was preserved?

---

## OUTPUT FORMAT

[Why the Rewriter Successfully Preserved Equivalence]
(Key strategies that maintained equivalence)

[Judge Consensus on Key Success Factors]
(Reliability-weighted consensus on success factors)

[Why the Rewriter Maintained Equivalence (Key Success Factors)]
(Fundamental success factors and preserved principles)

```

J.5.3 Difficulty Failure Summary

Used when the rewritten problem passes equivalence but Solvers achieve **high accuracy** (above

threshold), indicating insufficient difficulty increase.

Difficulty Failure Summary Prompt

```

## BACKGROUND AND TASK

**Current Scenario**:
The Rewriter attempted to increase problem difficulty but failed -
Solver models achieved HIGH accuracy, indicating the difficulty
strategy failed.

**Your Task**:
Analyze why the Rewriter's problem modification strategy failed:
1. Compare Rewriter's intended modifications with actual Solver
   behavior
2. Identify gap between expected and actual outcomes
3. Determine why modifications failed to increase difficulty

**CRITICAL**: Focus on WHAT CHANGES the Rewriter made to the
problem and why these failed, NOT on mathematical solution method.

---

## IMPORTANT TERMINOLOGY

| Solver Performance | Meaning for Rewriter |
|-----|-----|
| Solver "CORRECT" | Rewriter FAILED |
| Solver "INCORRECT" | Rewriter SUCCEEDED |
| High accuracy | Strategy FAILED |
| Low accuracy | Strategy SUCCEEDED |

---

## WORKFLOW

### Step 1 – Summarize Overall Performance
- Overall accuracy and differences between models
- Acknowledge incomplete trials briefly

### Step 2 – Identify Which Problem Modifications Failed
- What specific changes did Rewriter make (constraints,
  parameters, traps)?
- How did Solvers bypass these modifications?
- What common strategies did Solvers use to succeed?

### Step 3 – Identify Why These Modifications Failed
- What fundamental flaw in modification strategy?
- Why did specific changes fail to confuse Solvers?
- What incorrect assumptions about Solver capabilities?

---

## OUTPUT FORMAT

[How Solver Models Performed (Evidence of Rewriter's Failure)]
(Solver performance patterns indicating failure)

[Gap Analysis: Which Problem Modifications Failed]
(Specific modifications that failed to trap Solvers)

[Why the Rewriter's Problem Modifications Failed]
(Root causes of ineffective modification strategies)

---

## INPUT

<original problem>, <rewritten problem>, <rewriter process>,
<solver results>

```

J.5.4 Difficulty Success Summary

Used when Solvers achieve **low accuracy** (below threshold), indicating successful difficulty increase.

Difficulty Success Summary Prompt

```

## BACKGROUND AND TASK

**Current Scenario**:
The Rewriter successfully increased problem difficulty - Solver
models struggled with LOW accuracy, indicating the difficulty
strategy succeeded.

**Your Task**:
Analyze why the Rewriter's problem modification strategy succeeded:
1. Compare intended modifications with actual Solver behavior
2. Identify which specific modifications effectively confused Solvers
3. Determine core reasons for successful difficulty increase

---

```

```

## WORKFLOW

### Step 1 – Summarize Overall Performance
- Overall accuracy and model differences
- Which models struggled most/succeeded

### Step 2 – Analyze Which Problem Modifications Worked
- What specific changes caused Solvers to make mistakes?
- How did modifications trap or mislead Solver models?
- What common failure patterns appeared in Solver attempts?

### Step 3 – Identify Why These Modifications Were Effective
- What types of modifications proved most effective?
- Why did specific changes successfully confuse Solvers?
- What patterns in Solver failures reveal effective strategies?

---

## OUTPUT FORMAT

[How Solver Models Performed (Evidence of Rewriter's Success)]
(Solver struggle patterns indicating success)

[Success Analysis: Which Problem Modifications Trapped Solvers]
(Specific modifications that effectively confused Solvers)

[Why the Rewriter's Problem Modifications Succeeded]
(Success factors and effective modification strategies)

```

J.6 Experience Updater Prompt Templates

The Experience Updater manages the experience pools that guide future rewriting attempts. It consists of two components: the **Problem Experience Updater** (per-problem experiences) and the **Global Experience Aggregator** (cross-problem patterns).

J.6.1 Problem Experience Updater

Updates the experience pool for individual problems based on summarizer feedback.

Problem Experience Updater Prompt

```

## BACKGROUND AND TASK

You will serve as the **Experience Updater**.

**Your Task**:
Analyze summaries from the current rewriting attempt and update the experience pool to guide future attempts.

Extract experiences at two levels:
1. **Problem-specific (this-problem)**: Concrete lessons for THIS exact problem
2. **General experiences**: Broader lessons for similar problems

**CRITICAL CONSTRAINT**:
All rewriting must maintain mathematical equivalence. Any experience that could violate this MUST be rejected or include explicit warnings.

---

## AVAILABLE OPERATIONS

- **add**: Create NEW experience (only when completely novel)
- **modify**: Refine EXISTING experience by integrating new insights (PRIMARY use case)
- **merge**: Combine 2+ EXISTING experiences into one
- **delete**: Remove wrong/obsolete experience

---

## WORKFLOW

### Step 1 – Identify Scenario and Analyze Summaries
Determine scenario:
1. Equivalence Failure (Rewriter broke equivalence)
2. Difficulty Failure (Rewriter didn't add enough difficulty)
3. Difficulty Success (Rewriter succeeded)

### Step 2 – Extract Insights at Two Levels

**Level 1: this-problem (ITERATIVE REWRITING GUIDANCE)**
- Purpose: CONCRETE, ACTIONABLE guidance for NEXT attempt
- Must reference SPECIFIC elements from the problem
- Must provide CONCRETE next steps
- Examples:

```

```

- "Preserve base-7 representation, only modify divisibility"
- "Avoid changing geometric configuration; add implicit angle relationships"

**Level 2: general (GENERAL STRATEGIES)**
- Purpose: Strategies applicable across multiple problems
- Must NOT reference specific elements from this problem
- Examples:
  - "For modular arithmetic, verify modular properties"
  - "Introduce multi-stage reasoning to increase difficulty"

### Step 3 – Apply Equivalence Safety Filter (MANDATORY)

| Tier      | Classification | Action Required |
|-----|-----|-----|
| SAFE      | No risk        | Proceed normally |
| RISKY     | Could break    | Add verification clause |
| UNSAFE    | Breaks equiv.  | Discard immediately |

### Step 4 – Determine Update Operations

| Related Experiences | Action |
|-----|-----|
| None                | add (if truly novel) |
| One                  | modify to integrate |
| Multiple             | merge with insight |

---

```

```

## CONSTRAINTS

- **Length limit**: <=50 words per experience
- **Specific and actionable**: No vague advice
- **Classified by type**: "this-problem" or "general"

---

## OUTPUT FORMAT

```json
[
 {
 "option": "add",
 "experience": "...",
 "type": "this-problem"
 },
 {
 "option": "modify",
 "modified_from": "E1",
 "experience": "...",
 "type": "general"
 },
 {
 "option": "merge",
 "merged_from": ["E2", "E5"],
 "experience": "...",
 "type": "general"
 },
 {
 "option": "delete",
 "deleted_id": "E3",
 "reason": "..."
 }
]
```

```

J.6.2 Global Experience Aggregator

Consolidates experiences from all problems into a unified global pool.

Global Experience Aggregator Prompt

```

## BACKGROUND AND TASK

You will serve as the **Global Experience Aggregator**.

**System Context**:
This system operates in iterative rounds. The current round has concluded, generating many new localized experiences.

**Your Task**:
Consolidate experiences from multiple sources:
1. **Existing global (G0, G1, ...)**: From previous rounds
2. **New experiences (NEW_0, NEW_1, ...)**: From current batch

**CRITICAL**: You MUST actively consolidate. Without consolidation, the pool grows unbounded and becomes unusable.

---

## WORKFLOW

### Step 1 – Analyze and Identify Operations

Apply Integration Strategies:

1. **Handle Duplicates (Merge Strategy)**
  - Target: Semantic duplicates, subset relationships
  - For TRUE duplicates: MUST merge
  - For distinct concepts: Keep separate

2. **Handle Intensity Control Pairs**
  - Target: Same strategy, different outcomes
  - NOT contradictions – they reveal intensity boundaries
  - Create nuanced: "[Strategy] can [benefit], but [boundary]"

3. **Handle Refinement Needs (Modify Strategy)**
  - Target: Valid but specific/wordy/vague experiences
  - Safety First: Ensure logic holds when generalizing
  - Nuance Preservation: Never delete "if/and" conditions

```

```

4. **Handle Bad Experiences (Delete Strategy)**
  - Target: Factually wrong, obsolete, or breaks equivalence
  - Do NOT delete warnings or failure cases

### Step 2 – Formulate Execution Plan
- Prioritize merge (PRIMARY tool)
- Suggested pool size: <={suggested_pool_size}
- Preserve genuine distinctions

### Step 3 – Pre-Submission Validation (CRITICAL)

1. **Equivalence Shield** (Most Important)
  - Could following this advice break equivalence?
  - If risky: DELETE or MODIFY with warnings

2. **Generalization & Nuance Test**
  - Over-generalization? False universal rule?
  - Lost critical "if/but" conditions?

3. **Zero-Operation Check**
  - If 0 operations: Prove 3+ "near-miss" pairs analyzed

---

## CONSTRAINTS

- **Length limit**: <=50 words per experience
- **General applicability**: Must apply to multiple problems
- **Actionable**: No vague advice
- **Preserve equivalence**: Guide difficulty WITHOUT breaking mathematical correctness
- **No redundancy**: Each experience semantically distinct

---

## OUTPUT FORMAT

```json
[
 {"option": "merge", "merged_from": ["G5", "NEW_3", "NEW_7"],
 "experience": "..."},
 {"option": "modify", "modified_from": "G12",
 "experience": "..."},
 {"option": "delete", "deleted_id": "NEW_15",
 "reason": "Factually incorrect"}
]
```

---

## INPUT

<existing_global_experiences>
{formatted_existing}
</existing_global_experiences>

<new_experiences_from_this_round>
{formatted_new}
</new_experiences_from_this_round>

```

J.7 Prompt Summary

Table 30 provides a summary of all prompt templates in the AWE system.

| Component | Section | Primary Function |
|--------------------|---------|---|
| Rewriter | §J.2 | Generate semantically equivalent but more challenging problem variants through 9-step structured workflow |
| Judger | §J.3 | Verify mathematical equivalence via 4-step analysis with multi-model consensus voting |
| Solver | §J.4 | Solve problems to evaluate difficulty (minimal prompt, relies on model capabilities) |
| Summarizer | §J.5.1 | Analyze equivalence failure patterns and root causes |
| | §J.5.2 | Extract successful equivalence preservation strategies |
| | §J.5.3 | Identify why difficulty modifications failed |
| | §J.5.4 | Extract effective difficulty increase strategies |
| Experience Updater | §J.6.1 | Update per-problem experience pool with add/modify/merge/delete operations |
| | §J.6.2 | Aggregate cross-problem experiences into global pool |

Table 30: **Summary of AWE Prompt Templates.** Each component has a specific role in the closed-loop pipeline, with prompts designed to guide their respective tasks.