# CoDiQ: Test-Time Scaling for Controllable Difficult Question Generation

Anonymous ACL submission

## Abstract

Large Reasoning Models (LRMs) benefit substantially from training on challenging competition-level questions. However, existing automated question synthesis methods lack precise difficulty control, incur high computational costs, and struggle to generate competition-level questions at scale. In this paper, we propose **CoDiQ** (**Co**ntrollable **Di**fficult **Q**uestion Generation), a novel framework enabling fine-grained difficulty control via test-time scaling while ensuring question solvability. Specifically, first, we identify a test-time scaling tendency (extended reasoning token budget boosts difficulty but reduces solvability) and the intrinsic properties defining the upper bound of a model's ability to generate valid, high-difficulty questions. Then, we develop **CoDiQ-Generator** from Qwen3-8B, which improves the upper bound of difficult question generation, making it particularly well-suited for challenging question construction. Building on the CoDiQ framework, we build **CoDiQ-Corpus** (44K competition-grade question sequences). Human evaluations show these questions are significantly more challenging than LiveCodeBench/AIME with over 82% solvability. Training LRMs on CoDiQ-Corpus substantially improves reasoning performance, verifying that scaling controlled-difficulty training questions enhances reasoning capabilities. We open-source CoDiQ-Corpus, CoDiQ-Generator, and implementations to support related research.

## 1 Introduction

The rapid advancement of Large Reasoning Models (LRMs) has demonstrated remarkable capabilities in complex reasoning, with recent works achieving impressive performance on challenging benchmarks across mathematics and coding (Zhao et al., 2025; Manem et al., 2025; Pei et al., 2025). A crucial factor driving these improvements is high-quality training data that truly stress reasoning. Importantly, much like scientific discovery, finding the right difficult questions can be as critical as solving them. As human-driven pipelines are expensive and hard to scale, purely synthetic pipelines often struggle to push difficulty due to three major challenges. First, there is a *generator capacity ceiling*, where models fail to synthesize questions substantially harder than what they can reliably reason about. Second, the *solvability–complexity trade-off* implies that forcing complexity often breaks logical consistency. Finally, *difficulty definition and control* remains an obstacle; without a measurable surrogate, "make it harder" becomes uncontrollable.

To address these challenges, we propose **CoDiQ**, a framework that introduces *depth-wise* test-time scaling into question generation—in contrast to traditional width-wise test-time compute (e.g., best-of-$N$ or tree search (Snell et al., 2024)). CoDiQ utilizes iterative refinement to drive the generator toward its capability frontier through three coupled innovations: (1) an iterative pipeline utilizing six *Difficulty-Enhancement Strategies* with hybrid verification; (2) the **CoDiQ-Generator**, trained via Reinforcement Learning on its own *breaking-point* trajectories to track the capability boundary rather than average-case difficulty; and (3) a relative difficulty paradigm using LLM listwise ranking and a *ValueNetwork* that yields continuous scores. Using this framework, we construct **CoDiQ-Corpus**, comprising 44K competition-grade math and coding question sequences. Figure 1 summarizes CoDiQ's overall advantages across five key dimensions compared to recent synthetic baselines.

Our key contributions are:

- **A Depth-Wise Test-Time Scaling Framework**. We pioneer test-time scaling for question generation, coupling iterative mutation with breaking-point RL to effectively target the generator's capability frontier.

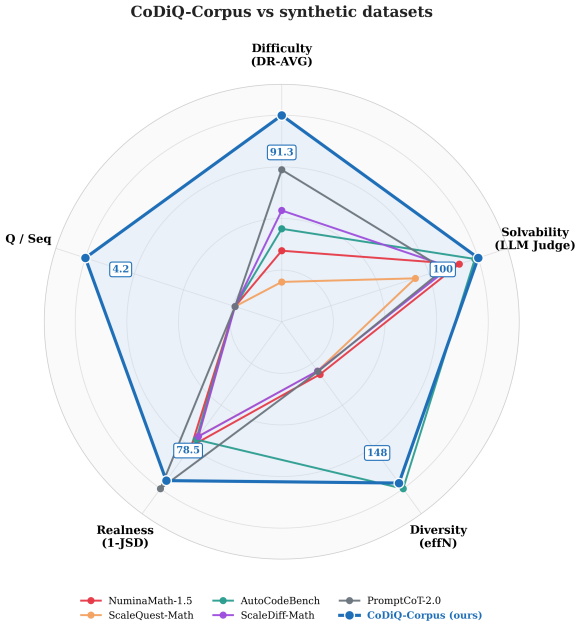


Figure 1: **CoDiQ-Corpus vs. recent synthetic datasets** on difficulty, solvability, diversity, realness ($1-$JSD to real corpora), and questions per sequence.

- **Conditional Scaling Tendency for Difficulty**. We validate a scaling tendency linking test-time compute to difficulty, demonstrating it holds under explicit boundary conditions and corroborating it with external evidence on natural problems.
- **CoDiQ-Corpus**. We release 44K sequences that surpass prior baselines on human-validated difficulty and real-world domain alignment (lowest JSD to real competition corpora). Curriculum fine-tuning consistently improves Qwen3 models across MATH-500, AIME 2024, MBPP, and LiveCodeBench v5.

We open-source CoDiQ-Corpus, CoDiQ-Generator, and all implementations.

## 2 Related Works

Generating difficult yet valid questions has become a key lever for scaling reasoning supervision (He et al., 2024; Sun et al., 2025; Luong et al., 2025).

**Prompt-based and agentic pipelines.** These methods bootstrap from seeds or concepts and iteratively refine candidates with self-critique, filtering, and verification (Liu et al., 2024; Wang et al., 2025; Zheng et al., 2025). PromptCoT (Zhao et al., 2025) conditions on concept samples and design cues, then applies rejection sampling. CogAtom (Chen et al., 2025) decomposes solutions into cognitive atoms and recombines them. For code, AutoCode (Zhou et al., 2025) uses closed-loop setter pipelines that jointly produce statements and tests. These approaches are effective but rely on multi-step orchestration and heavy post-hoc filtering.

**Trained generators.** A complementary line trains dedicated generators to amortize agentic cost (Hu et al., 2025; Christ et al., 2024; Xie et al., 2024). ScaleQuest (Ding et al., 2025a) aligns generators via Question Fine-Tuning and Preference Optimization; ScaleDiff (Pei et al., 2025) trains a generator on a curated hard subset. These methods scale well but offer coarse difficulty knobs and still depend on post-hoc validity checks.

In contrast, CoDiQ uses depth-wise test-time scaling as the core mechanism for fine-grained difficulty control under verifiable solvability, enabling frontier tracking of hard-yet-solvable questions without relying on heavy filtering.

## 3 Method

### 3.1 Overview

Our method aims to endow LRMs with scalable test-time question generation capability by enabling them to synthesize progressively challenging yet valid questions. In contrast to width-wise test-time scaling (e.g., best-of-$N$, tree search) that searches *outputs* for a fixed problem (Snell et al., 2024), CoDiQ uses *depth-wise* scaling—iterative, sequential refinement—to evolve the *problem itself* along the generator's capability frontier. The three components below are designed to be coupled rather than additive: the strategies define the search direction, the pipeline enforces validity along every depth step, and the verification modules make the depth-vs-validity trade-off measurable so that RL can target the precise breaking point. To achieve this, we first introduce six Difficulty-Enhancement Strategies (§3.2) that explicitly guide LRMs toward difficulty-scaling reasoning and hard question construction. These strategies are instantiated within the CoDiQ Pipeline (§3.3), which integrates two verification modules—difficulty estimation (§3.3.1) and solvability verification (§3.3.2)—to jointly regulate both difficulty and validity. Leveraging this pipeline, we construct CoDiQ-Bench to systematically benchmark models' question-generation performance under a unified evaluation framework. Then, we develop a specialized CoDiQ-Generator through reinforcement learning (§3.4), utilizing pipeline-derived feedback signals to further enhance the difficulty and reliability of synthesized

**Algorithm 1** CoDiQ Iterative Pipeline

**Input:** Seed Question $Q_0$, Max Rounds $T_{max}$
**Output:** Evolved Questions $\mathcal{Q}$
Initialize $\mathcal{Q} \leftarrow \emptyset$, $d_0 \leftarrow \textsc{Difficulty}(Q_0)$
**for** $i = 1$ **to** $T_{max}$ **do**
  $Q_i \leftarrow \text{LLM}(Q_{i-1})$
  $d_i \leftarrow \textsc{Difficulty}(Q_i)$
  **if** $\textsc{Valid}(Q_i) = \text{False}$ **or** $d_i < d_{i-1}$ **then**
    **break**
  **end if**
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q_i\}$
**end for**
**return** $\mathcal{Q}$

questions. Finally, we construct CoDiQ-Corpus, a dataset of 44K competition-grade mathematical and coding questions based on our CoDiQ-Generator. The detailed statistics are provided in Appendix G.

## 3.2 Difficulty-Enhancement Strategies

To systematically scale problem difficulty beyond naive prompting (e.g., "make this harder"), we design six Difficulty-Enhancement Strategies (detailed in Appendix S) that serve as explicit cognitive scaffolds for LLMs. These strategies—*Dimensionality & Constraints*, *Mathematical Abstraction*, *Inverse & Constructive*, *State Explosion*, *Theorem Disguise*, and *Edge Case & Rigor Engineering*—guide the model to inject algorithmic difficulty systematically, ensuring difficulty arises from reasoning depth rather than superficial modifications.

## 3.3 CoDiQ Pipeline

Building on the strategies (§3.2), the CoDiQ Pipeline (Algorithm 1) systematically scales difficulty through iterative refinement: a seed question $Q_0$ progressively evolves into harder variants $\{Q_1, \ldots, Q_n\}$ over up to $T_{\text{max}}$=8 rounds, where the model is prompted with "*Can you make it more difficult?*" at each step to trigger deeper reasoning.

To ensure generation quality, the pipeline incorporates two validation modules—*Difficulty Estimation* (§3.3.1) to enforce monotonic complexity growth and *Solvability Verification* (§3.3.2) to enforce logical validity—coupled with strict stopping rules (§3.3.3).

### 3.3.1 Difficulty Estimation

Because CoDiQ targets the capability frontier, absolute scoring saturates. We instead use *relative* estimation combining explicit *LLM-Ranking* and implicit *ValueNetwork Scoring*.

**LLM-Ranking.** We use Doubao-Seed-1.8 (Seed, 2025) as a listwise ranker over a query batch $\mathcal{Q}$, with stochastic shuffling $\tau(\mathcal{Q})$ to mitigate positional bias. The structured `JSON` output yields ranked indices used to allocate compute $K$ to harder queries (template in Appendix P).

**ValueNetwork Scoring.** Extending (Zhu et al., 2025), we extract Qwen3-8B hidden states under a quadratic sampling schedule (denser at reasoning onset, window ≤4,096 tokens) and train an MLP via binary cross-entropy to predict correctness on a mixture of benchmarks (Cobbe et al., 2021; Jia LI et al., 2024; Jain et al., 2024; Li et al., 2022); lower predicted probability indicates higher difficulty (Appendix E.5, E).

**Normalization.** Discrete group rankings are linearly scaled to $[0, 1]$: $d_i = (j-1)/(G-1)$ for group $g_j$ among $G$ groups (Eq. 1).

$$d_i = \frac{j-1}{G-1}, \quad j \in \{1, \ldots, G\}. \tag{1}$$

### 3.3.2 Solvability Verification

While difficulty estimation ensures monotonic complexity growth, it does not guarantee logical validity. To prevent invalid or unsolvable instances, we utilize Qwen3-32B (Yang et al., 2025) to verify the solvability of generated instances. Following the template in Appendix Q, the model generates responses in `JSON` format, from which we extract a binary solvability status together with a confidence score. **Roll-back rule.** If the candidate $Q_i$ is rejected, the pipeline does not abort the trajectory: it discards $Q_i$ and emits the last accepted version $Q_{i-1}$, so each retained sequence is solvable end-to-end. Human evaluation (§5.2) confirms 82% precision for accepted instances and 90% NPV for rejected ones, validating this rule.

### 3.3.3 Termination Criteria

The pipeline terminates at step $i$ if (1) the difficulty is non-monotonic ($d_i < d_{i-1}$) or (2) $Q_i$ is flagged unsolvable; the invalid $Q_i$ is dropped and the sequence $\{Q_1, \ldots, Q_{i-1}\}$ is emitted. Cases and failure modes are in Appendix C, D.

CoDiQ-Bench, a 200-question diagnostic set (100 math + 100 code), is described in Appendix A.

### 3.4 CoDiQ-Generator

To further enhance the pipeline's capacity for high-difficulty, high-quality generation, we develop a specialized **CoDiQ-Generator** via reinforcement learning. By directly optimizing the model's question-setting behavior with reward signals derived from solvability confidence, difficulty progression, and validity, the generator improves both the upper bound and the stability of synthesized problems.

**RL Data: Breaking-Point Trajectories.** Rather than maximizing absolute difficulty, we target Qwen3-8B's *capability boundary* (Zhang et al., 2025): from CoDiQ Pipeline runs we retain trajectories where the model succeeds for rounds $1{:}i{-}1$ but fails at round $i$ (unsolvable or stagnant), and use the failed step together with the last accepted state as a training pair. This converts each "breaking point" into a precise learning signal anchored at the frontier rather than at average-case difficulty. Seeds span eight benchmarks—Math12k (hiyouga, 2025), GSM8K (Cobbe et al., 2021), SVAMP (ChilleD, 2024), ASDiv (yimingzhang, 2025), CodeAlpaca (Chaudhary, 2023), LeetCodeDataset (Xia et al., 2025), MBPP (Austin et al., 2021), DS-1000 (Lai et al., 2022)—yielding $|\mathcal{D}_{RL}|$=1,173 samples.

**RL Training Paradigm.** Building on the success of R1-style methods with discrete, rule-based rewards (Shao et al., 2024), we further refine Qwen3-8B with GRPO (Shao et al., 2024) in the VeRL framework (Sheng et al., 2025). We design a difficulty-aware reward that balances validity guarantees with progressive scaling. Let $\text{conf}=\max(0.5, \text{confidence}(x))$ and $\Delta(D)=d_i - d_{i-1} \in [-1, 1]$ for $i \in \{1, \ldots, R\}$, where $d_i$ follows Eq. (1) and $R$ is the maximum evolution rounds:

$$r = \begin{cases} 0, & \text{invalid} \\ w_1 \cdot \text{conf}, & \Delta(D) = 0 \\ w_2 \cdot \text{conf} + (1{-}w_2)(w_3 + (1{-}w_3)\Delta(D)), & \Delta(D) > 0 \end{cases} \tag{2}$$

where invalid cases include unsolvable questions, repetitive outputs, or negative difficulty changes ($\Delta(D) < 0$).[1]

[1]We use $w_1$=0.6, $w_2$=0.2, $w_3$=0.8, partitioning the reward space into three non-overlapping bands (invalid $\rightarrow$ 0; held $\rightarrow [0.30, 0.60]$; increased $\rightarrow [0.74, 1.00]$). The pipeline is robust to small perturbations as long as this ordinal hierarchy is preserved; details and ablations are in Appendix N.

## 4 Experiments

### 4.1 Experimental Setup

**Baselines.** We compare against LRMs with extended reasoning capabilities: GLM-4.6 (GLM et al., 2024), GPT-OSS-20B (Agarwal et al., 2025), GLM-Z1-9B-0414 (GLM et al., 2024), and the Qwen3 series (Yang et al., 2025) (0.6B/1.7B/4B/8B/14B/32B). All baselines run inside the CoDiQ Pipeline (§3.3).

**Metrics.** **DR-LLM** (listwise ranking by Doubao-Seed-1.8 (Seed, 2025), §3.3.1) and **DR-VN** (ValueNetwork score, §3.3.1); both normalized to $[0, 1]$ and reported as percentages on CoDiQ-Bench.

### 4.2 Main Results

#### 4.2.1 Maximum Solvable Difficulty

Table 1: **Performance of different Long-CoT models on CoDiQ-Bench.** Group rankings based on the highest difficulty of solvable questions generated across 8 rounds without difficulty degradation on CoDiQ-Bench. The best, the second-best and the third-best scores for each indicator are shown in $\boxed{box}$, **bold** and underlined, respectively.

| Model | Rounds | Tokens | DR-LLM | DR-VN | DR(AVG) |
|---|---|---|---|---|---|
| ***Direct Prompt*** | | | | | |
| GPT-OSS-20B | 2.9 | 5528.2 | 68.5 | **74.4** | **71.5** |
| GLM-4.6 | 2.8 | 3385.8 | **71.2** | 65.8 | 68.5 |
| Qwen3-32B | 2.3 | 1239.3 | 50.6 | 54.8 | 52.7 |
| Qwen3-8B | 3.4 | 1130.5 | 39.2 | 59.6 | 49.4 |
| GLM-Z1-9B-0414 | 2.7 | 1229.8 | 48.8 | 43.7 | 46.3 |
| Qwen3-14B | 3.1 | 2076.4 | 45.9 | 44.4 | 45.2 |
| Qwen3-4B | $\boxed{4.2}$ | 1419.7 | 36.8 | 40.4 | 38.6 |
| Qwen3-1.7B | 3.3 | 844.5 | 25.6 | 37.1 | 31.4 |
| Qwen3-0.6B | 2.4 | 314.3 | 17.2 | 35.0 | 26.1 |
| ***CoDiQ Prompt(ours)*** | | | | | |
| GLM-4.6 | 2.7 | 7143.8 | $\boxed{73.2}$ | $\boxed{83.3}$ | $\boxed{78.3}$ |
| GPT-OSS-20B | 2.1 | $\boxed{8057.3}$ | 63.8 | 61.5 | 62.7 |
| Qwen3-32B | 2.2 | 4893.6 | 63.0 | 46.5 | 54.8 |
| Qwen3-14B | 2.6 | 5281.9 | 53.9 | 44.2 | 49.1 |
| Qwen3-4B | 2.8 | 4422.3 | 49.1 | 42.7 | 45.9 |
| Qwen3-8B | 2.4 | 4155.6 | 49.8 | 41.9 | 45.8 |
| GLM-Z1-9B-0414 | 1.7 | 3638.3 | 54.7 | 30.0 | 42.4 |
| Qwen3-1.7B | 1.4 | 2975.7 | 32.3 | 37.3 | 34.8 |
| Qwen3-0.6B | 1.0 | 2052.7 | 22.4 | 29.2 | 25.8 |
| ***CoDiQ Generator(ours)*** | | | | | |
| CoDiQ-Gen-8B | **3.4** | **7499.6** | 58.9 | 58.1 | 58.5 |

We instantiate each backbone as the generator inside the CoDiQ Pipeline and report difficulty on CoDiQ-Bench in Table 1.

**Effectiveness of the CoDiQ Prompt.** The CoDiQ Prompt induces a substantial expansion in reasoning tokens across every architecture, indicating that it successfully triggers extended reasoning trajectories rather than surface-level rewordings. As a result, almost all backbones produce

harder questions when conditioned on our prompt, with the largest gains observed on small-to-mid-sized models that previously plateaued under direct prompting.

**Superiority of CoDiQ-Generator.** CoDiQ-Gen-8B outperforms the much larger Qwen3-32B in generating high-complexity instances. We attribute this to RL alignment on breaking-point trajectories (§3.4): by optimizing for solvability and difficulty progression jointly, the generator maintains validity across iterative evolution and sustains the pipeline for more rounds, thereby accumulating complexity monotonically without premature termination.

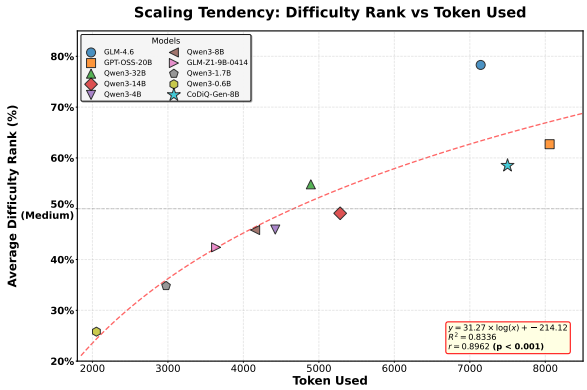


Figure 2: **Difficulty vs. tokens on CoDiQ-Bench.** Each point is one model under the CoDiQ Prompt; reasoning tokens correlate with DR-AVG.

#### 4.2.2 Difficulty Metrics Comparison

To further verify the number of tokens consumed by LRMs can estimate question difficulty, we highlight the positive correlation between token volume and difficulty rankings shown in Figure 2. We further validated this relationship by analyzing the correlation between token consumption and our established metrics (DR-LLM and DR-VN), yielding Pearson coefficients of $r = 0.8299$ ($p < 0.001$) and $r = 0.8545$ ($p < 0.001$), respectively.

**Boundary conditions of the scaling claim.** We do *not* claim that artificially forcing more tokens makes a problem intrinsically harder. The relation between solver tokens and difficulty holds under two explicit conditions: (i) a *fixed scaling protocol*—the prompt and difficulty-injection strategy are held constant across compared instances; and (ii) the *evaluator is within its competence range*—instances exceeding the verifier's reasoning horizon are excluded, otherwise we hit the Verifier Paradox (§5.2). To show that the relation is not specific to our pipeline, we additionally measure solver tokens on MATH-500, whose difficulty levels were assigned by human annotators prior to any LLM, and observe a strict monotonic increase from Level 1 to Level 5 (Appendix K, Table 14). This confirms that token consumption is a genuine—not pipeline-induced—proxy for intrinsic difficulty under the conditions above.

### 4.3 Ablation Study

**Upper Bound of Difficulty Generation.** Section 4.2.1 measured the maximum *solvable* difficulty, where the verifier inherently caps the ceiling—highly complex questions may not be unsolvable per se, but rather beyond the verifier's capacity to certify. To probe the theoretical upper bound independent of validity constraints, we remove the solvability verifier from the pipeline. The CoDiQ Prompt elevates the ceiling across all Qwen3 backbones, and CoDiQ-Gen-8B exceeds Qwen3-14B at far fewer parameters, indicating that RL alignment unlocks complex logical structures beyond what scale alone provides. Full scores are reported in Appendix B (Table 5).

**Impact of Maximum Token Budget.** We further examine difficulty efficiency under strict cumulative token budgets that bound both generation and verification (Figure 3). When the budget is exceeded, the iteration is discarded and the system reports the highest-difficulty valid problem from preceding rounds. Across 8k/16k/32k thresholds, CoDiQ-Gen-8B consistently dominates Qwen3-8B under both Direct and CoDiQ prompts, indicating that our RL-aligned generator extracts more difficulty per unit of compute rather than merely consuming more compute. The gap between CoDiQ-Prompted Qwen3-8B and the Direct-Prompt counterpart further validates the prompt itself as an independent source of efficiency.

### 4.4 Scaling Tendency Analysis

The previous analyses pinned down static ceilings (§4.2.1, §4.3). Here we shift to *how* models reach those ceilings by tracking per-round dynamics on difficulty and solvability (Figures 4–5; full results in Appendix L).

**Difficulty Scaling.** Compared to Direct Prompt, CoDiQ Prompt elicits markedly deeper reasoning: token consumption increases sharply, and a consistent upward difficulty trajectory appears across most backbones. Large models tend to saturate

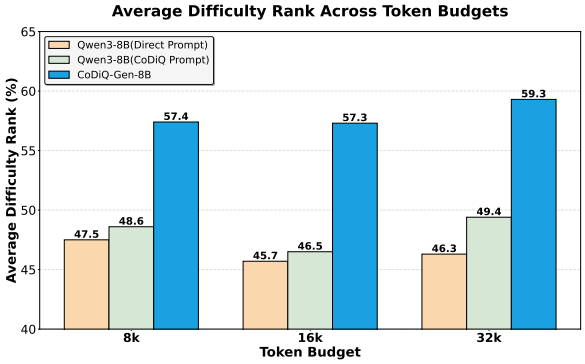


Figure 3: **Difficulty under cumulative token budgets** (8k/16k/32k) on CoDiQ-Bench.

in later rounds, which we attribute to approaching either the model's own generation capacity or the evaluator's limit. This dynamic view reinforces our earlier metric validation (§4.2.2) from a per-model perspective: token volume tracks difficulty within a single trajectory, not just across models.

**Solvability Scaling.** Solvability degrades as difficulty rises (Figure 5), exposing a fundamental depth-vs-validity trade-off. Three findings emerge:

- **Robustness of SOTA models.** Flagship models (e.g., GLM-4.6) maintain high solvability across difficulty levels, exhibiting balanced generation–verification.
- **Over-reasoning pitfall.** Smaller models suffer validity collapse under CoDiQ, generating complexity beyond their reasoning capacity.
- **Efficacy of RL alignment.** CoDiQ-Gen-8B breaks this degradation pattern, decoupling difficulty scaling from validity loss.

## 5 CoDiQ-Corpus

To comprehensively assess the value of this corpus, we conduct a multi-dimensional evaluation along five axes: *difficulty* (§5.1), *solvability* (§5.2), *diversity* (§5.3), *realness* (§5.4), and *training effectiveness* (§5.5). The first four axes characterize the corpus as a static artifact—how hard, well-posed, broad, and naturally distributed it is—while the last axis measures whether these intrinsic properties translate into downstream reasoning gains.

### 5.1 Difficulty

We sample 300 questions from CoDiQ-Corpus and from each baseline (Ding et al., 2025b; Jia LI et al., 2024; Hendrycks et al., 2021; Chou et al., 2025; Jain et al., 2024; Pei et al., 2025; Li et al., 2022; Veeraboina, 2023; Zhao et al., 2025) and score them with the ranking methodology in Section 4.1 together with human annotation. Our automated proxies correlate strongly with human judgement: DR-LLM achieves Pearson and Spearman correlations of 0.985 and 0.964 with DR-Human, while DR-VN reaches 0.893 and 0.697. These coefficients support DR-LLM and DR-VN as reliable scalable signals of intrinsic difficulty rather than artifacts of the same family of LLMs.

To complement these proxies with an estimator-free check, we additionally report the average solver tokens that Qwen3-32B consumes per question, motivated by recent evidence that solver-token usage tracks problem difficulty and reasoning depth (de Varda et al., 2025; Lee et al., 2025). As shown in Table 2, CoDiQ-Corpus dominates every column (DR-LLM, DR-VN, DR-Human and solver tokens), surpassing both human-curated competition corpora and recent synthetic baselines.

Table 2: **Datasets Difficulty Comparison.** The best, the second-best and the third-best scores for each indicator are shown in $\boxed{box}$, **bold** and underlined, respectively.

| Dataset | Response Tokens ↑ | DR-LLM | DR-VN | DR-Human | DR(AVG) |
|---|---|---|---|---|---|
| ***Baselines (Human-Curated)*** | | | | | |
| APPS | 12,122 | 39.1 | 44.8 | 38.8 | 40.9 |
| LiveCodeBench | 11,914 | 46.3 | 45.2 | 43.8 | 45.1 |
| Code-Contests | 14,981 | 56.7 | 43.7 | 54.0 | 51.5 |
| AIME (1983–2024) | 14,197 | **59.0** | 46.6 | **58.0** | **54.5** |
| ***Baselines (AI-Generated)*** | | | | | |
| ScaleQuest-Math | 5,592 | 15.0 | 30.5 | 7.2 | 17.6 |
| NuminaMath-1.5 | 8,042 | 32.5 | 31.3 | 30.4 | 31.4 |
| AutoCodeBench | 10,013 | 43.9 | 49.5 | 30.0 | 41.1 |
| ScaleDiff-Math | 14,341 | 50.0 | 52.5 | 45.2 | 49.2 |
| PromptCoT-2.0 | **16,817** | 64.8 | **68.3** | **68.4** | **67.2** |
| ***CoDiQ-Corpus (Ours)*** | | | | | |
| CoDiQ-Corpus | $\boxed{17{,}955}$ | $\boxed{91.5}$ | $\boxed{87.4}$ | $\boxed{95.0}$ | $\boxed{91.3}$ |

### 5.2 Solvability and Quality

We assess solvability at two complementary scales. (1) *Human evaluation.* Three experts (App. O) evaluated $N = 200$ stratified samples on Clarity, Completeness, and Reasoning Validity (App. M), achieving Fleiss' $\kappa = 0.76$. The accepted and rejected sets attain 82% precision and 90% NPV, respectively. False negatives highlight the *Verifier Paradox*: valid but overly complex instances misflagged as unsolvable.(2) *Automatic probe across datasets.* We further run a Qwen3-8B solvability classifier on 300 questions sampled from each dataset. Because CoDiQ-Corpus is verified end-to-end during construction, it passes this probe by design; the recent synthetic baselines, in contrast, span a wide range from ∼68% to ∼98%, with several falling well below the typical solvability rate

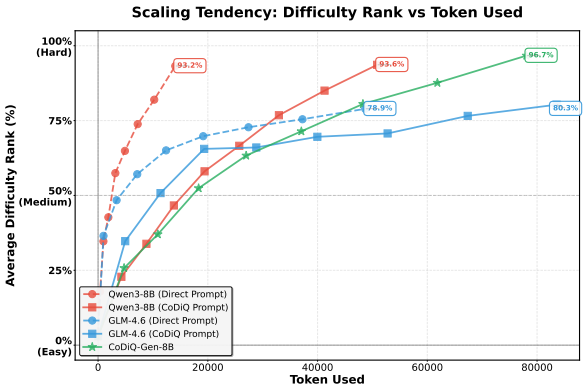


Figure 4: **Difficulty scaling.** Normalized DR-AVG across 8 rounds.

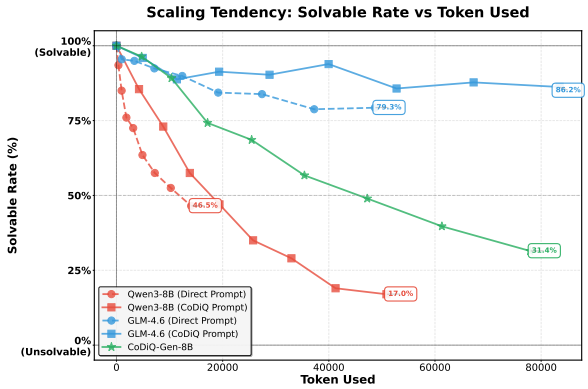


Figure 5: **Solvability scaling.** Solvable rate across 8 rounds.

of human-curated competition data. Per-dataset numbers are reported in Appendix J (Table 13).

### 5.3 Diversity

We quantify corpus diversity at the second-level domain granularity (e.g., *Algorithms* → *Dynamic Programming*) using the effective number of categories, defined as $\text{effN} = \exp(H)$, where $H$ denotes the Shannon entropy of the multi-label domain distribution, with labels assigned by Qwen3-32B following the prompt in Appendix T. Under this formulation, CoDiQ-Corpus exhibits exceptional taxonomic breadth, achieving an effN of 148.0. This represents a roughly threefold improvement over the prevailing standard established by recent mathematics and coding reasoning corpora. Comprehensive comparative statistics—encompassing richness (`uniq`, `top`), evenness (Pielou `even`, Gini–Simpson `simp`), and coverage (`cov`)—are deferred to Appendix H (Table 11).

### 5.4 Realness

A corpus may exhibit high difficulty, solvability, and taxonomic breadth, yet remain practically unrealistic if its pipeline generates templated phrasings or category profiles absent from actual examinations. To formalize this, we introduce a *realness* axis, defined by distributional alignment to authentic, human-curated corpora rather than by the intrinsic properties of the synthetic data.

**Setup.** We extract the multi-label domain annotation for each question to formulate a probability distribution over second-level categories (matching the granularity in §5.3). For any two datasets $A$ and $B$ with distributions $p$ and $q$, we measure alignment via the Jensen–Shannon divergence: $\text{JSD}(p\|q) = \frac{1}{2}\text{KL}(p\|m) + \frac{1}{2}\text{KL}(q\|m)$, where $m = (p+q)/2$. We adopt JSD over KL divergence because it is symmetric, bounded in $[0, \log 2]$, and robust to mismatched supports.

**Results.** The full JSD matrix between six synthetic datasets and four real corpora (AIME, LiveCodeBench, APPS, Code-Contests) is shown in Figure 8 (Appendix I). Among code-oriented synthetic datasets, CoDiQ-Corpus attains the smallest JSD against APPS and Code-Contests—roughly 2× tighter than AutoCodeBench. Math-oriented synthetic datasets (NuminaMath-1.5, ScaleDiff-Math, ScaleQuest-Math) correctly cluster near AIME and far from code corpora, cross-validating that the metric captures genuine domain semantics. Figure 6 visualizes the underlying coverage profiles: CoDiQ-Corpus broadens the long tail without abandoning the dominant categories that anchor real competition data.

**Discussion.** Realness serves as a necessary counterbalance to unconstrained diversification. While aggressive data synthesis can artificially inflate pure diversity metrics, it frequently risks catastrophic drift from target distributions (as observed in alternative baselines in Figure 8). The JSD analysis directly substantiates that CoDiQ's iterative mutation process avoids injecting pipeline-specific structural artifacts. Instead of diverging into unrealistic semantic spaces, CoDiQ synthesizes a corpus whose taxonomic footprint closely mirrors that of elite human competitions.

### 5.5 Training Effectiveness Validation

**Reinforcement Learning Validation via Curriculum.** A distinct advantage of CoDiQ is its inher-

[2]Due to the limited number of samples in this benchmark, we report the average score across 24 runs.

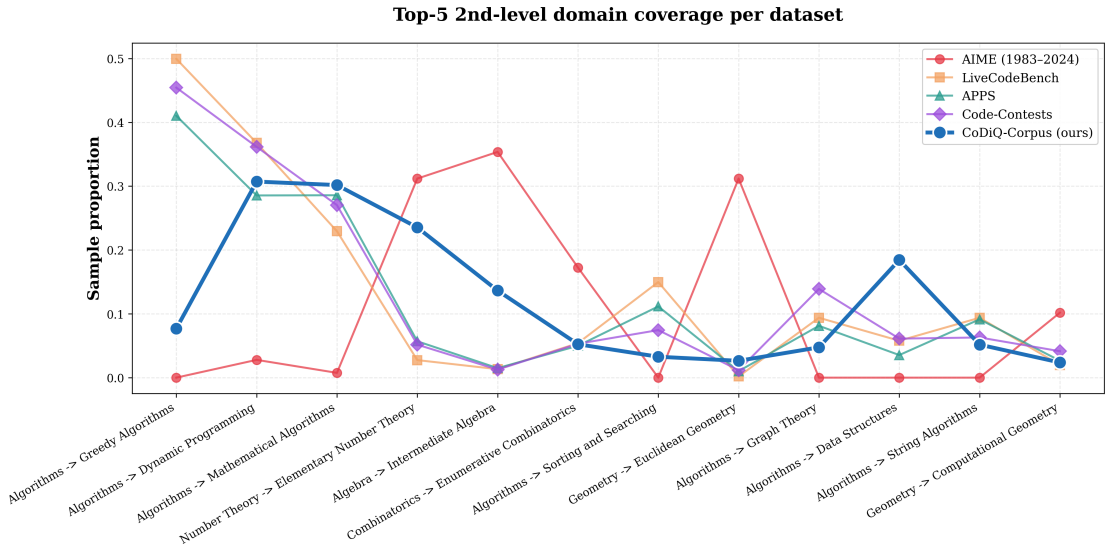


Figure 6: **Top-tier domain coverage profiles.** While authentic contest corpora (e.g., APPS, Code-Contests, LiveCodeBench) exhibit sharply peaked distributions, CoDiQ-Corpus achieves a more balanced, broader profile that successfully preserves the dominant categories anchoring real-world evaluations.

Table 3: **Curriculum Learning Model Performance Comparison.** The best, the second-best and the third-best scores for each indicator are shown in $\boxed{box}$, **bold** and underlined, respectively.

| Model | MATH-500 | AIME 2024[2] | MBPP | LiveCodeBench v5 |
|---|---|---|---|---|
| *Base Models* | | | | |
| Qwen3-4B | 94.4 | 63.1 | 66.4 | 53.6 |
| Qwen3-8B | 96.2 | 70.0 | 69.2 | 56.9 |
| *Baselines* | | | | |
| Qwen3-RL-4B | 95.2 | 64.3 | 66.4 | 52.0 |
| Qwen3-ScaleQuest-4B | 95.8 | 63.1 | 66.8 | 51.5 |
| Qwen3-ScaleDiff-4B | 96.0 | 66.1 | 65.2 | 54.0 |
| Qwen3-PromptCoT2-4B | 94.8 | 70.0 | 67.0 | 53.6 |
| Qwen3-RL-8B | 96.4 | 68.4 | 68.0 | 57.0 |
| Qwen3-ScaleQuest-8B | 96.6 | 69.4 | 68.6 | 56.0 |
| Qwen3-ScaleDiff-8B | **96.8** | 71.0 | 69.8 | 57.0 |
| Qwen3-PromptCoT2-8B | 96.2 | 71.6 | 69.2 | 57.6 |
| *Curriculum Learning Models (Ours)* | | | | |
| CoDiQ-L1-4B | 96.0 | 65.0 | 65.2 | 53.2 |
| CoDiQ-L2-4B | 94.8 | 66.7 | 66.6 | 54.8 |
| CoDiQ-L3-4B | 96.0 | 70.6 | 67.6 | 54.2 |
| CoDiQ-L1-8B | 95.4 | **71.4** | 68.8 | 57.2 |
| CoDiQ-L2-8B | 96.6 | 69.7 | **69.6** | 57.2 |
| CoDiQ-L3-8B | $\boxed{97.0}$ | $\boxed{73.2}$ | $\boxed{70.2}$ | $\boxed{58.0}$ |

ent controllability: by modulating the generation budget, the pipeline naturally yields question sequences of progressive complexity. This directly facilitates a curriculum learning strategy (Bengio et al., 2009) that tightly synchronizes with the model's evolving capabilities during post-training. We operationalize this via a multi-stage Reinforcement Learning paradigm, sequentially training models across data strata of escalating difficulty (reward formulations detailed in Appendix N). When evaluated against the base model, standard RL applied to un-stratified data, and baselines trained on equivalently sized subsets sampled from contemporary synthetic corpora (Zhao et al., 2025; Pei et al., 2025; Ding et al., 2024), our curriculum-aligned models consistently establish superior performance across premier mathematical reasoning and competitive programming benchmarks (Table 3).

**When does each difficulty level help?** The L1→L3 stratification reveals that the optimal training stage is benchmark-dependent. Simpler benchmarks (e.g., MBPP, MATH-500) saturate their performance gains at low-to-mid stages, whereas frontier evaluations (AIME 2024, LiveCodeBench v5) critically depend on L3 data. This aligns with the principle that training difficulty should match or modestly exceed the target distribution—a trade-off CoDiQ makes explicitly controllable rather than incidental.

## 6 Conclusion

We presented *CoDiQ*, a principled framework for synthesizing verifiable, high-difficulty reasoning problems at scale. By addressing the generator capacity ceiling through test-time scaling and mitigating "fake hard" instances via a hybrid verification pipeline, we successfully trained the **CoDiQ-Generator** using reinforcement learning. The resulting **CoDiQ-Corpus** features budget-driven difficulty stratification, and its effective application in curriculum learning validates the method's superiority. We open-source our pipeline to facilitate future research into scaling laws and automated curriculum learning.

## Limitations

Our scope is currently restricted to English math and code tasks, and the verification cost limits real-time use. Long-CoT validation primarily uses the Qwen3 family, and the human study spans three PhD annotators on 200 stratified samples (Fleiss' $\kappa$=0.76); we partially compensate via an automated solvability probe (§5.2) and solver-token measure (§5.1). The token–difficulty relation we exploit holds under a fixed scaling protocol and an evaluator within its competence range; beyond these conditions, the pipeline faces the *Verifier Paradox*—valid problems exceeding the verifier's reasoning horizon are at risk of being discarded—which we leave to future work on scalable oversight.

## References


Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Zhuofan Chen, Jiyuan He, Yichi Zhang, Xing Hu, Haoxing Wen, Jun Bai, and Wenge Rong. 2025. Cogatom: From cognitive atoms to olympiad-level mathematical reasoning in large language models. *Preprint*, arXiv:2509.17318.

ChilleD. 2024. Svamp dataset.

Jason Chou, Ao Liu, Yuchi Deng, Zhiying Zeng, Tao Zhang, Haotian Zhu, Jianwei Cai, Yue Mao, Chenchen Zhang, Lingyun Tan, and 1 others. 2025. Autocodebench: Large language models are automatic code benchmark generators. *arXiv preprint arXiv:2508.09101*.

Bryan R Christ, Jonathan Kropko, and Thomas Hartvigsen. 2024. Mathwell: Generating educational math word problems using teacher annotations. *Preprint*, arXiv:2402.15861.

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

Andrea Gregor de Varda, Ferdinando Pio D'Elia, Hope Kean, Andrew Lampinen, and Evelina Fedorenko. 2025. The cost of thinking is similar between large reasoning models and humans. *Proceedings of the National Academy of Sciences*, 122(47):e2520077122.

Yuyang Ding, Xinyu Shi, Juntao Li, Qiaoming Zhu, Min Zhang, and 1 others. 2024. Unleashing reasoning capability of llms via scalable question synthesis from scratch. *arXiv preprint arXiv:2410.18693*.

Yuyang Ding, Xinyu Shi, Xiaobo Liang, Juntao Li, Zhaopeng Tu, Qiaoming Zhu, and Min Zhang. 2025a. Unleashing llm reasoning capability via scalable question synthesis from scratch. *Preprint*, arXiv:2410.18693.

Yuyang Ding, Xinyu Shi, Xiaobo Liang, Juntao Li, Zhaopeng Tu, Qiaoming Zhu, and Min Zhang. 2025b. Unleashing llm reasoning capability via scalable question synthesis from scratch. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13414–13438.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, and 37 others. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *Preprint*, arXiv:2406.12793.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *Preprint*, arXiv:2402.14008.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with apps. *NeurIPS*.

hiyouga. 2025. Math12k dataset.

Hanxu Hu, Xingxing Zhang, Jannis Vamvas, Rico Sennrich, and Furu Wei. 2025. Quest: Incentivizing llms to generate difficult problems. *Preprint*, arXiv:2510.17715.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free eval-

uation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Lewis Tunstall Jia LI, Edward Beeching and 1 others. 2024. Numinamath tir.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. Ds-1000: A natural and reliable benchmark for data science code generation. *ArXiv*, abs/2211.11501.

Sunbowen Lee, Qingyu Yin, Chak Tou Leong, Jialiang Zhang, Yicheng Gong, Shiwen Ni, Min Yang, and Xiaoyu Shen. 2025. Probing the difficulty perception mechanism of large language models. *arXiv preprint arXiv:2510.05969*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Haoxiong Liu, Yifan Zhang, Yifan Luo, and Andrew Chi-Chih Yao. 2024. Augmenting math word problems via iterative question composing. *Preprint*, arXiv:2401.09003.

Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Clara Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu H. Trinh, Quoc V. Le, and Junehyuk Jung. 2025. Towards robust mathematical reasoning. *Preprint*, arXiv:2511.01846.

Chaitanya Manem, Pratik Prabhanjan Brahma, Prakamya Mishra, Zicheng Liu, and Emad Barsoum. 2025. Sand-math: Using llms to generate novel, difficult and useful mathematics questions and answers. *arXiv preprint arXiv:2507.20527*.

Qizhi Pei, Zhuoshi Pan, Honglin Lin, Xin Gao, Yu Li, Zinan Tang, Conghui He, Rui Yan, and Lijun Wu. 2025. Scalediff: Scaling difficult problems for advanced mathematical reasoning. *arXiv preprint arXiv:2509.21070*.

Bytedance Seed. 2025. Seed1. 8 model card: Towards generalized real-world agency.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.

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

Haoxiang Sun, Yingqian Min, Zhipeng Chen, Wayne Xin Zhao, Zheng Liu, Zhongyuan Wang, Lei Fang, and Ji-Rong Wen. 2025. Challenging the boundaries of reasoning: An olympiad-level math benchmark for large language models. *Preprint*, arXiv:2503.21380.

Hemish Veeraboina. 2023. Aime problem set 1983-2024.

Shengbo Wang, Mingwei Liu, Zike Li, Anji Li, Yanlin Wang, Xin Peng, and Zibin Zheng. 2025. Evolmatheval: Towards evolvable benchmarks for mathematical reasoning via evolutionary testing. *Preprint*, arXiv:2508.13003.

Yunhui Xia, Wei Shen, Yan Wang, Jason Klein Liu, Huifeng Sun, Siyue Wu, Jian Hu, and Xiaolong Xu. 2025. Leetcodedataset: A temporal dataset for robust evaluation and efficient training of code llms. *arXiv preprint arXiv:2504.14655*.

Roy Xie, Chengxuan Huang, Junlin Wang, and Bhuwan Dhingra. 2024. Adversarial math word problem generation. *Preprint*, arXiv:2402.17916.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

yimingzhang. 2025. asdiv dataset.

Charlie Zhang, Graham Neubig, and Xiang Yue. 2025. On the interplay of pre-training, mid-training, and rl on reasoning language models. *arXiv preprint arXiv:2512.07783*.

Xueliang Zhao, Wei Wu, Jian Guan, Zhuocheng Gong, and Lingpeng Kong. 2025. Promptcot 2.0: Scaling prompt synthesis for large language model reasoning. *arXiv preprint arXiv:2509.19894*.

Xinyue Zheng, Haowei Lin, Shaofei Cai, Zilong Zheng, and Yitao Liang. 2025. Unicode: A framework for generating high quality competitive coding problems. *Preprint*, arXiv:2510.17868.

Shang Zhou, Zihan Zheng, Kaiyuan Liu, Zeyu Shen, Zerui Cheng, Zexing Chen, Hansen He, Jianzhu Yao, Huanzhi Mao, Qiuyang Mang, Tianfu Fu, Beichen Li, Dongruixuan Li, Wenhao Chai, Zhuang Liu, Aleksandra Korolova, Peter Henderson, Natasha Jaques, Pramod Viswanath, and 2 others. 2025. Autocode: Llms as problem setters for competitive programming. *Preprint*, arXiv:2510.12803.

Yubo Zhu, Dongrui Liu, Zecheng Lin, Wei Tong, Sheng Zhong, and Jing Shao. 2025. The llm already knows:

Estimating llm-perceived question difficulty via hidden representations. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1160–1176.

# A CoDiQ-Bench

To systematically evaluate the question-generation capability of LRMs, we construct CoDiQ-Bench, a curated 200-case set (Table 4). For coding tasks, we randomly sample 50 cases each from CodeAlpaca_20K (general programming) and LeetCode-Dataset (algorithmic challenges). For mathematical reasoning, we sample 50 cases each from GSM8K and MATH12K. We intentionally start from simple seeds so the evaluated metric reflects the generator's ability to scale difficulty rather than the seed's intrinsic hardness; selection criteria regarding solvability and quality are detailed in Appendix F.

Table 4: Dataset statistics of CoDiQ-Bench.

| Statistics | Number |
|---|---|
| **#Questions** | 200 |
| - *math* | 100 |
| - *code* | 100 |
| **Question Tokens Length** | |
| - *max/min/avg* | 726/9/128.2 |

# B Upper Bound of Difficulty Generation

This ablation removes the solvability verifier from the CoDiQ Pipeline, exposing the theoretical upper bound that each backbone can reach when validity constraints are dropped. The CoDiQ Prompt elevates the ceiling for every Qwen3 backbone, and CoDiQ-Gen-8B—at fewer parameters—exceeds Qwen3-14B, indicating that RL alignment unlocks complex logical structures beyond what scale alone provides.

Table 5: **Upper-bound difficulty generation on CoDiQ-Bench** (no solvability filter, 8 rounds). Best, second, and third in each column are marked $\boxed{box}$, **bold**, underlined.

| Model | Rounds | Tokens | DR-LLM | DR-VN | DR(AVG) |
|---|---|---|---|---|---|
| ***Direct Prompt*** | | | | | |
| Qwen3-8B | $\boxed{6.0}$ | 2439.7 | 33.5 | 39.1 | 36.3 |
| Qwen3-14B | 5.6 | 4927.4 | 45.6 | 55.6 | 50.6 |
| Qwen3-32B | 5.7 | 4124.9 | **65.3** | 47.5 | 56.4 |
| ***CoDiQ Prompt (ours)*** | | | | | |
| Qwen3-8B | 5.8 | 7282.2 | 53.5 | 53.3 | 53.4 |
| Qwen3-14B | 5.6 | 9590.2 | 58.6 | 63.1 | 60.9 |
| Qwen3-32B | 5.7 | **9762.4** | $\boxed{74.6}$ | **65.0** | $\boxed{69.8}$ |
| ***CoDiQ Generator (ours)*** | | | | | |
| CoDiQ-Gen-8B | **5.9** | $\boxed{12591.6}$ | 52.6 | $\boxed{72.2}$ | **62.4** |

# C CoDiQ Pipeline: Case Study

We demonstrate the CoDiQ pipeline through a complete workflow from an initial easy problem to iterative difficulty escalation, illustrating both successful upgrades and failure modes. Each generated problem undergoes solvability verification (Appendix Q) and difficulty assessment (Appendix P).

## C.1 Initial Problem

**Problem Statement:** Count subsequences with an odd sum from array `nums`, returning the result modulo $10^9 + 7$.

**Example:** For `nums = [1,1,1]`, the answer is 4 (subsequences from positions: $\{0\}, \{1\}, \{2\}, \{0, 1, 2\}$, all with odd sums).

**Solution:** Simple DP tracking sum parity in $O(n)$ time.

## C.2 Round 1: Controlled Escalation

### C.2.1 Upgraded Problem

Count non-empty subsequences satisfying three simultaneous conditions:

1. Sum is **odd**
2. Length is **even**
3. Sum $\bmod 3 = 1$

**Difficulty Enhancement:** The upgrade introduces multi-dimensional state tracking, expanding the DP state space from 2 (sum parity) to $2 \times 2 \times 3 = 12$ states (sum parity, length parity, sum mod 3).

### C.2.2 Verification

- **Solvability Score:** 0.90
- **Time Complexity:** $O(12n) \approx 1.2 \times 10^6$ operations for $n = 10^5$ (feasible)
- **Solution Density:** $\sim 8\%$ of subsequences satisfy all conditions (non-trivial)
- **Solvability:** PASS
- **Difficulty:** INCREASED

## C.3 Round 2: Further Escalation

### C.3.1 Upgraded Problem

Count subsequences satisfying five conditions:

1. Sum is **odd**
2. Length is **even**

3. Sum $\bmod 3 = 1$

4. Sum $\bmod 5 = 2$

5. Length $\bmod 4 = 2$

**Mathematical Simplification:** By the Chinese Remainder Theorem (CRT), conditions 1, 3, and 4 can be unified:

$$\text{sum} \equiv 1 \pmod 2, \quad \text{sum} \equiv 1 \pmod 3, \quad \text{sum} \equiv 2 \pmod 5 \quad \Rightarrow \quad \text{sum} \equiv 7 \pmod{30}$$

The effective state space becomes $30 \times 4 = 120$ states.

#### C.3.2 Verification

- **Solvability Score:** 0.85
- **Time Complexity:** $O(120n) \approx 1.2 \times 10^7$ operations for $n = 10^5$ (acceptable)
- **Solution Density:** $\sim 0.83\%$ (still non-trivial)
- **Solvability:** PASS
- **Difficulty:** UNCHANGED

### C.4 Round 3: Over-Escalation Failure

#### C.4.1 Upgraded Problem

Count subsequences satisfying six conditions:

1. Sum is **odd**
2. Sum $\bmod 3 = 1$
3. Sum $\bmod 5 = 2$
4. Sum $\bmod 7 = 4$
5. Sum $\bmod 11 = 6$
6. Length $\bmod 8 = 2$ (which ensures even length)

By CRT, conditions 1–5 unify to sum $\equiv c \pmod{2310}$ for some constant $c$, yielding a state space of $2310 \times 8 = 18{,}480$ states.

#### C.4.2 Verification

- **Solvability Score:** 0.65
- **Solvability:** FAIL
- **Difficulty:** INCREASED

**Failure Analysis: 1. Computational Infeasibility**

- Time complexity: $O(18{,}480n) \approx 1.8 \times 10^9$ operations for $n = 10^5$
- Exceeds practical competitive programming limits (typically $\sim 10^8$–$10^9$ operations within time constraints)

**2. Solution Space Collapse** (Critical Issue)

- While constraints are mathematically consistent via CRT, they create an extremely sparse solution space
- Probability that a random subsequence satisfies all conditions: $\approx \frac{1}{2310} \times \frac{1}{8} = \frac{1}{18{,}480}$
- Expected number of valid subsequences: $\frac{2^n}{18{,}480}$
- For $n \leq 14$: $\frac{2^{14}}{18{,}480} \approx 0.89 < 1$
- **Practical impact:** For typical inputs with small to moderate $n$, the answer is almost always 0, making the problem vacuously trivial

**Pipeline Termination:** The pipeline correctly terminates at Round 3, discarding $Q_3$ and outputting $\{Q_0, Q_1, Q_2\}$. Despite the increased theoretical difficulty, the problem becomes unsolvable due to computational infeasibility and solution space collapse, demonstrating the effectiveness of solvability verification in preventing quality degradation.

## D CoDiQ Pipeline: Failure Type Analysis

To systematically understand the failure modes of the CoDiQ pipeline, we conduct a comprehensive clustering analysis on the collected failure reasons. Our analysis follows a three-stage hierarchical approach: initial K-means clustering, keyword extraction, and hierarchical merging with manual refinement.

### D.1 Clustering Methodology

**Stage 1: K-means Pre-clustering.** We first apply K-means clustering to the failure reason descriptions to obtain an initial partitioning of the data. This pre-clustering step reduces computational complexity and provides a coarse-grained grouping of similar failure patterns.

**Stage 2: Keyword Extraction.** For each cluster obtained from K-means, we extract representative keywords using TF-IDF weighting. These

keywords serve as semantic signatures that characterize the dominant failure patterns within each cluster, facilitating interpretability and subsequent hierarchical analysis.

**Stage 3: Hierarchical Clustering and Manual Refinement.** We then perform hierarchical clustering on the cluster centroids, leveraging the extracted keywords to compute semantic similarity. Finally, we manually merge related clusters and consolidate small clusters (containing fewer than a predefined threshold of samples) with their semantically nearest neighbors. This hybrid approach balances computational efficiency with semantic coherence.

### D.2 Failure Category Distribution

Table 6 categorizes the identified failure modes. The analysis reveals two dominant distinct failure dynamics: **validity breaches** (Unsolvable) and **complexity degradation** (Difficulty Decreased).

**Unsolvable Scenarios.** The majority of pipeline failures stem from fundamental deficits in problem formulation. Specifically, *Definition & Information Missing* combined with *Constraints & Logic Conflicts* collectively account for the lion's share of unsolvable cases. This indicates that the primary challenge lies not in parsing or formatting (which constitute a negligible fraction), but in the model's capacity to maintain semantic consistency and logical completeness during generation.

**Difficulty Preservation.** A critical observation is the prevalence of the *Difficulty Decreased* category ($N = 12,916$). In these instances, the generated problems remain solvable but fail to meet the intended cognitive demand. The high frequency of *Constraint Simplification* and *Numerical Range Reduction* suggests a model tendency towards "safe" or simplified generative paths, inadvertently pruning the solution space or removing key logical hurdles required for high-quality mathematical reasoning.

Table 6: Distribution of Failure Cases in CoDiQ Pipeline

| Failure Type | Failure Subtype | Count |
|---|---|---|
| Unsolvable | Definition & Information Missing | 8,630 |
| | Constraints & Logic Conflicts | 8,142 |
| | Computationally Infeasible | 1,948 |
| | Implementation Details Missing | 1,926 |
| | No Suitable Algorithm/Structure | 1,285 |
| | Overly Complex | 1,104 |
| | Requires Specific Capability | 726 |
| | Parsing & Rule Ambiguity | 611 |
| | Number-Theoretic Constraints | 566 |
| | Other | 1,184 |
| Difficulty Decreased | Constraint Simplification | 3,245 |
| | Numerical Range Reduction | 2,890 |
| | Key Condition Removal | 2,654 |
| | Solution Space Narrowing | 1,987 |
| | Structural Simplification | 1,456 |
| | Other | 684 |

## E ValueNetwork Training Detail

### E.1 Dataset split

We compiled a labeled dataset by selecting samples from standard benchmarks (Cobbe et al., 2021; Xia et al., 2025) for the *Easy* class and competition-level datasets (Jia LI et al., 2024; Hendrycks et al., 2021) for the *Hard* class. We maintained an easy-to-hard ratio of 2:3 to prioritize the identification of challenging samples. We partition the compiled dataset into an 85:15 train-test split to ensure robust evaluation.

### E.2 Training Data

#### E.2.1 Input Features

For training data, we employ QWEN3-8B (in non-thinking mode) to capture generation dynamics. We define a sampling window from the last token of the question extending to $\min(4096, L_r)$ generated tokens.

Within this window, we apply a *quadratic sampling strategy* to select $K$ hidden states ($K = 10$ for windows $> 1024$, else $K = 8$) at positions:

$$p_i = \left\lfloor |W| \cdot \left( \frac{i}{K-1} \right)^2 \right\rfloor, \quad i = 0, 1, \ldots, K-1. \tag{3}$$

#### E.2.2 Output Labels

This strategy allocates higher sampling density to the onset of generation, capturing critical information for establishing the reasoning path. To mitigate stochasticity, scores are averaged over 5 independent passes.

For each question, we generate a response using QWEN3-8B and assign a binary label $y \in \{0, 1\}$ based on the final answer's correctness. The input features $x$ are extracted via the quadratic sampling strategy (Eq. 3) applied to the first 4096 tokens.

### E.3 Network Architecture

The Value Network is implemented as a lightweight Multi-Layer Perceptron (MLP) designed to project

Table 7: Hyperparameter settings.

| HYPERPARAMETER | VALUE |
|---|---|
| INPUT DIM ($d_{in}$) | 4096 |
| HIDDEN DIM | 512 |
| BATCH SIZE | 512 |
| LEARNING RATE | $1 \times 10^{-4}$ |
| WEIGHT DECAY | $1 \times 10^{-2}$ |
| DROPOUT | 0.3 |
| OPTIMIZER | ADAMW |
| SCHEDULER | STEPLR ($\gamma = 0.8$) |
| MAX EPOCHS | 30 |
| SPLIT | 85% / 15% |

Table 8: Performance on held-out test set.

| METRIC | VALUE | METRIC | VALUE |
|---|---|---|---|
| ACCURACY | 72.52% | PRECISION | 54.21% |
| RECALL | 95.62% | F1 SCORE | 69.20% |
| **ROC-AUC** | **84.84%** | PR-AUC | 65.77% |

high-dimensional hidden states ($d_{in} = 4096$) to a scalar correctness score. The architecture consists of an initial projection layer, Layer Normalization, GELU activation, and a final regression head.

This setup allows the network to minimize the discrepancy with the correctness label $y$ via a weighted binary cross-entropy objective, effectively estimating the likelihood of a successful generation solely from the reasoning dynamics captured in the early stages.

### E.4 Training Configuration

The model is trained using the AdamW optimizer with a step learning rate scheduler. To address class imbalance, we apply a positive class weight in the loss function, dynamically calculated as the ratio of negative to positive samples. Complete hyperparameter settings are listed in Table 7.

### E.5 Performance Evaluation

We evaluate the trained Value Network on the held-out test set (15% split). As shown in Table 8, the model achieves an **ROC-AUC of 84.84%**, demonstrating robust discriminative power in distinguishing correct reasoning paths from incorrect ones despite the challenging nature of the dataset.

It is worth noting that our training strategy prioritizes identifying all potential correct answers. This is reflected in the **high Recall of 95.62%**, which ensures that the Value Network successfully preserves valid reasoning paths. While this focus on coverage results in a moderate Precision (54.21%) due to the trade-off inherent in class-weighted training, the high ROC-AUC indicates that the predicted scores effectively rank correct generations higher, making the model reliable for difficulty estimation and filtering.

## F CoDiQ-Bench Selection Criteria

To ensure the quality and reliability of our benchmark, we establish three primary criteria for data selection:

**Solvability:** We verify that each problem is well-defined and admits at least one valid solution, ensuring the benchmark's validity and fairness.

**Difficulty Level:** We assess whether the difficulty level is appropriate for the intended evaluation purpose, maintaining a balanced distribution across different complexity levels.

**Quality Assessment:** We conduct rigorous quality checks to ensure that all selected problems meet acceptable standards in terms of clarity, correctness, and relevance.

## G Statistics of CoDiQ-Corpus

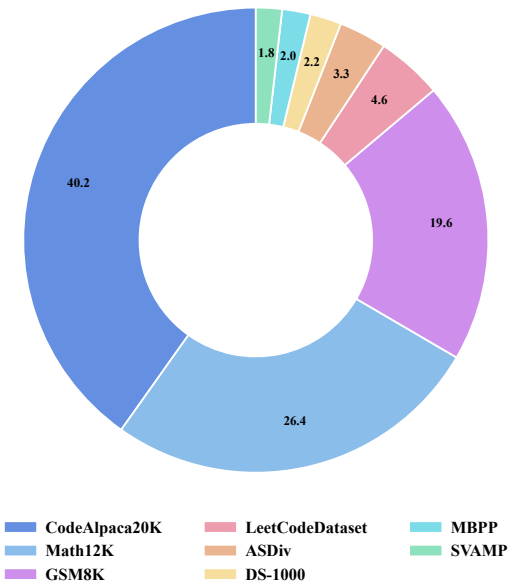


Figure 7: Distribution of CoDiQ-Corpus Dataset.

We employ CoDiQ-Gen-8B following the CoDiQ Pipeline (Section 3.3) to transform eight diverse mathematical and programming datasets into the more challenging CoDiQ-Corpus, which comprises approximately 44,453 question sequences with progressive difficulty from easy to hard. The detailed distribution is presented in Table 9.

Table 9: Dataset statistics of CoDiQ-Corpus.

| Dataset | Question Tokens Length | | | AVG Round | Category | Sequences |
|---|---|---|---|---|---|---|
| | Minimum | Maximum | Average | | | |
| Math12K (hiyouga, 2025) | 38 | 7,829 | 995.4 | 4.7 | Math | 11,764 |
| GSM8K (Cobbe et al., 2021) | 52 | 6,896 | 1,093.7 | 4.5 | Math | 8,685 |
| SVAMP (ChilleD, 2024) | 172 | 3,992 | 971.3 | 3.3 | Math | 804 |
| ASDiv (yimingzhang, 2025) | 55 | 4,703 | 1,013.1 | 4.7 | Math | 1,480 |
| CodeAlpaca20K (Chaudhary, 2023) | 70 | 7,174 | 1,106.1 | 3.8 | Code | 17,845 |
| LeetCodeDataset (Xia et al., 2025) | 254 | 4,365 | 1,281.0 | 3.8 | Code | 2,027 |
| MBPP (Austin et al., 2021) | 52 | 3,440 | 1,000.4 | 3.4 | Code | 876 |
| DS-1000 (Lai et al., 2022) | 192 | 4,138 | 1,240.7 | 3.2 | Code | 972 |
| Total | 38 | 7,829 | 1,073.0 | 4.2 | - | 44,453 |

**Second-level domain composition.** To verify that CoDiQ-Corpus inherits the long-tail composition of real competition corpora rather than being concentrated on a few synthetic templates, we report the top-20 second-level domains in Table 10. The corpus covers more than 600 distinct subcategories with the long tail anchored on *Number Theory* $\rightarrow$ *Elementary Number Theory*, *Algorithms* $\rightarrow$ *Mathematical Algorithms*, and *Algorithms* $\rightarrow$ *Dynamic Programming*—the same dominant sub-categories observed in human-curated competition data, which explains the low JSD against APPS / Code-Contests reported in Section 5.4.

Table 10: Top-20 second-level domains in CoDiQ-Corpus.

| Top-level | Sub-category | Count |
|---|---|---|
| Number Theory | Elementary Number Theory | 19,826 |
| Algorithms | Mathematical Algorithms | 15,433 |
| Algorithms | Dynamic Programming | 14,040 |
| Algebra | Intermediate Algebra | 10,798 |
| Algorithms | Data Structures | 9,999 |
| Computer Science | Algorithms | 4,765 |
| Mathematics | Number Theory | 4,319 |
| Algorithms | Greedy Algorithms | 3,417 |
| Mathematics | Algebra | 3,054 |
| Algorithms | Graph Theory | 2,565 |
| Algorithms | String Algorithms | 2,417 |
| Combinatorics | Enumerative Combinatorics | 2,392 |
| Geometry | Euclidean Geometry | 1,754 |
| Algebra | Linear Algebra | 1,503 |
| Algorithms | Sorting and Searching | 1,454 |
| Algorithms | Lazy Propagation | 1,366 |
| Algorithms | Advanced Data Structures | 1,321 |
| Mathematics | Combinatorics | 1,158 |
| Mathematics | Calculus | 1,135 |
| Geometry | Computational Geometry | 1,122 |

## H Detailed Diversity Statistics

To complement the single-scalar effN reported in Section 5.3, we report a full panel of diversity statistics in Table 11, computed at the second-level domain granularity over 300 sampled questions per dataset. Column definitions are summarized in Table 12.

## I Distributional Alignment to Real Corpora

This appendix complements Section 5.4 with the full Jensen–Shannon divergence (JSD) matrix between synthetic datasets and authentic competition corpora at the second-level domain granularity.

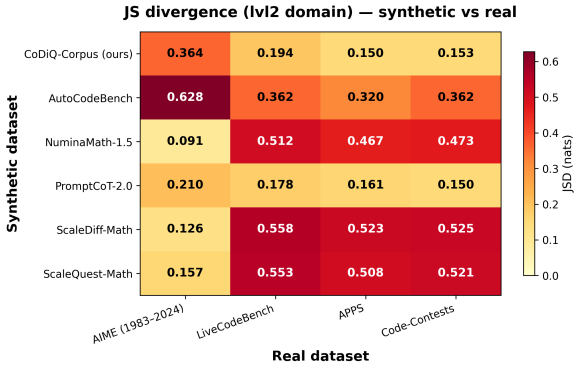


Figure 8: **Distributional alignment (JSD) between synthetic and authentic datasets.** Lower is more aligned. CoDiQ-Corpus exhibits the tightest semantic alignment with human-curated programming benchmarks among all evaluated synthesis pipelines.

## J Automatic Solvability Probe Across Datasets

To complement the human study (Section 5.2) with a fully automated, dataset-level signal, we run a

Table 11: **Full diversity statistics at the second-level domain granularity** ($N$=300 questions per dataset). The best, second-best and third-best scores for each indicator are shown in $\boxed{box}$, **bold** and underlined, respectively.

| Dataset | N | labels | lab/s | uniq | top | H | effN | simp | even | cov |
|---|---|---|---|---|---|---|---|---|---|---|
| ***Baselines (Human-Curated)*** | | | | | | | | | | |
| APPS | 300 | 651 | 2.17 | 143 | 11 | 3.667 | 39.1 | 0.932 | 0.739 | 0.477 |
| LiveCodeBench | 300 | 643 | 2.14 | 141 | 11 | 3.503 | 33.2 | 0.920 | 0.708 | 0.470 |
| Code-Contests | 300 | 695 | 2.32 | 176 | 13 | 3.846 | 46.8 | 0.942 | 0.744 | 0.587 |
| AIME (1983–2024) | 300 | 598 | 1.99 | 74 | 12 | 3.354 | 28.6 | 0.945 | 0.779 | 0.247 |
| ***Baselines (AI-Generated)*** | | | | | | | | | | |
| ScaleQuest-Math | 300 | 428 | 1.43 | 121 | 15 | 3.812 | 45.2 | 0.952 | 0.795 | 0.403 |
| NuminaMath-1.5 | 300 | 495 | 1.65 | 114 | 15 | 3.877 | 48.3 | 0.962 | 0.819 | 0.380 |
| AutoCodeBench | 300 | **800** | **2.67** | $\boxed{368}$ | $\boxed{19}$ | $\boxed{5.032}$ | $\boxed{153.2}$ | $\boxed{0.980}$ | $\boxed{0.852}$ | $\boxed{1.227}$ |
| ScaleDiff-Math | 300 | 574 | 1.91 | 116 | **16** | 3.741 | 42.1 | 0.954 | 0.787 | 0.387 |
| PromptCoT-2.0 | 300 | 658 | 2.19 | 115 | **16** | 3.821 | 45.7 | 0.962 | 0.805 | 0.383 |
| ***CoDiQ-Corpus (Ours)*** | | | | | | | | | | |
| CoDiQ-Corpus | 300 | $\boxed{898}$ | $\boxed{2.99}$ | **360** | 15 | **4.997** | **148.0** | **0.979** | **0.849** | **1.200** |

Table 12: Definition and interpretation of each diversity column.

| **Column** | **Meaning** | **Formula / range** | **Interpretation** |
|---|---|---|---|
| dataset | Dataset name (display alias) | — | Row identifier |
| N | Number of sampled questions | Fixed at 300 | Denominator baseline; `uniq` is not directly comparable across different $N$ |
| labels | Total label occurrences (multi-label, with duplicates) | $\sum_i \lvert\text{labels}(q_i)\rvert$ | Annotation density, not category count |
| lab/s | Average labels per sample | labels$/N$ | Multi-label width: 1.43=mostly single-label, 2.99=most $\geq$ 3 labels |
| uniq | Unique full domain paths | $\lvert$ set(full paths) $\rvert$ | Fine-grained richness; higher is broader |
| top | Unique top-level categories | $\lvert$ set(first segment) $\rvert$ | Coarse-grained richness; offsets leaf-level sparsification |
| H | Shannon entropy (full paths) | $-\sum p_i \log p_i$, nats, $\in [0, \log \text{uniq}]$ | Higher is more dispersed; not directly comparable across different `uniq` |
| effN | Effective number of categories | $\exp(H)$, $\in [1, \text{uniq}]$ | Equivalent uniform-distribution category count; the most intuitive cross-dataset summary |
| simp | Gini–Simpson index | $1 - \sum p_i^2$, $\in [0, 1]$ | Probability that two random labels differ; higher is more diverse |
| even | Pielou evenness | $H/\log \text{uniq}$, $\in [0, 1]$ | Normalized evenness; values $<$ 0.7 indicate dominant head categories |
| cov | Coverage | uniq$/N$ ($>$ 1 possible under multi-label) | New categories per sample; $>$ 1 means the catalog is not yet saturated |

Qwen3-32B solvability classifier (prompt in Appendix Q) on 300 questions sampled from each dataset. Results are summarized in Table 13.

Table 13: Automatic solvability rate via a Qwen3-32B classifier on 300 sampled questions per dataset. CoDiQ-Corpus is verified end-to-end during construction (Section 3.3.2); other synthetic baselines span a wide range.

| **Dataset** | **Solvability (%)** |
|---|---|
| *Synthetic baselines* | |
| ScaleQuest-Math | 68.0 |
| PromptCoT-2.0 | 82.7 |
| ScaleDiff-Math | 85.7 |
| NuminaMath-1.5 | 90.3 |
| AutoCodeBench | 98.3 |
| *Ours (verified end-to-end)* | |
| CoDiQ-Corpus | verified |

# K Solver-Token Scaling on Natural Problems

To verify that the token–difficulty relation discussed in Section 4.2.2 is not an artifact of our pipeline, we measure average Qwen3-32B solver tokens on MATH-500, whose difficulty levels were assigned by human annotators prior to any LLM. As shown in Table 14, the average solver-token cost grows monotonically with the human-defined difficulty level, exhibiting more than a 13× gap between Level 1 and Level 5. This confirms that solver-token consumption tracks intrinsic difficulty on naturally occurring problems, under the boundary conditions stated in the main text.

Table 14: Average Qwen3-32B solver tokens per question on MATH-500, grouped by human-assigned difficulty level.

| **Difficulty Level** | **Avg. Solver Tokens** |
|---|---|
| Level 1 | 337 |
| Level 2 | 460 |
| Level 3 | 522 |
| Level 4 | 1,409 |
| Level 5 | 4,705 |

# L Scaling Tendency Analysis Details

This section presents the complete scaling tendency analysis with all evaluated models. Figure 9 shows the full results of difficulty and solvability scaling across 8 generation rounds for all Long-CoT models under both Direct Prompt and CoDiQ Prompt settings.

The complete results reveal consistent scaling patterns across all models: (1) increased reasoning computation correlates with higher problem difficulty, and (2) a trade-off exists between difficulty and solvability, with larger models maintaining better balance between the two metrics.

# M CoDiQ-Corpus Quality Criteria

We establish rigorous criteria to assess the quality and solvability of problems in CoDiQ-Corpus. Three PhD-level domain experts independently evaluate 300 randomly sampled problems following these standardized guidelines:

## M.1 Information Completeness

- **Sufficient Parameters**: All necessary numerical values, variables, and constraints are explicitly provided.
- **Clear Objectives**: The problem goal is unambiguous and well-defined.
- **Complete Context**: No truncation or missing problem statements.

## M.2 Logical Consistency

- **Non-contradictory Conditions**: All given constraints are mutually consistent.
- **Valid Premises**: For logical problems, premises are sufficient to support conclusions.
- **Feasible Solutions**: The problem admits at least one valid solution path.

## M.3 Problem Well-definedness

- **Determinable Answer**: The answer can be uniquely determined or bounded within a reasonable range.
- **Appropriate Scope**: The problem complexity matches its stated domain and difficulty level.
- **Standard Formulation**: Follows conventional mathematical or logical notation.

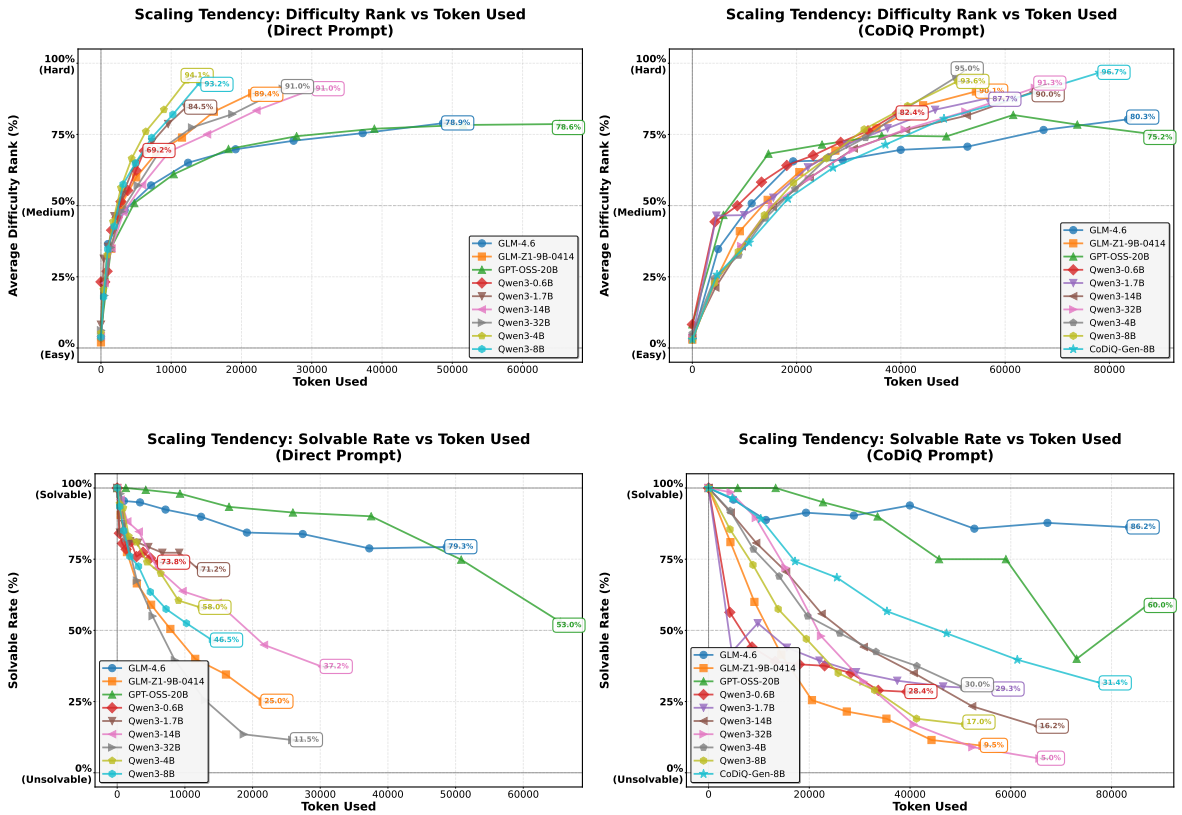


Figure 9: **Complete Scaling Analysis on CoDiQ-Bench.** Normalized average difficulty ranking (top row) and solvable rate (bottom row) of questions generated by all evaluated Long-CoT models across 8 rounds, using Direct Prompt (left) and CoDiQ Prompt (right). Higher rankings indicate higher question difficulty; higher rates indicate better question quality.

### M.4 Evaluation Protocol

Each expert assigns a binary solvability label (solvable/unsolvable) with confidence scores. A problem is marked as **solvable** only when at least two experts agree. Disagreements are resolved through discussion. The inter-annotator agreement (Fleiss' $\kappa$) reaches 0.78, indicating substantial consensus.

## N Curriculum learning Detail

### N.1 Training Data Selection for Curriculum Learning

To validate the effectiveness of CoDiQ-Corpus for curriculum learning, we carefully select question sequences with progressive difficulty structures. Specifically, we sample 480 question sequences from CoDiQ-Corpus where each sequence length $|S| \geq 3$, forming the curriculum learning dataset:

$$\mathcal{D}_{\text{curriculum}} = \{S_n\}_{n=1}^{480}, \quad |S_n| \geq 3 \tag{4}$$

We construct three training stages with increasing complexity for each sequence $S_n = \{q_0^n, q_1^n, q_2^n, \ldots, q_{|S_n|-1}^n\}$ with progressive difficulty:

- **Level 1 (L1):** Contains all initial questions $q_0^n$ from each sequence, representing the starting point of each difficulty progression.
- **Level 2 (L2):** Randomly samples one question from intermediate positions $\{q_1^n, q_2^n, \ldots, q_{|S_n|-2}^n\}$ for each sequence, capturing mid-stage complexity.
- **Level 3 (L3):** Contains all final questions $q_{|S_n|-1}^n$ from each sequence, representing the highest difficulty level within each progression.

Formally, the data selection strategy is defined as:

$$\mathcal{L}_1 = \{q_0^n \mid S_n \in \mathcal{D}_{\text{curriculum}}\} \tag{5}$$

$$\mathcal{L}_2 = \{\text{random}(\{q_i^n\}_{i=1}^{|S_n|-2}) \mid S_n \in \mathcal{D}_{\text{curriculum}}\} \tag{6}$$

$$\mathcal{L}_3 = \{q_{|S_n|-1}^n \mid S_n \in \mathcal{D}_{\text{curriculum}}\} \tag{7}$$

This design ensures a clear difficulty progression: $\text{Difficulty}(\mathcal{L}_1) < \text{Difficulty}(\mathcal{L}_2) < \text{Difficulty}(\mathcal{L}_3)$. The sample distribution across levels follows the ratio $|\mathcal{L}_1| : |\mathcal{L}_2| : |\mathcal{L}_3| = 2 : 2 : 1$,

achieved by duplicating $\mathcal{L}_1$ and $\mathcal{L}_2$ during training to balance exposure to different difficulty levels. This ratio is designed to provide sufficient foundational learning before progressing to more challenging problems, following curriculum learning principles (Bengio et al., 2009).

For the baseline model Qwen3-RL-4B, we use the original untransformed datasets (before applying the CoDiQ Pipeline) as training data, maintaining the same total number of training samples to ensure fair comparison. This allows us to isolate the impact of progressive difficulty transformation on model performance.

**Training Schedule:** Models are trained sequentially through three stages:

1. CoDiQ-$L_1$-4B: Trained on $\mathcal{L}_1$ (starting level)
2. CoDiQ-$L_2$-4B: Initialized from CoDiQ-$L_1$-4B, further trained on $\mathcal{L}_2$ (intermediate level)
3. CoDiQ-$L_3$-4B: Initialized from CoDiQ-$L_2$-4B, further trained on $\mathcal{L}_3$ (advanced level)

### N.2 Reward Signal Design

We design a multi-dimensional reward function to evaluate answer quality by prompting Qwen3-32B as an expert evaluator. The reward signal $r \in [0, 1]$ is computed based on four key dimensions:

**Evaluation Dimensions:**

- **Problem Resolution** ($s_{\text{pr}}$): Measures how completely the answer addresses all aspects of the question (0.0-1.0).
- **Reasoning Correctness** ($s_{\text{rc}}$): Evaluates the correctness and coherence of the reasoning process (0.0-1.0).
- **Information Completeness** ($s_{\text{ic}}$): Assesses whether all necessary information, steps, and explanations are included (0.0-1.0).
- **Accuracy** ($s_{\text{acc}}$): Measures factual correctness, calculation accuracy, and conceptual clarity (0.0-1.0).

The reward function aggregates these dimensions with carefully tuned weights optimized for high-difficulty mathematical reasoning tasks:

$$r = w_{\text{pr}} \cdot s_{\text{pr}} + w_{\text{rc}} \cdot s_{\text{rc}} + w_{\text{ic}} \cdot s_{\text{ic}} + w_{\text{acc}} \cdot s_{\text{acc}} \tag{8}$$

where the default weights are set as:

$$w_{\text{pr}} = 0.20, \quad w_{\text{rc}} = 0.35, \quad w_{\text{ic}} = 0.25, \quad w_{\text{acc}} = 0.20 \tag{9}$$

This configuration emphasizes reasoning quality (35%) and information completeness (25%), which are critical for complex problem-solving. The evaluation prompt instructs Qwen3-32B to assess each dimension independently using continuous scores and return results in JSON format. Special handling is applied for edge cases, such as correctly identifying unsolvable problems, which receives high problem resolution scores (0.8-1.0) despite not providing a numerical solution.

To ensure evaluation quality, we implement automatic validation of the returned scores, retry mechanisms (up to 3 attempts), and text truncation to handle long inputs (max 4096 tokens for questions, 16384 tokens for answers). The confidence score returned by the evaluator helps identify uncertain assessments for potential manual review.

## O Annotator Recruitment and Compensation

This section outlines the protocols for participant recruitment, compensation, and the qualification requirements for the human evaluation conducted in this study.

**Recruitment.** Three PhD students in quantitative disciplines were recruited on-campus through internal calls. All passed a short calibration exercise on a held-out set, and a subset of items was double-annotated to monitor inter-annotator agreement (Fleiss' $\kappa$=0.76).

**Compensation.** Each annotator received a flat honorarium of USD 50 via the host institution's standard participant-payment process, in line with local rates for graduate-student research participation. Payment was unconditional on annotation content; participants could withdraw at any time.

**Anonymity and consent.** No demographic attributes or identifiers linkable to individual annotations are retained. Only the minimum information required for honorarium disbursement was collected and deleted after payment. Informed consent was obtained prior to participation.

## P Instruction for LLMs Ranking

**Instruction for LLMs Ranking**

You are an expert in assessing question difficulty. Evaluate questions based on:

1. Knowledge Complexity: Number and depth of concepts required
2. Cognitive Load: Reasoning levels and abstract thinking needed
3. Computational Complexity: Steps and calculations involved
4. Traps and Common Mistakes: Hidden pitfalls in the question
5. Integration Skills: Cross-domain knowledge application required

Your task is to group questions by difficulty level and sort groups from easiest to hardest.

**Important:** Questions with the SAME difficulty level should be grouped together.

Analyze each question carefully and return them grouped by difficulty level.

**Output format requirements:**

- Return ONLY a valid JSON object with TWO fields:
- **"result"**: A list of lists (groups), each containing question indices of the SAME difficulty level
- **"reason"**: A list of strings, where `reason[i]` explains why questions in `result[i]` share the same difficulty
- Groups in both arrays should be ordered from easiest to hardest
- The length of "result" and "reason" arrays MUST be identical
- Use 0-based indexing matching the input order

**Example output format:**

```
{
  "result": [[1, 3], [0], [2, 4]],
  "reason": [
    "Both require only basic arithmetic operations with single-step reasoning
        ",
    "Multi-step algebraic manipulation with intermediate concepts",
    "Complex integration of advanced concepts and non-obvious strategies"
  ]
}
```

This means:

- Questions 1 and 3 are easiest (Group 0) - they both involve only basic arithmetic and single-step reasoning
- Question 0 is medium difficulty (Group 1) - it requires multi-step algebraic manipulation with intermediate concepts
- Questions 2 and 4 are hardest (Group 2) - they both demand complex integration of advanced concepts and non-obvious strategies

**Important:**

- Each reasoning string should explain the COMMON difficulty characteristics that unite all questions in the corresponding group
- Ensure `reason[i]` corresponds to `result[i]` for all groups

Please group the following questions by difficulty level and sort groups from easiest to hardest:
{questions}

Return the result as a JSON object with format:

```
{
  "result": [[indices of easiest group], [indices of next group], ...],
  "reason": ["reasoning for group 0", "reasoning for group 1", ...]
}
```

## Q Instructions for Solvability Check

**Instructions for Solvability Check**

You are an expert in analyzing mathematical and logical problems. Your task is to determine whether a given question is solvable.
A question is considered **SOLVABLE** if:

1. It provides all necessary information and conditions
2. The problem is well-defined with clear objectives
3. It has a determinable answer (even if complex)
4. The constraints are consistent (not contradictory)

A question is considered **UNSOLVABLE** if:

1. Missing critical information or parameters
2. Contains contradictory conditions
3. The problem statement is ambiguous or unclear
4. Asks for information that cannot be determined from given data

5. The question is incomplete or truncated

**Important Guidelines:**

- Be strict but reasonable in your judgment
- Consider if a reasonable person could solve the problem with the given information
- For mathematical problems, check if all necessary values are provided
- For logical problems, verify if the premises are sufficient for the conclusion

**Output format requirements:**

- Return ONLY a valid JSON object
- Must have exactly these fields:
  - "solvable": boolean (true/false)
  - "confidence": number (0.0-1.0, your confidence in the judgment)
  - "reason": string (brief explanation in English, max 200 characters)
  - "missing_info": list of strings (what information is missing, empty list if solvable)

**Example Outputs:**

```
{"solvable": true, "confidence": 0.95, "reason": "All necessary parameters
    provided, problem is well-defined", "missing_info": []}
{"solvable": false, "confidence": 0.85, "reason": "Missing the radius value
    needed to calculate circle area", "missing_info": ["radius"]}
```

## R Instruction for Direct Prompt

**Instruction for Direct Prompt**

**# Problem Difficulty Upgrade Generator**
**## Task Description**
You are an expert competitive programming problem creator. Your task is to take a given problem and create a significantly more challenging, competition-level version.

**## Input**
**Original Problem:**
{original_problem}

**## Output Format**
Return ONLY the new upgraded problem, nothing else.

[Your upgraded competitive programming problem here]

## S Instruction for CoDiQ Prompt

**Instruction for CoDiQ Prompt**

**# Problem Difficulty Upgrade Generator**

**## Task Description**

You are an expert competitive programming problem creator. Your task is to take a given problem and create a significantly more challenging, competition-level version by strategically adding difficulty elements that test deeper understanding and more complex reasoning.

**## Design Standards (Mandatory Quality Check)**

To ensure the upgraded problem is competition-worthy, you must strictly adhere to these principles:

1. **Deep Synthesis**: The difficulty element must naturally intertwine with the original logic. The solution should feel like a single cohesive challenge, not a "patchwork".
2. **Multi-Step Reasoning**: The solution must require 2-3 non-trivial intermediate logical jumps. The solver must derive lemmas or intermediate states before applying standard algorithms.
3. **No Trivial Upgrades**: Avoid simply increasing N to $10^5$ if the logic remains $O(N)$. The upgrade must force a change in complexity class (e.g., from Greedy to Flow, from Simulation to Matrix Exponentiation).
4. **Disguise & Abstraction**: (If applicable) Hide the core theorem or data structure behind a unique story or abstract mathematical setting. Never explicitly name the required algorithm.

**## Difficulty Elements Library (Select 1-2 distinct elements)**

**### Category A:**

**Dimensionality & Constraints**

**Best for:** Array/Sequence/Tree problems with simple naive solutions (e.g., $O(N)$, $O(N^2)$, or $O(N^3)$).

**Avoid when:** Original problem already requires logarithmic or sublinear complexity.

**Description:** Explode the data scale or dimensionality to invalidate simple simulation or brute force.

**Core Strategy:**

1. **Identify** the naive complexity (e.g., $O(N)$, $O(N^2)$, or $O(N^3)$).
2. **Impose** constraints that compel a superior complexity class (e.g., $O(\log N)$ or $O(N \log N)$).
3. **Introduce** dynamic updates, higher-dimensional spaces, or multiple query types to break linear scans.

**Examples:**

```
[
    {
        "original": "Given an array of size N (N≤1000), find the sum of
            elements in range [L, R].",
```

```
        "upgrade": "Given an array of size N (N≤10^5), handle M (M≤10^5)
            operations: 1. Update range [L, R] by adding V. 2. Query sum of
            range [L, R]. (Requires Segment Tree with Lazy Propagation)"
    },
    {
        "original": "Given a grid, find the shortest path from (0,0) to (R,C)
            avoiding obstacles.",
        "upgrade": "Given a grid where obstacles appear and disappear at
            specific time intervals modulo K. Find the shortest path. (
            Requires BFS in State Space (x, y, time%K))"
    },
    {
        "original": "Find the maximum value in an array.",
        "upgrade": "Given a tree with N nodes (N≤10^5), support path updates (
            add value V to all nodes on path u-v) and path maximum queries. (
            Requires Heavy-Light Decomposition)"
    },
    {
        "original": "Given a set of points, find the two closest points.",
        "upgrade": "Given a set of points in 3D space, find the size of the
            largest subset where every pair has Manhattan distance > D. (
            Requires Coordinate Transformation + Data Structures)"
    },
    {
        "original": "Check if a string S contains pattern P.",
        "upgrade": "Given a text S and K patterns. Support dynamic insertion
            of new patterns and query if any pattern appears in S. (Requires
            Aho-Corasick Automaton or Suffix Structures)"
    }
]
```

### Category B:

**Mathematical Abstraction**

**Best for:** Problems that can be reframed into mathematical structures, e.g., simulation or iterative problems with clear patterns.

**Avoid when:** The problem is already focused on advanced specialized theorems or complex data structures.

**Description:** Transform a procedural or descriptive problem into a formal model, e.g., using number theory, combinatorics, or game theory.

**Core Strategy:**

1. **Increase** constraints to push beyond computational limits (e.g., $N \geq 10^{18}$), making simple iteration or simulation impossible.

2. **Force** the discovery of underlying structures, such as closed-form formulas, recurrence relations, or invariant properties.

3. **Introduce** formal constraints (e.g., modular arithmetic, coordinate systems) that require rigorous mathematical modeling.

**Anti-pattern:** Simply making N large without ensuring a mathematical insight exists is not valid.

**Examples:**

```
[
    {
        "original": "Simulate a process where bacteria double every hour. Find
             count at hour N (N≤50).",
        "upgrade": "Bacteria have a complex growth rule F(n) = a*F(n-1) + b*F(
            n-2). Find count at hour N (N≤10^18) modulo 10^9+7. (Requires
            Matrix Exponentiation)"
    },
    {
        "original": "Given N items, in how many ways can you pick K items?",
        "upgrade": "Given N items with specific color constraints, calculate
            the number of ways to pick K items modulo 10^9+7 where N is up to
            10^9. (Requires Lucas Theorem or Generating Functions)"
    },
    {
        "original": "Two players take turns removing 1-3 stones. Who wins?",
        "upgrade": "Played on a graph with N≤10^5 nodes. A token moves along
            edges. A player loses if they cannot move. The graph has cycles. (
            Requires Game Theory on Graphs / Sprague-Grundy with loop handling
            )"
    },
    {
        "original": "Calculate the Greatest Common Divisor (GCD) of two
            numbers.",
        "upgrade": "Calculate the sum of GCD(i, j) for all 1 ≤ i, j ≤ N where
            N≤10^7. (Requires Euler Totient Function / Mobius Inversion)"
    },
    {
        "original": "Find the area of a polygon given integer coordinates.",
        "upgrade": "Given N lines in the plane, find the area of their union
            region accurately. Handle parallel and concurrent lines. (Requires
             Integration logic or Green's Theorem application)"
    }
]
```

**### Category C:**

**Inverse & Constructive**

**Best for:** Problems with well-defined algorithms and a clear "input → algorithm → output" flow.

**Avoid when:** The original problem's core challenge is already in the "design/construction" phase rather than "computation/processing" (i.e., problems that lack a standard algorithm to reverse).

**Description:** Instead of asking for the result of a process, ask for the input that produces a specific result.

**Core Strategy:**

1. **Reverse**the problem direction: from "Given X, find Y" to "Construct X such that Y holds".
2. **Require** understanding of structural properties (e.g., what makes a graph have a specific flow?).
3. **Add** multiple constraints to make construction non-trivial.

**Examples:**

```
[
    {
        "original": "Given a graph, find the shortest path from A to B.",
        "upgrade": "Construct a graph with N vertices and M edges such that
            the shortest path from 1 to N is exactly L, and the MST weight is
            exactly W."
    },
    {
        "original": "Sort an array using QuickSort.",
        "upgrade": "Construct a permutation of size N that causes a standard
            QuickSort implementation (with first element as pivot) to hit its
            worst-case O(N^2) time complexity."
    },
    {
        "original": "Given a binary tree, print its pre-order traversal.",
        "upgrade": "Given the pre-order and post-order traversals, reconstruct
             all possible binary trees. Determine if the solution is unique or
             count how many such trees exist."
    },
    {
        "original": "Check if a string is a palindrome.",
        "upgrade": "Construct a string of length N containing exactly K
            distinct palindromic substrings. Prove that no such string exists
            if K exceeds a certain bound."
    },
    {
        "original": "Find the maximum flow in a network.",
        "upgrade": "Given a desired max flow value F, construct a network with
             minimum edges that achieves this flow, subject to capacity
            constraints on each edge."
    }
]
```

**### Category D:**

**State Explosion**

**Best for:** Problems with simple, polynomial DP states that can be enriched.

**Avoid when:** The original state space is already exponential (e.g., TSP); adding dimensions would make it computationally infeasible.

**Description:** Add complex dependencies or history requirements that necessitate advanced Dynamic Programming or Network Flow by expanding the state space.

**Core Strategy:**

1. **Redefine** the state: move from simple states (e.g., ‘dp[i]‘) to composite, multi-dimensional states (e.g., adding an exponential ‘mask‘ for sets or a polynomial ‘remainder‘ for constraints).

2. **Introduce** constraints that depend on past choices or specific history (e.g., "cannot visit a node visited k steps ago," requiring a sliding window or history state).

3. **Add** multiple orthogonal restrictions (e.g., count, parity, or modularity) that must be tracked simultaneously.

1299

**Anti-pattern:** Simply adding variables that don't interact. The new dimensions must fundamentally change the recurrence logic or transition dependencies.

**Examples:**

```
[
    {
        "original": "Climb stairs taking 1 or 2 steps. How many ways?",
        "upgrade": "Cover a 3×N grid with 1×2 dominoes. How many ways modulo
            10^9+7? (Requires Broken Profile DP / Bitmask DP to track cross-
            section state)"
    },
    {
        "original": "Knapsack Problem: Max value with weight limit W.",
        "upgrade": "Knapsack on a Tree: Each node has value/weight. Max value
            by selecting nodes such that no two selected nodes are adjacent,
            and total weight ≤ W. (Requires Tree DP + Knapsack dimensions)"
    },
    {
        "original": "Longest Increasing Subsequence in an array.",
        "upgrade": "Count the number of permutations of length N that have a
            Longest Increasing Subsequence of length exactly K. (Requires DP
            with Young Tableaux or RSK correspondence)"
    },
    {
        "original": "Find min cost to traverse a grid.",
        "upgrade": "Find min cost to traverse a grid with K 'batteries' to
            jump obstacles; recharge depends on grid value modulo M, and cells
             must be unlocked in order. (State: position + battery_count +
            mod_state + lock_mask)"
    },
    {
        "original": "Edit Distance between two strings.",
        "upgrade": "Given strings A and B, find the number of strings S of
            length L such that EditDistance(A, S) ≤ K and EditDistance(B, S) ≤
             K. (Requires DP on DP / Automaton DP)"
    }
]
```

### Category E:

**Theorem Disguise**

**Best for:** Problems that can map to classic high-level algorithms but appear in unrelated or abstract domains.

**Avoid when:** The original problem explicitly mentions the algorithm or data structure.

**Description:** Hide a sophisticated algorithmic core behind a narrative or alternative mathematical structure that misleads intuition.

**Core Strategy:**

1. **Map** the problem to a well-known non-trivial algorithm (e.g., Network Flow, Linear Basis, Generating Functions, or Advanced DS).

2. **Remove** all technical terminology and explicit constraints that hint at the solution.

3. **Create** a "Red Herring" narrative that suggests an intuitive but suboptimal approach (e.g., Greedy, simple DP, or naive Simulation).
4. **Ensure** the bridge between the surface problem and the hidden theorem requires a deep structural insight.

**Examples:**

```
[
    {
        "original": "Find the maximum number of meetings one can attend.",
        "upgrade": "Given a set of intervals, color them with minimum colors
            so no overlapping intervals share a color. Some intervals are 'VIP
            ' and must use specific colors. (Disguised as Greedy, actually
            Interval Graph Coloring / Dilworth's Theorem with constraints)"
    },
    {
        "original": "Can we partition an array into two equal sum subsets?",
        "upgrade": "Given a graph, determine if it is bipartite. If not,
            remove minimum edges to make it bipartite while preserving
            connectivity. (Disguised Graph Theory, actually Odd Cycle
            Transversal / Max Cut variant)"
    },
    {
        "original": "Assign tasks to workers to minimize total time.",
        "upgrade": "N workers and M tasks. Each pair (worker, task) has a cost
            . Some workers conflict and cannot work simultaneously. Minimize
            total cost. (Disguised Minimum Cost Maximum Flow with conflict
            resolution)"
    },
    {
        "original": "Find the maximum number of compatible intervals.",
        "upgrade": "Given a circle with N chords represented by their
            endpoints. Find the maximum number of non-intersecting chords. (
            Disguised DP on Circle Graph or Catalan-related structure)"
    },
    {
        "original": "Find a number in an array that appears more than N/2
            times.",
        "upgrade": "Given a stream of N elements, find all elements appearing
            more than N/K times using O(K) space and one pass. Handle K up to
            10^3. (Disguised Boyer-Moore Voting generalized / Misra-Gries
            Algorithm)"
    }
]
```

**### Category F:**

**Edge Case & Rigor Engineering**

**Best for:** Problems with simple logic but tricky implementation details.

**Avoid when:** The original problem is already highly technical or implementation-heavy.

**Description:** Focus on logical pitfalls, geometric precision, tricky edge cases, or requiring formal proof for correctness.

**Core Strategy:**

1301

1. **Target** weaknesses of standard data types (e.g., float precision, integer overflow, empty sets).
2. **Introduce** degenerate cases (e.g., disconnected graphs, collinear points, zero denominators).
3. **Require** strict mathematical proofs (like "Exchange Arguments") to justify a strategy where intuition fails.
4. **Add** precision constraints (e.g., "answer accurate to $10^{-9}$") or handle extreme ranges.

**Examples:**

```
[
    {
        "original": "Given coordinates of 3 points, calculate the area of the
            triangle.",
        "upgrade": "Given N (N≤10^5) lines in a plane. Find the area of the
            finite region formed by their intersection. Handle parallel lines,
             lines at infinity, and precision errors. (Requires Half-plane
            Intersection with strict boundary handling and epsilon comparison)
            "
    },
    {
        "original": "Find if there is a path from A to B in a graph.",
        "upgrade": "Find if A reaches B. The graph may contain self-loops,
            multiple edges, and consists of up to 10^5 disconnected components
            . Answer 10^5 online queries. Handle the case where A=B and the
            case where A or B doesn't exist. (Tests Edge Cases: Connectivity,
            Disjoint Set Union with path compression, query validation)"
    },
    {
        "original": "Divide A by B.",
        "upgrade": "Divide A by B where A and B are strings representing
            integers up to 10^1000. Handle B=0, negative signs, leading zeros,
             and output the quotient and remainder. Prove your division
            algorithm's correctness. (Requires BigInteger Implementation +
            comprehensive Edge Case handling)"
    },
    {
        "original": "Greedy: Always pick the largest coin available to make
            change.",
        "upgrade": "Given a non-standard coin system (e.g., 1, 3, 4), find the
             smallest value V where the greedy approach fails to find the
            minimum number of coins. Prove that for all values below V, greedy
             is optimal. (Requires finding the Edge Case of the algorithm
            itself + Exchange Argument)"
    },
    {
        "original": "Sort an array of tasks by duration.",
        "upgrade": "Given N tasks with processing time and decay rates. Each
            task loses value at a different rate while waiting. Find a
            processing order to minimize total decay. Prove your sorting
            strategy using an Exchange Argument. (Requires formal proof to
            derive the correct comparator; simple sorting is wrong)"
    }
]
```

## Construction Protocol (Internal Thinking Process)

1. **Analyze Original**: Identify the naive solution and its complexity.
2. **Select Category**: Choose 1-2 categories from the library above that best fit the problem's potential.
3. **Apply Core Strategy**: Use the "Core Strategy" defined in your selected category to redesign the problem constraints and objectives.
4. **Review**: Check against the "Design Standards". Does it require multi-step reasoning? Is the theorem disguised?
5. **Final Output**: Write the problem statement clearly using standard CP formatting.

## Input
**Original Problem:**
{original_problem}

## Output Format
Return ONLY the new upgraded problem, nothing else.

[Your upgraded competitive programming problem here]

# T Instruction for Domain Classification

**Instruction for Domain Classification**

I am working with natural language statements of advanced competition-level problems across mathematics and computer science. Before formalization or implementation, I need to categorize each problem into its appropriate domain.
**#OBJECTIVE#**

1. Summarize the problem in one or two sentences, highlighting the key concepts, strategies, or structures involved.
2. Classify the problem into one or more domains using a hierarchical classification chain.
   For example: Algebra -> Intermediate Algebra -> Inequalities.
   Or: Algorithms -> Dynamic Programming -> Knapsack Problems.
   The classification should be based on the content of the natural language problem, even if no formal notation is used yet.

**#DOMAIN TAXONOMY#** The domain classification should follow a standard taxonomy. If a problem does not fit any listed category, you are free to create a reasonable new classification chain following the same three-level format: Major Domain -> Subdomain -> Specific Topic.
<domains>
## Mathematics

```
Algebra -> Intermediate Algebra -> Inequalities
Algebra -> Intermediate Algebra -> Polynomials
Algebra -> Intermediate Algebra -> Functional Equations
Algebra -> Intermediate Algebra -> Other
Algebra -> Linear Algebra -> Vector Spaces
Algebra -> Linear Algebra -> Matrices
Algebra -> Linear Algebra -> Other

Number Theory -> Elementary Number Theory -> Divisibility
Number Theory -> Elementary Number Theory -> Modular Arithmetic
Number Theory -> Elementary Number Theory -> Diophantine Equations
Number Theory -> Elementary Number Theory -> Prime Numbers
Number Theory -> Elementary Number Theory -> Other

Geometry -> Euclidean Geometry -> Triangles
Geometry -> Euclidean Geometry -> Circles
Geometry -> Euclidean Geometry -> Coordinate Geometry
Geometry -> Euclidean Geometry -> Transformations
Geometry -> Euclidean Geometry -> Other
Geometry -> Analytic Geometry -> Conic Sections
Geometry -> Analytic Geometry -> Other
Geometry -> Computational Geometry -> Convex Hull
Geometry -> Computational Geometry -> Other

Combinatorics -> Enumerative Combinatorics -> Permutations
Combinatorics -> Enumerative Combinatorics -> Combinations
Combinatorics -> Enumerative Combinatorics -> Other
Combinatorics -> Extremal Combinatorics -> Pigeonhole Principle
Combinatorics -> Extremal Combinatorics -> Other
Combinatorics -> Constructive Combinatorics -> Invariants
Combinatorics -> Constructive Combinatorics -> Other
Combinatorics -> Graph Theory -> Trees
Combinatorics -> Graph Theory -> Shortest Paths
Combinatorics -> Graph Theory -> Network Flow
Combinatorics -> Graph Theory -> Other

Calculus -> Differential Calculus -> Derivatives
Calculus -> Differential Calculus -> Other
Calculus -> Integral Calculus -> Definite Integrals
Calculus -> Integral Calculus -> Other

Probability -> Discrete Probability -> Expected Value
Probability -> Discrete Probability -> Conditional Probability
Probability -> Discrete Probability -> Other
Probability -> Continuous Probability -> Distributions
Probability -> Continuous Probability -> Other

Discrete Mathematics -> Logic -> Propositional Logic
Discrete Mathematics -> Logic -> Other
Discrete Mathematics -> Set Theory -> Cardinality
Discrete Mathematics -> Set Theory -> Other
Discrete Mathematics -> Other -> Other
```

## Computer Science

```
Algorithms -> Dynamic Programming -> Knapsack Problems
```

```
Algorithms -> Dynamic Programming -> Sequence Problems
Algorithms -> Dynamic Programming -> Other
Algorithms -> Greedy Algorithms -> Scheduling
Algorithms -> Greedy Algorithms -> Other
Algorithms -> Divide and Conquer -> Merge Sort Variants
Algorithms -> Divide and Conquer -> Other
Algorithms -> Backtracking -> Constraint Satisfaction
Algorithms -> Backtracking -> Other
Algorithms -> Sorting and Searching -> Binary Search
Algorithms -> Sorting and Searching -> Other
Algorithms -> Mathematical Algorithms -> Fast Exponentiation
Algorithms -> Mathematical Algorithms -> Number Theoretic Algorithms
Algorithms -> Mathematical Algorithms -> Other
Algorithms -> Other -> Other

Data Structures -> Trees -> Binary Search Trees
Data Structures -> Trees -> Segment Trees
Data Structures -> Trees -> Other
Data Structures -> Graphs -> Adjacency Representations
Data Structures -> Graphs -> Other
Data Structures -> Heaps -> Priority Queues
Data Structures -> Heaps -> Other
Data Structures -> Union-Find -> Weighted Union-Find
Data Structures -> Union-Find -> Other
Data Structures -> Hash Structures -> Hash Maps
Data Structures -> Hash Structures -> Other
Data Structures -> Other -> Other

String Algorithms -> Pattern Matching -> KMP
String Algorithms -> Pattern Matching -> Other
String Algorithms -> Suffix Structures -> Suffix Arrays
String Algorithms -> Suffix Structures -> Other
String Algorithms -> Other -> Other

Artificial Intelligence -> Machine Learning -> Supervised Learning
Artificial Intelligence -> Machine Learning -> Unsupervised Learning
Artificial Intelligence -> Machine Learning -> Other
Artificial Intelligence -> Deep Learning -> Neural Networks
Artificial Intelligence -> Deep Learning -> Other
Artificial Intelligence -> Search -> Heuristic Search
Artificial Intelligence -> Search -> Other
Artificial Intelligence -> Other -> Other

Cryptography -> Classical Cryptography -> Substitution Ciphers
Cryptography -> Classical Cryptography -> Other
Cryptography -> Modern Cryptography -> Symmetric Encryption
Cryptography -> Modern Cryptography -> Asymmetric Encryption
Cryptography -> Modern Cryptography -> Other
Cryptography -> Cryptanalysis -> Other
Cryptography -> Other -> Other

Systems -> Operating Systems -> Process Management
Systems -> Operating Systems -> Other
Systems -> Databases -> Query Optimization
Systems -> Databases -> Other
```

1307

```
Systems -> Networks -> Protocol Design
Systems -> Networks -> Other
Systems -> Other -> Other

Complexity Theory -> Time Complexity -> Big-O Analysis
Complexity Theory -> Time Complexity -> Other
Complexity Theory -> Space Complexity -> Other
Complexity Theory -> Computability -> Other
Complexity Theory -> Other -> Other

Formal Methods -> Proof Assistants -> Lean
Formal Methods -> Proof Assistants -> Coq
Formal Methods -> Proof Assistants -> Other
Formal Methods -> Program Verification -> Other
Formal Methods -> Other -> Other
```

</domains>

**#RESPONSE FORMAT#**

**## Summarization**

*[A brief summary of the problem, describing the key concepts, strategies, or structures involved.]*

**## Domains**

*[A hierarchical classification of the domains involved in the problem.]*

**#INSTRUCTIONS#**

- You may include up to three domain classification chains, separated by semicolons.
- The format must be: Major Domain -> Subdomain -> Specific Topic.
- If a problem does not fit any listed category, create a new reasonable classification chain following the same three-level format. Do not force an ill-fitting category.
- Avoid using vague or overlapping categories.
- End each report with the line `=== report over ===`.

**#INPUT#** Below is the original natural language problem statement:

```
<statement>
{statement}
</statement>
```

**#OUTPUT FORMAT#** Please return your response in JSON format. For example:

```
{
  "Summary": "This problem involves finding the optimal subarray sum using a
      divide and conquer strategy combined with dynamic programming.",
  "Domains": [
    "Algorithms -> Dynamic Programming -> Sequence Problems",
    "Algorithms -> Divide and Conquer -> Other"
  ]
}
```