# VDE: Verifiable Dynamic Evaluation of Mathematical Reasoning via Typed Bipartite Graphs

**Yihan Lu** **Junzhe Zhang** **Xiaojun Wan***
Wangxuan Institute of Computer Technology, Peking University
Beijing, China
luyihan@stu.pku.edu.cn, junzhezhang@stu.pku.edu.cn
wanxiaojun@pku.edu.cn

## Abstract

Mathematical reasoning benchmarks are increasingly limited by static test sets, which are vulnerable to contamination and slow to adapt as models improve, while manually building fresh high-quality problems is expensive. We propose VDE (Verifiable Dynamic Evaluation), a dynamic evaluation framework that generates replay-verifiable math problems from typed value–theorem bipartite graphs. Each instance is an executable graph with model-independent ground truth recovered by deterministic replay rather than model-produced solutions. Beyond compositional depth, VDE supports explicit constraint injection and controlled branching to produce harder problems while preserving verifiability. We instantiate the same framework in analytic geometry and number theory, showing that a unified graph-based generation paradigm can cover structurally different domains. Experiments on frontier models show that performance remains far from saturation and degrades predictably as we increase construction depth, constraints, and branching, making VDE a scalable and trustworthy testbed for dynamic mathematical reasoning evaluation.

## 1 Introduction

Mathematical reasoning is a central testbed for modern language models because it requires multi-step composition, precise symbolic manipulation, and long-range consistency under verifiable final answers. Widely used benchmarks span grade-school word problems to competition mathematics, including GSM8K [Cobbe et al., 2021], MATH [Hendrycks et al., 2021], and OlympiadBench [He et al., 2024].

Yet static math benchmarks are increasingly limited. Fixed test sets become less discriminative as models improve and are vulnerable to contamination, while evaluation often depends on free-form solutions that are difficult to verify robustly at scale. Without a trustworthy external verifier, model errors, hallucinated steps, and brittle answer normalization can obscure what is actually being measured.

*Dynamic evaluation* offers a natural alternative by generating fresh items at evaluation time. Prior work such as DyVal [Zhu et al., 2023], DyVal2 [Zhu et al., 2024], TreeEval [Li et al., 2025a], and LatestEval [Li et al., 2024] explores controllable generation, adaptive probing, or freshness-aware construction, including dynamically generated mathematical reasoning tasks. However, dynamic evaluation for mathematics still faces domain-specific challenges: mathematics spans heterogeneous subfields with different object types and inference rules, so many generators cover only a narrow slice of mathematical structure; difficulty control often comes from longer operation chains rather

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

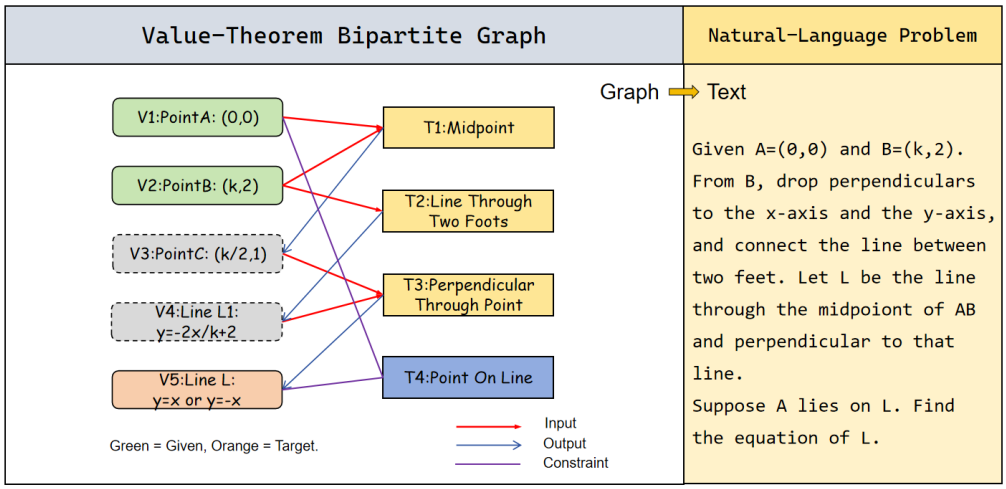


Figure 1: From executable graph to problem text. An example of our value-theorem bipartite graph representation on the left, where green nodes are givens and orange denotes the target; theorem nodes encode theorem applications with typed input/output dependencies. The corresponding natural-language problem statement rendered from the presentation graph is shown on the right.

than explicit constraints or multi-solution disambiguation; generated instances can be quickly solved by frontier models if they do not induce deeper dependency structure; and some pipelines still rely on model-produced solutions or LLM judges rather than model-independent verification. Proof-centric settings address verification more directly, but they also introduce substantial formalization and tooling overhead.

We address this gap with Verifiable Dynamic Evaluation (VDE)[2], a framework for mathematical reasoning based on executable *typed bipartite graphs* over *Value* nodes and *theorem* nodes. Each instance is an auditable graph whose ground-truth answer is recovered by deterministic replay, not by solving with the evaluated model. Beyond compositional depth, VDE supports explicit constraint injection and controlled branching to produce harder instances while preserving exact verifiability. The same graph-based construction paradigm extends across domains through domain-specific type systems and theorem libraries; in this paper, we instantiate it in *analytic geometry* and *number theory*. A geometry example is shown in Figure 1.

VDE therefore targets the central tension in dynamic math evaluation: generating fresh, frontier-discriminative problems across mathematical domains while maintaining accurate, reproducible, model-independent ground truth. We include reproducibility details and artifact documentation in the appendix to facilitate future research. Our main contributions are:

- We propose VDE, a dynamic evaluation framework for *mathematical* reasoning based on automated construction over typed bipartite graphs, making each instance an auditable executable graph with replay-verifiable, model-independent ground truth.
- We introduce constraint injection and controlled branching to increase difficulty beyond compositional depth while preserving exact symbolic/numeric verifiability.
- We instantiate the framework in analytic geometry and number theory, showing that a unified executable-graph paradigm can support multiple domains with stable executability and interpretable difficulty control.

[2]Project repository: `https://github.com/LuYanjie12/VDE`.

## 2 Related Work

**Static math reasoning benchmarks.** Static math benchmarks remain central for evaluating reasoning, spanning grade-school and competition-style problem solving, including GSM8K [Cobbe et al., 2021], MATH [Hendrycks et al., 2021], and OlympiadBench [He et al., 2024]. Geometry-focused benchmarks and solvers form an important neighboring line: GeoQA studies multimodal geometric QA [Chen et al., 2021], Inter-GPS combines formal languages with symbolic reasoning for geometry problem solving [Lu et al., 2021], and AlphaGeometry targets Olympiad-level Euclidean geometry solving [Trinh et al., 2024]. These resources are valuable for measuring or improving problem-solving ability on fixed tasks, but they do not directly address our target setting: dynamically generating fresh, controllably difficult, replay-verifiable evaluation instances at test time.

**Dynamic evaluation and leakage-resistant protocols.** Dynamic and contamination-resistant evaluation instead emphasizes freshness at evaluation time. DyVal [Zhu et al., 2023] and DyVal2 [Zhu et al., 2024] generate graph-structured reasoning tasks with controllable complexity, TreeEval [Li et al., 2025a] explores benchmark-free probing through tree planning, LatestEval [Li et al., 2024] leverages time-sensitive sources, and NPHardEval periodically refreshes algorithmic instances [Fan et al., 2024]. MathArena similarly highlights the value of uncontaminated mathematical evaluation through fresh competition problems [Balunović et al., 2025]. Our work is closest in spirit to this line, but differs in two ways that are especially important for mathematics: we target executable instance generation with model-independent ground truth, and we seek difficulty control beyond longer operation chains, using typed theorem applications, explicit constraints, and controlled branching. This distinction matters because dynamic math evaluation can otherwise have limited coverage across mathematical subfields, weak separation among frontier systems when generated problems are too easily solved, and validation pipelines that rely on model-produced solutions or LLM judges [Wei et al., 2025, Jacovi et al., 2024].

**Geometry, number theory, and formal mathematical reasoning.** Beyond benchmarks, a broader body of work studies geometry theorem proving, symbolic solving, and formal verification. Classical automated geometry theorem proving includes Wu's method and related mechanical theorem-proving principles [Wenjun, 2008]. More recent systems include PyEuclid for formal plane geometry [Li et al., 2025b], autoformalization for Euclidean geometry [Murphy et al., 2024], and LeanGeo for formalizing competition geometry in Lean [Song et al., 2025]. In formal mathematical reasoning more broadly, AlphaProof [Hubert et al., 2025], miniF2F [Zheng et al., 2021], LeanDojo [Yang et al., 2023], and ProofNet [Azerbayev et al., 2023] study theorem proving and autoformalization at scale. Number-theoretic formalization has also advanced substantially, including formalized developments for zeta and $L$-functions, p-adic $L$-functions, Mason–Stothers, and the Ramanujan–Nagell theorem [Loeffler and Stoll, 2025, Narayanan, 2023, Baek and Lee, 2024, Banwait, 2026]. These works are highly relevant neighboring directions, but their primary goal is to solve, prove, or formalize given mathematical statements. VDE instead targets benchmark construction: generating new natural-language problems with auditable execution traces and exact replayable answers. This distinction is particularly important now that our framework is instantiated in both analytic geometry and number theory rather than a single geometry-only setting.

## 3 Method

### 3.1 Overview

In our VDE framework, each problem is treated as an executable *typed value–theorem bipartite graph*. Concretely, a problem instance is represented by a directed bipartite graph

$$G = (V \cup T, E), \qquad E \subseteq (V \times T) \cup (T \times V), \tag{1}$$

whose bipartite structure explicitly separates *value nodes* (typed mathematical objects) from *theorem nodes* (typed construction rules), with edges encoding typed input–output dependencies. Each value node stores a typed mathematical object or quantity under a type system $\mathcal{T}$ (e.g., $\mathsf{Point}$ in geometry or $\mathsf{Integer}$ in number theory). For each type $\tau \in \mathcal{T}$, $\mathcal{D}_\tau$ denotes its domain of valid values (e.g., $\mathcal{D}_{\mathsf{Point}} = \mathbb{R}^2$ or $\mathcal{D}_{\mathsf{Integer}} = \mathbb{Z}$). A theorem node is an executable typed construction rule that maps its input objects to output objects.

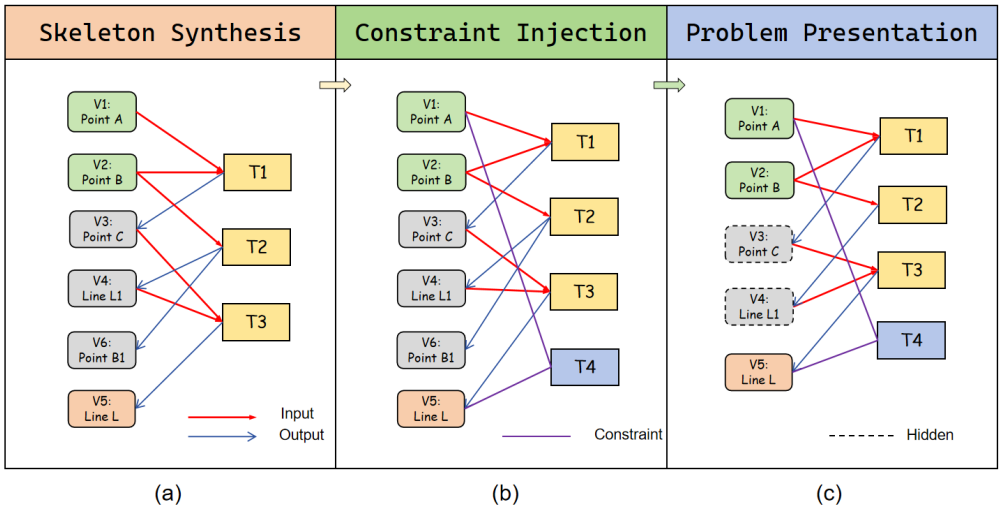


Figure 2: Three graph views of the same generated instance, illustrating our pipeline from *skeleton synthesis* to *constraint injection* and *problem presentation*. (a) **Skeleton synthesis:** a constructive bipartite graph where value nodes (green/gray) feed theorem nodes (T1–T3) via input edges (red) and produce new values via output edges (blue). (b) **Constraint injection:** additional constraint theorem nodes (e.g., T4) introduce extra relations (purple) that couple previously constructed values and densify dependencies. (c) **Problem presentation:** a compact, human-facing subgraph obtained by hiding auxiliary values/relations (dashed), while preserving the same executable structure and model-independent ground truth.

$$\prod_{v \in \mathrm{In}(o)} \mathcal{D}_{\tau(v)} \to \prod_{v \in \mathrm{Out}(o)} \mathcal{D}_{\tau(v)}, \tag{2}$$

$$\mathcal{C}_o = \{\, c(\cdot) = 1 \,\}. \tag{3}$$

Here $\mathcal{C}_o$ is the set of predicate constraints associated with theorem node $o$ (e.g., non-degeneracy checks).

Our framework consists of three stages: (i) **Skeleton Synthesis**, which reverse-expands from a target to obtain an executable theorem skeleton; (ii) **Constraint Injection**, which adds theorem-backed predicates to reshape the feasible solution space; and (iii) **Problem Presentation**, which extracts a compact presentation subgraph, instantiates the graph to obtain model-independent ground truth, and renders it into natural language with an LLM.

### 3.2 Theorem Library

The theorem library $\mathcal{L}$ defines the admissible schemas and executable semantics for each domain. We manually author candidate theorems with mathematical meaning, typed input/output signatures, and preconditions, then use an LLM to convert them into a unified machine-readable specification. After manual verification, we retain only sound, executable, non-degenerate schemas. Formally, each theorem $o \in \mathcal{L}$ provides a typed signature $(\mathrm{In}(o), \mathrm{Out}(o))$, an executor $\mathrm{Exec}_o$, and predicate checks $\mathcal{C}_o$. We instantiate this interface in analytic geometry and number theory; representative schemas appear in Table 1.

The same interface supports two theorem roles. *Construct theorems* create new values and form the forward-computable skeleton $G_c$; if a theorem has multiple valid outputs, the corresponding value is propagated as a solution space. *Constraint theorems* add predicates over existing objects, shrinking $\Omega(G_c, G_k)$ without extending the construction skeleton. Appendix B reports the executable library sizes and value-node type coverage; Table 1 shows representative schemas from both roles and domains.

Table 1: Representative theorem schemas from our theorem library $\mathcal{L}$, including examples from analytic geometry and number theory.

| ROLE | THEOREM | SIGNATURE (IN $\rightarrow$ OUT) | SYMPY EXECUTOR |
|---|---|---|---|
| CONSTRUCT | MIDPOINT (GEO.) | {POINT $A$, POINT $B$} $\rightarrow$ {POINT $M$} | `M = Point((A.x + B.x)/2, (A.y + B.y)/2)` |
| CONSTRUCT | GCD (NUM.) | {INTEGER $a$, INTEGER $b$} $\rightarrow$ {INTEGER $g$} | `g = gcd(a, b)` |
| CONSTRAINT | RADICAL_AXIS (GEO.) | {CIRCLE $c_1$, CIRCLE $c_2$, POINT $P$} $\rightarrow$ | `pow(P, c1) == pow(P, c2)` |
| CONSTRAINT | DIVIDES (NUM.) | {INTEGER $d$, INTEGER $n$, PREDICATE} $\rightarrow$ | `Mod(n, d) == 0` |

### 3.3 Skeleton Synthesis

As shown in Figure 2(a), we synthesize an executable theorem skeleton $G_c = (V \cup T_c, E_c)$ by reverse expansion from a designated target value node $v_\star$(e.g. target V5 Line L in Figure 2(a)). Here, $T_c$ denotes the set of *constructive theorem nodes* (theorem instances that produce non-given values in the skeleton), and $E_c$ denotes the directed bipartite dependency edges connecting value nodes and constructive theorems. Starting with the unresolved set $U = \{v_\star\}$, we repeatedly select an unresolved value node $v \in U$, sample a theorem schema $o \in \mathcal{L}$ together with an output slot $y \in \mathrm{Out}(o)$ whose type matches $\tau(v)$, and bind $y \mapsto v$ so that $o \in T_c$ becomes the unique producer of $v$. We then introduce (or reuse) the typed inputs required by $o$ as value nodes and add those without producers to $U$. The process stops when a theorem budget is reached (or $U$ becomes empty); any remaining unresolved values are declared as givens. The detailed procedure is deferred to Appendix G (Algorithm 1). The resulting graph is a directed bipartite DAG in which each non-given value has exactly one producer, enabling deterministic forward replay in topological order once the givens are fixed.

### 3.4 Constraint Injection

Skeletons produced by SKELETON SYNTHESIS alone often yield weakly structured instances: givens are largely arbitrary, and difficulty is governed mainly by the length of the construction chain. As shown in Figure 2(b), we therefore keep the synthesized skeleton $G_c$ fixed and inject an additional *constraint graph* $G_k = (V_k \cup T_k, E_k)$ composed of constraint theorems that restrict admissible assignments to selected value nodes in $G_c$. Here, $T_k$ denotes the set of injected constraint-theorem nodes and $V_k$ contains any auxiliary values introduced to support these predicates when needed. Constraint theorems primarily contribute predicate sets $\mathcal{C}_o$ (and may introduce auxiliary hidden values when needed) to express deeper relations while keeping the eventual statement concise. Let $A$ be a (partial) assignment over value nodes. Constraint injection defines the feasible set

$$\Omega(G_c, G_k) \;=\; \{A :\; \forall o \in T_k,\; \forall c \in \mathcal{C}_o,\; c(A) = 1\}\,, \tag{4}$$

thereby reshaping the instance distribution from a weakly structured prior into a structured, theorem-backed solution space. We choose injection locations using graph-aware heuristics (e.g., nodes with high structural influence or small distance to the target $v_\star$), and we optionally allow constraints to introduce auxiliary constructs required to articulate the intended relation.

### 3.5 Problem Presentation

As shown in Figure 2(c), from the instantiated executable graph, we first extract a minimal presentation graph $G_p$, a compact subgraph that contains only the givens and the minimal construction chain needed to determine the target $v_\star$. In particular, $G_p$ includes (i) the given value nodes and their typed declarations, and (ii) the query node $v_\star$ together with a minimal set of construction relations linking givens to $v_\star$.

**Graph Instantiation** We instantiate an instance by treating all givens as free parameters and executing the construct skeleton $G_c$ symbolically/parametrically in a forward pass. Concretely, starting from a parameter vector $\theta$ that assigns values to all givens, forward execution propagates these parameters through theorems and produces, for each value node $v$, a solution set $\mathcal{S}(v;\theta) \subseteq \mathcal{D}_{\tau(v)}$

rather than a single value. This set-valued semantics is essential: due to multi-solution theorems, each value node represents the entire feasible solution space under the current givens.

We define the branching factor $S$ as the number of candidate solution branches induced by set-valued theorems before enforcing all downstream constraints. Let $Y(\theta) \subseteq V$ denote the set of ambiguous value nodes encountered during forward execution (i.e., $|\mathcal{S}(y;\theta)| > 1$). The corresponding pre-constraint branch space is

$$\mathcal{B}(\theta) \;=\; \prod_{y \in Y(\theta)} \mathcal{S}(y;\theta), \tag{5}$$

and we define the branching-node count

$$B \;=\; |Y(\theta)|. \tag{6}$$

Thus, $B$ counts how many value nodes induce a genuine branch choice, while $S$ measures the total size of the pre-constraint branch space. We then set

$$S \;=\; |\mathcal{B}(\theta)|. \tag{7}$$

Note that $S$ counts *possible* branches prior to full constraint propagation; after enforcing injected constraints, fewer branches may remain feasible.

**Constraint Satisfaction and Pruning** Constraint satisfaction is then formulated as a system of predicate equations over $\theta$ induced by the injected constraint graph $G_k$. Let $A(\theta)$ denote the induced (set-valued) assignment after forward propagation. Feasibility requires that every constraint predicate is satisfiable by at least one consistent selection from the solution sets:

$$\begin{aligned} \theta \text{ is feasible} \quad \Longleftrightarrow \quad & \exists\, A \in \prod_{v \in V} \mathcal{S}(v;\theta) \\ & \text{s.t. } \forall o \in T_k,\ \forall c \in \mathcal{C}_o,\ c(A) = 1. \end{aligned} \tag{8}$$

Operationally, when a theorem yields multiple candidates for an ambiguous value $y$, we branch over its solution set $\mathcal{S}(y;\theta) = \{s_1, \ldots, s_m\}$ where $m = |\mathcal{S}(y;\theta)|$, expanding

$$A^{(k)} = A \cup \{y \mapsto s_k\}, \qquad k = 1, \ldots, m, \tag{9}$$

and retain only those partial assignments that can still be extended to satisfy all active constraints.

To prevent numerically pathological instances from obscuring the intended mathematical structure, we apply a suite of *numeric filters* to reject branches when intermediate expressions become overly complex, thereby keeping generated problems within a controlled arithmetic complexity regime while preserving the underlying theorem-driven reasoning.

**Problem Statement Rendering** We then render $G_p$ into a natural-language statement by invoking GPT-5.2 as the primary rendering model to verbalize the typed givens and the selected relations, and to pose a query for $v_\star$. Finally, we validate every released instance by deterministic replay: we re-execute the stored bipartite graph from the givens, re-check all predicates, and confirm that the recomputed target matches the stored answer within tolerance. Instances that fail replay, exhibit numerical instability, or violate non-degeneracy constraints are discarded. Detailed prompt is shown in Appendix H.1.

# 4 Experiments

## 4.1 Data Generation via VDE

We build a synthetic benchmark of mathematical reasoning problems with our VDE framework in two domains: analytic geometry and number theory. We generate 500 instances for each domain–difficulty bucket, yielding 3,000 VDE instances in total across two domains and three difficulty tiers. We partition both benchmarks into three difficulty tiers (Easy/Medium/Hard) using the same three generation knobs defined in §3: (1) **construct depth** $|T_c|$ in the skeleton $G_c$; (2) **constraint budget** $|T_k|$ in the constraint graph $G_k$; and (3) **branching-node count** $B$, the number of ambiguous value nodes induced by set-valued theorems. The Easy tier uses $|T_c| = 0$–$5$, $|T_k| = 0$, and $B = 0$; the Medium tier uses $|T_c| = 6$–$10$, $|T_k| = 1$–$2$, and $B = 1$; and the Hard tier uses $|T_c| = 11$–$15$, $|T_k| = 3$–$4$, and $B \geq 2$. We use $B$ rather than the total branch-space size $S$ for tiering, since a single branching node may produce more than two candidates in domains such as number theory.

### 4.2 Evaluated Models and Main Results

We benchmark a compact set of widely-used frontier models under a uniform zero-shot setting. *Closed-source models* include OpenAI GPT-5.2 and GPT-4o [OpenAI, 2025, 2024]; Anthropic Claude Sonnet (4.5, 3.5) [Anthropic, 2025, 2024]; Google Gemini (Gemini 3 Pro, Gemini 2.5 Pro) [Google, 2025b,a, Comanici et al., 2025]. We additionally include the *open-source* Qwen2.5-14B-Instruct [Qwen et al., 2025]. All models are evaluated zero-shot with a unified prompt and answer-extraction protocol; the full evaluation prompt is provided in Appendix H.2. We compare against GSM8K, MATH, and AIME 2025; auxiliary statement-level profiling is deferred to Appendix A.

Table 2: Evaluation results across our benchmark and classic static benchmarks. Results are reported as Zero-shot Accuracy (%).

| MODEL | GEOMETRY | | | NUMBER THEORY | | | CLASSIC | | |
|---|---|---|---|---|---|---|---|---|---|
| | EASY | MED | HARD | EASY | MED | HARD | GSM8K | MATH | AIME'25 |
| GPT-5.2 | 75.0 | 63.7 | 55.4 | 82.6 | 73.4 | 66.8 | 99.2 | 96.1 | 100.0 |
| GPT-4O | 38.1 | 23.7 | 11.1 | 49.4 | 37.0 | 25.8 | 94.3 | 76.9 | 26.7 |
| CLAUDE SONNET 4.5 | 68.4 | 52.1 | 41.5 | 77.8 | 66.2 | 56.4 | 98.5 | 91.4 | 87.0 |
| CLAUDE SONNET 3.5 | 51.2 | 35.6 | 21.2 | 62.8 | 49.0 | 35.8 | 96.4 | 71.1 | 16.0 |
| GEMINI 3 PRO | 62.8 | 51.9 | 48.5 | 71.0 | 62.4 | 55.6 | 98.9 | 94.5 | 100.0 |
| GEMINI 2.5 PRO | 42.5 | 28.3 | 18.7 | 54.6 | 42.0 | 31.4 | 96.2 | 78.3 | 88.0 |
| QWEN2.5-14B-INSTRUCT | 31.4 | 15.1 | 10.8 | 43.2 | 29.6 | 21.4 | 96.1 | 74.5 | 20.2 |

Table 2 reports zero-shot accuracy across our VDE benchmarks and the classic static benchmarks. Across models, VDE exhibits the intended Easy-to-Hard degradation pattern wherever results are available, while classic static benchmarks are substantially closer to saturation for frontier systems. This trend matches our generation protocol: increasing construct depth $|T_c|$, injected constraint budget $|T_k|$, and branching-node count $B$ jointly induce deeper dependency chains, tighter long-range coupling, and more ambiguity to resolve. Together with the structural profiling in Table 3, these findings provide initial evidence that VDE's difficulty knobs and graph abstractions transfer across the two studied domains.

A natural concern is that VDE may be substantially easier for systems that can write and execute code. Appendix D evaluates Python tool-use and coding-agent variants on the same rendered statements without exposing the executable graph or replay trace. Tool access improves accuracy, especially on number-theory arithmetic, but does not saturate VDE and preserves the Easy-to-Hard degradation.

### 4.3 External Structural Comparison

After converting each item into a common *presentation-level* typed bipartite graph format, we compare VDE against sampled subsets of external mathematical datasets. This projection does not reconstruct a fully executable VDE graph and is not used for answer verification; it is a profiling tool that maps visible objects, relations, construction clauses, constraints, and queries into comparable value/operator counts. Appendix E details the sampling sources, projection rules, and metric definitions.

Table 3 gives exploratory evidence about structural differences between VDE and external static datasets under this shared projection. As difficulty increases, VDE becomes more construct-heavy and accumulates more derived values and explicit constraints. By contrast, projected subsets of MATH and OlympiadBench are more constraint-driven and expose less explicit constructive structure under our presentation-level abstraction, with representative geometry and number-theory instances in Appendix F.1–F.2 and the Appendix case studies in Appendix F.

### 4.4 Error Analysis

One possible confound is that failures attributed to reasoning may instead come from renderer-specific phrasing. We check this in Appendix C by re-rendering the same executable graphs with GPT-, Gemini-, and Qwen-based renderers while keeping the replay target fixed; accuracy changes by less than one percentage point on average. Thus, rendering-sensitive cases are routed to representation/misreading or manual review, while replay-grounded dependency and consistency failures are

Table 3: Bipartite graph statistics across VDE and external datasets (means). **V/T** denote value/theorem nodes, and **Derived** counts non-given values. Bold marks the maximum within each domain block.

| Dataset | Domain | $N$ | V | T | Given | Derived | Construct | Constraint |
|---|---|---|---|---|---|---|---|---|
| JGEX-231 | Geometry | 231 | 9.25 | 8.02 | 1.02 | 8.23 | 7.01 | 1.00 |
| GeoQA | Geometry | 3503 | 10.18 | 3.95 | 6.23 | 3.95 | 2.11 | 0.85 |
| MATH | Geometry | 1349 | 18.19 | 7.47 | **10.71** | 7.47 | 1.21 | **5.34** |
| OlympiadBench | Geometry | 129 | 13.69 | 3.30 | 10.39 | 3.30 | 0.28 | 2.05 |
| VDE-Easy | Geometry | 500 | 10.80 | 4.41 | 5.10 | 5.70 | 4.41 | 0.00 |
| VDE-Med | Geometry | 500 | 12.72 | 8.04 | 6.68 | 6.04 | 6.98 | 1.06 |
| VDE-Hard | Geometry | 500 | **19.26** | **16.08** | 8.30 | **10.96** | **12.88** | 3.20 |
| MATH | Number Theory | 1409 | 7.01 | 1.94 | 5.08 | 1.94 | 0.07 | 0.97 |
| OlympiadBench | Number Theory | 127 | 12.14 | 3.17 | **8.97** | 3.17 | 0.26 | 1.91 |
| VDE-Easy | Number Theory | 500 | 10.34 | 4.42 | 4.92 | 5.42 | 4.42 | 0.00 |
| VDE-Med | Number Theory | 500 | 15.46 | 11.00 | 5.44 | 10.02 | 9.04 | 1.96 |
| VDE-Hard | Number Theory | 500 | **21.44** | **18.60** | 5.36 | **16.08** | **14.46** | **4.14** |

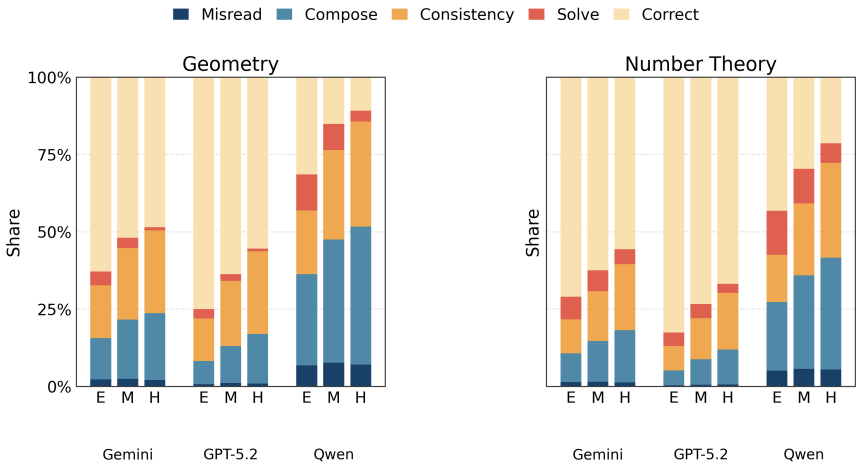


Figure 3: Error composition by model, difficulty, and domain on our benchmark. The two panels report analytic geometry and number theory separately. Each bar sums to 1 (all instances): the top segment is accuracy, and the remainder is split into four attributed error types.

kept separate. We attribute incorrect predictions to four replay-grounded failure types: (1) **Representation / relation misunderstanding**; (2) **Multi-step composition and planning**; (3) **Global consistency and disambiguation**; and (4) **Symbolic / numeric solving**. We apply this scheme to Gemini 3 Pro, GPT-5.2, and Qwen2.5-14B on Easy/Medium/Hard buckets in both domains using deterministic replay evidence; the prompt is provided in Appendix H.3.

We report the results in Figure 3. Across tiers, stronger models show fewer **Solving** mistakes, while **Composition** and **Consistency** dominate, most notably on Medium/Hard with deeper dependencies and more branching. The results also indicate frequent overlap between **Composition** and **Consistency**, whereas **Misreading** is relatively rare for the strongest model, suggesting that the main limitation is cross-relation integration and global self-consistency rather than single-step computation.

### 4.5 Controlled Difficulty Analysis

A key design goal of VDE is interpretable difficulty control through explicit generation knobs. We probe construct depth, constraint tightness, and solution ambiguity by sweeping one knob at a time while holding the others fixed (Figure 4).

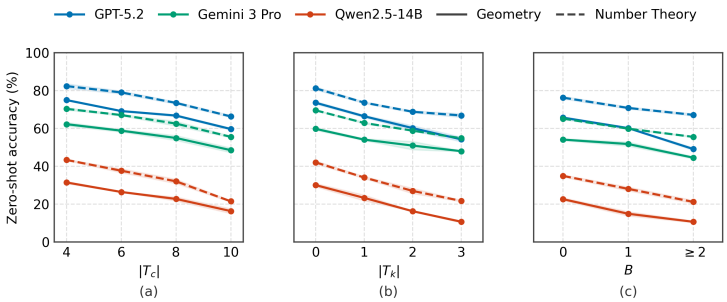


Figure 4: **Interpretable difficulty control.** Zero-shot accuracy (%) as we vary (a) construct depth $|T_c|$, (b) constraint count $|T_k|$, and (c) solution-branching level. Each panel contains both analytic geometry and number theory curves; each sweep changes one knob while holding the others fixed.

Construct-theorem difficulty ($|\boldsymbol{T_c}|$): we vary $|\boldsymbol{T_c}| \in \{\mathbf{4, 6, 8, 10}\}$ while holding $|\boldsymbol{T_k}| = \mathbf{1}$ and $\boldsymbol{B} = \mathbf{1}$; accuracy drops steadily as dependency chains lengthen (Figure 4a). Constraint-theorem difficulty ($|\boldsymbol{T_k}|$): we vary $|\boldsymbol{T_k}| \in \{\mathbf{0, 1, 2, 3}\}$ while fixing $|\boldsymbol{T_c}| = \mathbf{8}$ and $\boldsymbol{B} = \mathbf{1}$; even small $|\boldsymbol{T_k}|$ can substantially reduce accuracy (Figure 4b), indicating that the harder cases are driven by global consistency rather than extra steps. Multi-solution difficulty ($\boldsymbol{B}$): we sweep $\boldsymbol{B} \in \{\mathbf{0, 1, \geq 2}\}$ while holding $|\boldsymbol{T_c}| = \mathbf{8}$ and $|\boldsymbol{T_k}| = \mathbf{1}$ fixed, and accuracy decreases as $\boldsymbol{B}$ increases (Figure 4c), with failures dominated by *early branch commitment*.

Taken together, the three sweeps separate compositional depth, consistency pressure, and disambiguation pressure. The monotone trends in both domains indicate that the generation knobs reliably control distinct difficulty axes rather than simply changing surface length. We still record the pre-constraint branch-space size $\boldsymbol{S}$ as an auxiliary ambiguity statistic, but use $\boldsymbol{B}$ for main tiering because it is more stable across domains.

### 4.6 Human Quality Inspection

We conduct a stratified manual audit to verify that the rendered statements remain correct and solvable as difficulty increases. Annotators inspect 20 instances from each domain–difficulty bucket (120 total) for statement correctness, solvability, and hidden leakage; detailed guidelines are provided in Appendix I.

Table 4: Human audit pass rates (%) by domain–difficulty bucket. Each bucket contains 20 audited instances.

| Criterion | Geo-E | Geo-M | Geo-H | Num-E | Num-M | Num-H | Overall |
|---|---|---|---|---|---|---|---|
| # Audited | 20 | 20 | 20 | 20 | 20 | 20 | 120 |
| Correctness | 100.0 | 95.0 | 95.0 | 95.0 | 95.0 | 90.0 | 95.0 |
| Solvability | 100.0 | 100.0 | 95.0 | 100.0 | 95.0 | 95.0 | 97.5 |
| Hidden leakage | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 95.0 | 99.2 |

Table 4 shows consistently high pass rates across all three criteria, suggesting that the observed performance degradation is driven by reasoning difficulty rather than rendering artifacts. Inter-annotator agreement is strong (macro Cohen's $\boldsymbol{\kappa = 0.91}$).

## 5 Conclusion

We propose a dynamic, contamination-resistant evaluation framework for mathematical reasoning based on executable typed value–theorem bipartite graphs. This representation enables deterministic replay to compute ground-truth answers without relying on LLM judges, while constraint injection and controlled multi-solution branching strengthen difficulty control beyond longer operation chains. We instantiate the framework in both analytic geometry and number theory, and our main controlled

experiments show that, in the geometry setting, accuracy drops consistently as construct depth, constraint coupling, and branching increase. More broadly, the results support typed executable graphs as a practical foundation for replay-verifiable dynamic mathematical evaluation across domains.

## Acknowledgments

This work was supported by Beijing Natural Science Foundation (L253001), Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology) and National Engineering Research Center of New Electronic Publishing Technologies. We appreciate the anonymous reviewers for their helpful comments.

**Competing interests.** The authors declare no competing interests.

## References

Anthropic. Introducing Claude 3.5 sonnet. `https://www.anthropic.com/news/claude-3-5-sonnet`, 2024. Accessed 2026-01-26.

Anthropic. Introducing Claude sonnet 4.5. `https://www.anthropic.com/news/claude-sonnet-4-5`, 2025. Accessed 2026-01-26.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023.

Jineon Baek and Seewoo Lee. Formalizing mason-stothers theorem and its corollaries in lean 4. *arXiv preprint arXiv:2408.15180*, 2024.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *arXiv preprint arXiv:2505.23281*, 2025.

Barinder S Banwait. A formal proof of the ramanujan–nagell theorem in lean 4. *arXiv preprint arXiv:2604.09808*, 2026.

Jiaqi Chen, Jianheng Tang, Jinghui Qin, Xiaodan Liang, Lingbo Liu, Eric Xing, and Liang Lin. Geoqa: A geometric question answering benchmark towards multimodal numerical reasoning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 513–523, 2021.

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

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, and Yongfeng Zhang. NPHardEval: Dynamic benchmark on reasoning ability of large language models via complexity classes. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4092–4114, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.225. URL `https://aclanthology.org/2024.acl-long.225/`.

Google. Gemini 2.5: Our most intelligent AI model. `https://blog.google/innovation-and-ai/models-and-research/google-deepmind/gemini-model-thinking-updates-march-2025/`, 2025a. Accessed 2026-01-26.

Google. Gemini 3 pro: the frontier of vision AI. `https://blog.google/innovation-and-ai/technology/developers-tools/gemini-3-pro-vision/`, 2025b. Accessed 2026-01-26.

Dan Greenstein, Zohar Karnin, Chen Amiraz, and Oren Somekh. Linguistic and argument diversity in synthetic data for function-calling agents, 2026. URL https://arxiv.org/abs/2601.17829.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, 2024.

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

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, pages 1–3, 2025.

Alon Jacovi, Yonatan Bitton, Bernd Bohnet, Jonathan Herzig, Or Honovich, Michael Tseng, Michael Collins, Roee Aharoni, and Mor Geva. A chain-of-thought is as strong as its weakest link: A benchmark for verifiers of reasoning chains, 2024. URL https://arxiv.org/abs/2402.00559.

Xiang Li, Yunshi Lan, and Chao Yang. Treeeval: Benchmark-free evaluation of large language models through tree planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24485–24493, 2025a.

Yucheng Li, Frank Guerin, and Chenghua Lin. Latesteval: Addressing data contamination in language model evaluation through dynamic and time-sensitive test construction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18600–18607, 2024.

Zhaoyu Li, Hangrui Bi, Jialiang Sun, Zenan Li, Kaiyu Yang, and Xujie Si. Pyeuclid: A versatile formal plane geometry system in python. In *International Conference on Computer Aided Verification*, pages 405–420. Springer, 2025b.

David Loeffler and Michael Stoll. Formalizing zeta and l-functions in lean. *Annals of Formalized Mathematics*, 1, 2025.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-gps: Interpretable geometry problem solving with formal language and symbolic reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6774–6786, 2021.

Logan Murphy, Kaiyu Yang, Jialiang Sun, Zhaoyu Li, Anima Anandkumar, and Xujie Si. Autoformalizing euclidean geometry. *arXiv preprint arXiv:2405.17216*, 2024.

Ashvni Narayanan. Formalization of $p$-adic $l$-functions in lean 3. *arXiv preprint arXiv:2302.14491*, 2023.

OpenAI. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/, 2024. Accessed 2026-01-26.

OpenAI. Introducing GPT-5.2. https://openai.com/index/introducing-gpt-5-2/, 2025. Accessed 2026-01-26.

OpenCompass. opencompass/aime2025. Hugging Face Datasets, 2025. URL https://huggingface.co/datasets/opencompass/AIME2025. Accessed: 2026-01-26.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Chendong Song, Zihan Wang, Frederick Pu, Haiming Wang, Xiaohan Lin, Junqi Liu, Jia Li, and Zhengying Liu. Leangeo: Formalizing competitional geometry problems in lean. *arXiv preprint arXiv:2508.14644*, 2025.

Teerapaun Tanprasert and David Kauchak. Flesch-kincaid is not a text simplification evaluation metric. In Antoine Bosselut, Esin Durmus, Varun Prashant Gangal, Sebastian Gehrmann, Yacine Jernite, Laura Perez-Beltrachini, Samira Shaikh, and Wei Xu, editors, *Proceedings of the First Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*, pages 1–14, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.gem-1.1. URL `https://aclanthology.org/2021.gem-1.1/`.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Hui Wei, Shenghua He, Tian Xia, Fei Liu, Andy Wong, Jingyang Lin, and Mei Han. Systematic evaluation of llm-as-a-judge in llm alignment tasks: Explainable metrics and diverse prompt templates, 2025. URL `https://arxiv.org/abs/2408.13006`.

W Wenjun. Basic principles of mechanical theorem proving in elementary geometries. *Selected Works Of Wen-Tsun Wu*, page 195, 2008.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612, 2023.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang Gong, Diyi Yang, and Xing Xie. Dyval: Dynamic evaluation of large language models for reasoning tasks. *arXiv preprint arXiv:2309.17167*, 2023.

Kaijie Zhu, Jindong Wang, Qinlin Zhao, Ruochen Xu, and Xing Xie. Dyval 2: Dynamic evaluation of large language models by meta probing agents. *arXiv preprint arXiv:2402.14865*, 3, 2024.

## A Statement-Level Profiling

Our dataset is compared with three classic static benchmarks, GSM8K [Cobbe et al., 2021], MATH [Hendrycks et al., 2021], and AIME 2025 [OpenCompass, 2025]. Table 5 provides complementary dataset profiling statistics computed from the natural-language statements. We report basic length measures (**Chars**, **Tokens**), together with **MATTR** [Greenstein et al., 2026] as a lexical-diversity measure and **FKGL** [Tanprasert and Kauchak, 2021] as a coarse readability proxy. Overall, VDE provides a controllable three-tier benchmark family across two domains, and the profiling statistics contextualize the surface complexity of the generated statements relative to classic static datasets.

Table 5: Unified dataset profiling statistics computed on problem statements (means).

| Dataset | $N$ | Chars | Tokens | MATTR | FKGL |
|---|---|---|---|---|---|
| VDE-Geo Easy | 500 | 288.0 | 82.1 | 0.715 | 4.23 |
| VDE-Geo Med | 500 | 323.7 | 99.9 | 0.677 | 3.60 |
| VDE-Geo Hard | 500 | 336.7 | 106.5 | 0.671 | 3.39 |
| VDE-NT Easy | 500 | 235.9 | 61.6 | 0.738 | 4.85 |
| VDE-NT Med | 500 | 262.1 | 72.0 | 0.731 | 4.28 |
| VDE-NT Hard | 500 | 289.4 | 81.8 | 0.738 | 3.91 |
| GSM8K | 8792 | 235.3 | 45.9 | 0.699 | 5.29 |
| AIME'25 | 30 | 363.1 | 70.0 | 0.691 | 8.07 |
| MATH | 500 | 195.9 | 38.8 | 0.706 | 5.30 |

## B Theorem-Library Statistics

Table 6 reports the theorem-library sizes used in VDE after expanding the domain libraries. We count a schema only if it has a typed signature, executable semantics, and associated precondition/predicate fields. The predicate/check count includes all explicit preconditions and validation checks stored in the schema, so it is a proxy for the amount of theorem-local filtering rather than the number of generated problem constraints.

Table 6: Detailed theorem-library statistics for the schemas used in VDE.

| Domain | Role | #Schemas | #Predicates | Avg. pred./schema |
|---|---|---|---|---|
| Analytic geometry | Construct | 70 | 96 | 1.37 |
| Analytic geometry | Constraint | 24 | 48 | 2.00 |
| Number theory | Construct | 62 | 90 | 1.45 |
| Number theory | Constraint | 28 | 72 | 2.57 |
| Analytic geometry | Total | 94 | 144 | 1.53 |
| Number theory | Total | 90 | 162 | 1.80 |

The geometry value-node types cover coordinate-plane objects and derived scalar quantities, including Point2D, Line2D, Circle, Segment, Angle, and Real. The number-theory value-node types cover integer and modular objects, including Integer, PositiveInteger, Prime, CompositeInteger, Residue, and Modulus. The geometry library covers distances, angles, powers of points, intersections, projections, reflections, midpoints, parallel/perpendicular constructions, circumcircles, diameter circles, triangle centers, area constraints, and quadrilateral relations. The number-theory library covers integer arithmetic and modular reasoning, including gcd/lcm, divisibility, congruence, modular inverse, Chinese remaindering, modular roots, residues, prime factors, coprimality, valuation-style checks, and linear/quadratic congruence constraints. In both domains, construct schemas are used to build the executable skeleton, while constraint schemas add predicate checks over existing values without changing the target-producing construction chain.

## C Rendering Robustness

VDE treats the executable graph as the source of mathematical truth and the natural-language statement as a presentation layer. This separation creates a specific robustness question: are the main accuracy trends caused by graph difficulty, or by idiosyncratic phrasing from the LLM renderer? We therefore run a controlled rendering-robustness experiment on a held-out subset of generated graphs. The experiment keeps the *same executable graph and replay target* fixed while changing only the faithful natural-language rendering.

**Setup.** We sample the same number of graphs from each domain and difficulty tier, then render every graph with the same *presentation graph + rendering prompt* protocol used in the main method, varying only the LLM used for rendering. Table 7 uses short renderer labels: GPT denotes GPT-5.2, GEMINI denotes Gemini 3 Pro, and QWEN denotes Qwen2.5-14B-Instruct. The GPT render is the primary render used in the main benchmark. All variants share the same graph identifier, replay target, givens, selected relations, answer-normalization code, and deterministic replay checker; only the LLM-realized natural-language surface form changes. We evaluate two representative solver models from the main experiments and report accuracy averaged over the matched graph set.

**Faithfulness filtering.** Each rendered variant is filtered before it is used for model evaluation or error attribution. We reject a rendering if it introduces objects not present in the presentation graph, omits required givens or selected relations, exposes hidden replay-only intermediate values, changes the target, or makes the intended target under-specified. Automatic checks compare the rendered metadata against the stored graph and replay record; uncertain cases are sent to manual inspection using the same correctness, solvability, and hidden-leakage criteria as Appendix I.

Table 7: Rendering robustness on matched executable graphs. Each row evaluates the same graph set and replay targets under a different natural-language rendering. Entries report zero-shot accuracy (%); $\Delta$ is the absolute difference from the primary GPT render.

| Solver | Render | Geo-E | Geo-M | Geo-H | NT-E | NT-M | NT-H |
|---|---|---|---|---|---|---|---|
| GPT-5.2 | GPT | 75.0 | 63.7 | 55.4 | 82.6 | 73.4 | 66.8 |
| GPT-5.2 | GEMINI | 74.6 | 63.2 | 55.0 | 82.1 | 73.0 | 66.4 |
| GPT-5.2 | QWEN | 73.9 | 62.8 | 54.6 | 81.8 | 72.7 | 66.0 |
| Gemini 3 Pro | GPT | 62.8 | 51.9 | 48.5 | 71.0 | 62.4 | 55.6 |
| Gemini 3 Pro | GEMINI | 62.5 | 51.4 | 48.1 | 70.6 | 62.0 | 55.2 |
| Gemini 3 Pro | QWEN | 62.1 | 51.0 | 47.8 | 70.2 | 61.7 | 54.9 |
| Mean $\Delta$ vs. GPT | GEMINI | 0.35 | 0.50 | 0.40 | 0.45 | 0.40 | 0.40 |
| Mean $\Delta$ vs. GPT | QWEN | 0.90 | 0.90 | 0.75 | 0.80 | 0.70 | 0.75 |

**Interpretation.** The matched-graph design isolates LLM-renderer choice from graph difficulty. Across GPT, GEMINI, and QWEN renderings, all mean deltas are below one percentage point, which is much smaller than the Easy-to-Hard degradation in Table 2. Thus, the main trend is unlikely to be explained by renderer-specific phrasing. Rendering-sensitive cases are still useful diagnostically: if a model fails only under one LLM-rendered statement while succeeding on other faithful renderings of the same graph, we route the case to the representation/misreading category or manual review. If the same intermediate replay mismatch appears across renderings, we treat the failure as graph-level composition, consistency, branch-selection, or symbolic/numeric solving error.

**Statement-level validation.** The statement-correctness failures in Table 4 were caused by incomplete or inaccurate natural-language renderings; the underlying executable graphs still passed deterministic replay, predicate checking, and target verification. An LLM-assisted statement validator identified all rendering errors flagged by human annotators in the audited sample. This check assesses statement faithfulness and remains separate from replay-based ground-truth verification.

**Reported artifacts.** For reproducibility, each multi-renderer record stores the graph id, renderer id, rendering-prompt id, rendered statement, replay target, validation status, and any manual-review flag.

When model-provider terms prevent releasing raw model responses, the artifact stores extracted final answers, correctness labels, renderer ids, and hashes of the raw responses.

## D Tool-Use and Coding-Agent Baselines

VDE instances are generated from executable graphs, so a relevant stress test is whether the benchmark is solved once the model can call external computation. This experiment addresses a different question from deterministic replay: replay is used only by the evaluator to recover ground truth, while the evaluated system receives only the rendered natural-language problem statement. We therefore compare direct solving against two tool-enabled settings.

**Setup.** All settings use the same rendered statements, answer format, answer-extraction rules, and replay-derived targets as Table 2. The direct setting is the GPT-5.2 zero-shot baseline with no external tools. The two tool-enabled settings receive the problem statement as plain text and must return the same final-answer line used in the main evaluation; correctness is checked only after completion, using the deterministic replay target.

**Python tool-use prompt.** In this setting, GPT-5.2 is explicitly instructed to translate the problem into a short Python/SymPy computation when useful. It may issue one executable Python block, inspect the resulting stdout/stderr, and then produce a final answer. The environment provides standard Python arithmetic plus SymPy routines for algebraic simplification, equation solving, modular arithmetic, gcd/lcm, and numerical geometry checks. It does not provide any VDE-specific package, theorem library, graph parser, replay checker, or stored instance metadata.

**Coding-agent setting.** The coding-agent setting gives the model a lightweight iterative programming loop rather than a single tool call. The agent may create or edit a temporary Python script, run it, inspect failures or intermediate outputs, and revise the script within a fixed budget of five executions. This setting is intended to approximate a stronger code-assisted solver that can debug formalization mistakes. The agent still sees only the rendered statement and final-answer instruction; it does not receive the executable graph, theorem-node records, hidden intermediate values, branch annotations, replay trace, generated candidates, or answer key.

Table 8: Tool-use and coding-agent baselines on VDE. Entries report accuracy (%) under the same final-answer checking protocol as the main experiments.

| Setting | Geo-E | Geo-M | Geo-H | NT-E | NT-M | NT-H |
|---|---|---|---|---|---|---|
| Direct GPT-5.2 | 75.0 | 63.7 | 55.4 | 82.6 | 73.4 | 66.8 |
| GPT-5.2 + Python tool-use prompt | 79.2 | 66.5 | 57.6 | 88.4 | 80.1 | 73.2 |
| GPT-5.2 coding agent | 81.6 | 68.9 | 59.7 | 90.2 | 82.4 | 75.3 |

**Interpretation.** Tool access helps most when the main bottleneck is arithmetic, symbolic simplification, modular computation, or numerical verification. This explains the larger gains in number theory than in analytic geometry. However, the Hard buckets remain far from saturated: the coding-agent setting is still below 60% on geometry Hard and below 76% on number-theory Hard. The remaining failures are concentrated in translating the rendered statement into the right computational representation, choosing the correct branch among multiple feasible constructions, preserving global constraints across derived quantities, and recognizing which relations are givens rather than consequences to prove. These are precisely the graph-level dependencies targeted by VDE's construct-depth, constraint, and branching controls; they are not removed by merely adding a calculator or code executor.

**Protocol safeguards.** The tool-enabled systems are evaluated as solvers of the natural-language problem, not as consumers of VDE internals. Any programmatic solution must reconstruct a useful formalization from the rendered statement alone, and final correctness is still judged by the same deterministic replay target used for all models.

# E External Structural Projection

The structural comparison in Table 3 is computed from sampled external datasets using a presentation-level projection. The goal is to compare visible structural features, not to convert external problems into executable VDE instances.

**Sampling sources.** We include all available geometry and number-theory examples from the external sources used in the comparison: JGEX-231 ($N = 231$), GeoQA train split ($N = 3503$), MATH geometry ($N = 1349$), MATH number theory ($N = 1409$), OlympiadBench geometry ($N = 129$), and OlympiadBench number theory ($N = 127$). For VDE, we use 500 generated instances per domain and difficulty tier. When a source provides multiple splits, Table 3 reports the split used for the main comparison; the projection code also records split metadata.

**Projection procedure.** For natural-language datasets, we first segment each problem into clauses using punctuation and line breaks. We extract mathematical symbols, numbers, and domain keywords as value nodes with coarse types such as Point, Line, Circle, Integer, Real, Predicate, and Answer. Clauses containing construction verbs such as "construct", "define", "intersect", "midpoint", "reflect", or "choose" are mapped to construct operators. Clauses containing relation words such as "parallel", "perpendicular", "congruent", "divides", "modulo", "gcd", or "lcm" are mapped to constraint operators. Query clauses are mapped to query operators. For formal geometry resources such as JGEX-231, we parse the provided formal program lines directly: construction commands become construct operators, and goal or relation predicates become constraint operators.

**Metrics.** After projection, we report mean graph statistics over each dataset: **V** is the number of value nodes; **T** is the number of operator nodes; **Given** counts value nodes without producers; **Derived** counts non-given values; **Construct** counts operators classified as constructive; and **Constraint** counts operators classified as relational or predicate constraints. The projection is intentionally coarse and does not check theorem executability, branch feasibility, or answer correctness. It is therefore used only for structural profiling; all VDE answer verification relies on the executable graph and deterministic replay described in the main method.

# F Case Studies

To give a clearer qualitative sense of the two domain instantiations, we include one analytic-geometry example and one number-theory example below. These case studies are manually polished from the same executable-graph primitives used in VDE. Each example includes the replay-verified answer, a compact derivation, and an observed GPT-5.2 failure under the same final-answer evaluation style used in our experiments.

## F.1 Geometry Case Study

In the Cartesian plane, let $P_1 = (4, 5)$, $P_2 = (-5, 4)$, $P_3 = (-2, 4)$, $P_4 = (1, 2)$, $P_5 = (2, -3)$, $P_6 = (2, 1)$, and $P_7 = (-4, -3)$.
Let $C_1$ be the circle with diameter $P_4P_1$. Let $C_2$ be the circumcircle of triangle $P_4P_2P_3$. These two circles meet at $P_4$ and at one other point $U$.
Let $C_3$ be the circle with diameter $P_6P_5$. Let $C_4$ be the circle with diameter $P_6P_7$. These two circles meet at $P_6$ and at one other point $V$.
Find the exact value of the distance $UV$.

**Replay-verified answer.** $\dfrac{19\sqrt{305}}{61}$.

**Replay sketch.** The executable construction gives

$$C_1 : x^2 - 5x + y^2 - 7y + 14 = 0, \qquad C_2 : x^2 + 7x + y^2 + 3y - 18 = 0.$$

Their common points are $P_4 = (1, 2)$ and $U = (46/61, 140/61)$. Similarly,

$$C_3 : x^2 - 4x + y^2 + 2y + 1 = 0, \qquad C_4 : x^2 + 2x + y^2 + 2y - 11 = 0,$$

whose common points are $P_6 = (2, 1)$ and $V = (2, -3)$. Therefore

$$UV^2 = \left(\frac{46}{61} - 2\right)^2 + \left(\frac{140}{61} + 3\right)^2 = \frac{1805}{61},$$

so $UV = 19\sqrt{305}/61$.

**Observed GPT-5.2 failure.** In one zero-shot run with the same final-answer format, GPT-5.2 constructed $C_1$ and $C_2$ but propagated the wrong radical axis, using $x + y = 3$ instead of the replay-verified axis $6x + 5y = 16$. It then set

$$U = (1 + \sqrt{2},\ 2 - \sqrt{2}), \qquad V = (2, -3),$$

and returned

$$\texttt{Final Answer:}\quad \sqrt{30 - 12\sqrt{2}}.$$

The replay checker rejects this before final-distance evaluation: substituting the proposed $U$ into $C_1$ leaves residual $4$, and substituting it into $C_2$ leaves residual $4 + 2\sqrt{2}$. Thus the failure is not a final simplification issue, but an intermediate object/constraint propagation error in a multi-intersection construction.

### F.2 Number Theory Case Study

Let $x$ be the unique integer with $0 \le x < 7431799$ satisfying

$$x \equiv 7 \pmod{43}, \quad x \equiv 11 \pmod{47},$$
$$x \equiv 13 \pmod{59}, \quad x \equiv 17 \pmod{61}.$$

Let $r$ be the residue with $0 \le r < 211$ such that

$$r^2 \equiv x \pmod{211}, \qquad r \equiv 1 \pmod{8}.$$

Let $u$ be the inverse of $x - 3r$ modulo $223$, so that

$$(x - 3r)u \equiv 1 \pmod{223}, \qquad 0 \le u < 223.$$

Let $w$ be the inverse of $x + r + u$ modulo $227$, so that

$$(x + r + u)w \equiv 1 \pmod{227}, \qquad 0 \le w < 227.$$

Let $v$ be the least nonnegative solution of

$$(r + 2u + w)v \equiv x \pmod{229}.$$

Finally define

$$\begin{aligned} T = {} & \gcd(x - r, 30030) + \gcd(x + u, 39270) \\ & + \operatorname{lcm}(r + 11, u + 13) \\ & + \operatorname{lcm}(v + 17, w + 19) \\ & + ((x - r - u - v - w) \bmod 131), \end{aligned}$$

where the final modular term denotes the least nonnegative residue modulo $131$.
Find $T$.

**Replay-verified answer.** $6444$.

**Replay sketch.** Chinese remaindering gives $x = 1230753$. Modulo $211$, the square-root branches are $74$ and $137$, and the branch constraint $r \equiv 1 \pmod 8$ selects $r = 137$. Then

$$u = (x - 3r)^{-1} \bmod 223 = 35, \qquad w = (x + r + u)^{-1} \bmod 227 = 26,$$

and the linear congruence gives $v = 84$. The final components are

$$\gcd(x - r, 30030) = 2, \quad \gcd(x + u, 39270) = 2,$$
$$\operatorname{lcm}(r + 11, u + 13) = 1776, \quad \operatorname{lcm}(v + 17, w + 19) = 4545,$$

and $((x - r - u - v - w) \bmod 131) = 119$. Therefore $T = 2 + 2 + 1776 + 4545 + 119 = 6444$.

**Observed GPT-5.2 failure.** With a concise prompt asking for only key intermediates, GPT-5.2 returned the following trace:

$$x = \mathbf{1256770}, \quad r = \mathbf{192}, \quad u = \mathbf{52}, \quad w = \mathbf{186}, \quad v = \mathbf{215},$$
$$(\mathbf{66},\ \mathbf{2},\ \mathbf{13260},\ \mathbf{4719},\ \mathbf{65}) \text{ as the five summands,}$$
$$\texttt{Final Answer: } \mathbf{22112}.$$

Deterministic replay rejects the trace immediately: the CRT solution should be $x = \mathbf{1230753}$, so all downstream modular inverses and aggregation terms are computed from the wrong branch of the executable graph. This illustrates a global dependency-chain failure rather than a mere final-format error.

## G Algorithm

**Algorithm 1** Reverse expansion for constructing a theorem skeleton

**Input:** target value node $v_\star$; theorem library $\mathcal{L}$; theorem budget $B$
**Output:** skeleton construct graph $G_c = (V \cup T_c, E_c)$ with givens $G \subseteq V$
Initialize $V \leftarrow \{v_\star\}$, $T_c \leftarrow \emptyset$, $E_c \leftarrow \emptyset$
Initialize unresolved set $U \leftarrow \{v_\star\}$ and producer map $\mathbf{prod}(\cdot) \leftarrow \varnothing$
**while** $U \neq \emptyset$ **and** $|T_c| < B$ **do**
  Select an unresolved node $v \in U$
  Sample a theorem schema $o \in \mathcal{L}$ and an output slot $y \in \mathbf{Out}(o)$ such that $\tau(y) = \tau(v)$
  Instantiate $o$ by binding $y \mapsto v$
  $T_c \leftarrow T_c \cup \{o\}$; $\mathbf{prod}(v) \leftarrow o$; $E_c \leftarrow E_c \cup \{(o, v)\}$
  **for each** input slot $x \in \mathbf{In}(o)$ **do**
    Create or reuse a value node $u$ with type $\tau(u) = \tau(x)$
    $V \leftarrow V \cup \{u\}$; $E_c \leftarrow E_c \cup \{(u, o)\}$
    **if** $\mathbf{prod}(u) = \varnothing$ **then**
      $U \leftarrow U \cup \{u\}$
    **end if**
  **end for**
  $U \leftarrow U \setminus \{v\}$
**end while**
Declare givens $G \leftarrow \{v \in V : \mathbf{prod}(v) = \varnothing\}$
**return** $G_c = (V \cup T_c, E_c)$ and givens $G$

# H Prompt

## H.1 Problem Presentation

You are given a structured *presentation graph* of a math problem. Your job is to write a single, self-contained natural-language problem statement that uses *only* the provided givens and relations, and asks for the designated target. The statement must be solvable from the information given and must not rely on any hidden steps.

**Input :**

- **Typed givens:** a list of objects with types and (if present) numeric values.
- **Selected relations:** a list of constraints/relations that hold among the objects.
- **Target:** the value/object the solver must determine.
- **Optional:** units, coordinate system, domain restrictions, and non-degeneracy conditions.

**Output requirements:**

1. Output *only* a single, self-contained problem statement. Do not include a solution, hints, meta-commentary, or any extra formatting (no bullet lists, no headings).
2. Do *not* expose implementation artifacts: do not mention graphs, nodes, theorem names, internal IDs.
3. Ensure **solvability and completeness**: the statement must contain all givens and relations necessary to uniquely determine the target (or to match the intended solution multiplicity), without relying on implicit or hidden assumptions.
4. Ensure **clarity and unambiguity**: define all objects unambiguously (including the ambient space such as the Cartesian plane), avoid overloaded language, and use standard mathematical terminology and notation.
5. Ensure **faithfulness**: use *only* the provided typed givens and selected relations; do not introduce new objects, constructions, constraints, or auxiliary quantities beyond benign naming.
6. Use **consistent human-friendly naming**: if multiple objects of the same type appear, assign conventional symbols (e.g., points $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$; lines $\boldsymbol{\ell_1}, \boldsymbol{\ell_2}$; circles $\boldsymbol{\omega_1}, \boldsymbol{\omega_2}$) and reference them consistently throughout.
7. State **non-degeneracy conditions** *only when needed* to prevent ambiguity or pathological cases (e.g., distinct points for a line, two distinct intersections, non-parallel/non-coincident lines), and do not add stronger assumptions than necessary.
8. Specify the **query and answer form**: clearly ask for the designated target, and when the target is numeric, indicate whether an exact simplified form is expected (preferred) or a decimal approximation is acceptable.

**Now generate the problem statement from the following structured input.**

**[TYPED GIVENS]**
`{TYPED_GIVENS_JSON}`

**[SELECTED RELATIONS]**
`{RELATIONS_JSON}`

**[TARGET]**
`{TARGET_JSON}`

## H.2 Evaluation

Solve the following math problem. You must show a complete, step-by-step solution, and then provide the final answer in the required format.

**Instructions:**

- Provide a clear derivation with explicit intermediate steps (do not skip key algebra/geometry steps).
- Use standard mathematical notation.

- End your response with *exactly one* final answer line in the form:

  `Final Answer: <answer>`

- The final answer must be unambiguous and simplified as much as possible.
- Do not include anything after the final answer line.

**Problem:**
`{PROBLEM_STATEMENT}`

**Begin your solution below.**

### H.3 Error Analysis

You are an evaluation analyst. Given a math problem, a model solution (steps + final), and a *deterministic replay* record of the numericized ground truth (target, replay intermediates, and verifiable relations), assign **exactly one** error label.

**Inputs:**

- **Problem**: the natural-language question.
- **Model response**: full derivation and final answer.
- **Replay record**: $v_\star$, intermediate values, and predicates (plus ambiguous-branch candidates when applicable).

**Output:** one label in {C1,C2,C3,C4} with brief replay-grounded evidence.

**Labels (choose one):**

**C1. Entity/condition confusion**: wrong target/object, swapped entities, or misread constraints.

**C2. Broken dependency chain**: missing/incorrect required intermediates or invalid step composition vs. replay.

**C3. Ambiguous-branch mismatch**: picked a plausible candidate locally, but it cannot satisfy all remaining relations jointly.

**C4. Wrong final**: derivation is consistent with replay, but final value/simplification/format is wrong.

**Decision order:** C1 $\rightarrow$ C2 $\rightarrow$ C3 $\rightarrow$ C4. If unsure, set `needs_manual=true`.

**Output format (JSON, no extra text):**

```
{
  "label": "C1|C2|C3|C4",
  "needs_manual": true|false,
  "evidence": [
    {"type": "quote", "source": "model", "text": "..."},
    {"type": "replay", "field": "relation|intermediate|target", "text": "..."},
    {"type": "explanation", "text": "One-paragraph justification grounded in the above."}
  ]
}
```

**Payload:**
**[PROBLEM]** `{PROBLEM_TEXT}`
**[MODEL_RESPONSE]** `{MODEL_TEXT}`
**[REPLAY_RECORD]** `{REPLAY_JSON}`

# I Human Guideline

You will be shown a problem statement and (optionally) the target query. Judge each criterion as **Pass** or **Fail** based only on the text.

**1. Statement correctness (Pass/Fail).**

- **Pass** if all objects/relations are mathematically well-defined and mutually consistent (no self-contradiction, undefined construction, or impossible configuration implied by the text).
- **Fail** if the statement contains contradictory constraints, ill-defined terms, missing definitions for referenced objects, or obvious degeneracies that make the setup invalid.

**2. Solvability (Pass/Fail).**

- **Pass** if the target is determined by the stated information (allowing the intended multiplicity if the statement explicitly permits multiple solutions, e.g., "find all" or "give both").
- **Fail** if the target is under-specified (multiple distinct answers without being stated), over-specified (inconsistent), or the target cannot be derived from the provided information.

**3. Hidden leakage (Pass/Fail).**

- **Pass** if the statement does *not* reveal intermediate computed values or solver-only artifacts (e.g., "let $x$ be the computed slope 3/2", giving a midpoint that was meant to be derived, or exposing coefficients that trivialize the construction).
- **Fail** if it directly provides intermediate results that should be derived, or includes implementation-like traces (IDs, internal variable names, replay values) that leak the graph execution.

**General instructions:**

- Use a strict standard: if you are genuinely unsure, mark **Fail** and add a short note explaining why.
- Do not solve the full problem unless needed to decide solvability; quick sanity checks and minimal derivations are sufficient.
- Ignore minor stylistic preferences unless they affect clarity or ambiguity.

**Output format:**

```
correctness: PASS/FAIL
solvability: PASS/FAIL
hidden_leakage: PASS/FAIL
notes: <optional 1-2 sentences>
```

# J Checklist Support and Additional Documentation

This appendix collects details requested by the NeurIPS checklist. It is intended to make the submission self-contained with respect to limitations, reproducibility, statistical uncertainty, resource use, impacts, and artifact documentation.

This section directly addresses the documentation requested by the NeurIPS checklist. The main paper states the claims, scope, method, experimental setup, and results; the appendix below adds the supporting details for graph-theoretic assumptions, limitations, reproducibility, statistical uncertainty, compute, societal impact, released assets, human inspection, and LLM usage. The paper does not release a trained model and does not use scraped personal or sensitive data.

## J.1 Theory Support for the Bipartite Graph Representation

The formal object used by VDE is the typed value–theorem bipartite graph defined in §3. The relevant assumptions are: (i) each value node has a type $\tau$ with domain $\mathcal{D}_\tau$; (ii) each theorem node has a typed input/output signature, an executable map, and predicate checks $\mathcal{C}_o$; (iii) the constructive subgraph is a DAG; (iv) every non-given value in the constructive skeleton has exactly one producer; and (v) all injected constraints are evaluated as predicates over the induced assignment or over branch candidates. These assumptions are stated in §3, §3.2, and Algorithm 1.

Under these assumptions, deterministic replay is well-defined. Since the constructive skeleton is acyclic, its theorem nodes admit a topological execution order. Since each non-given value has exactly one producer, no two constructive rules assign conflicting values to the same node. Starting from the stored givens, executing theorem nodes in topological order therefore induces a unique replay assignment for all single-valued nodes and a finite branch set for set-valued nodes. Constraint predicates then prune this branch set without changing the constructive dependencies. A released instance is accepted only when at least one branch satisfies all predicates and the designated target $v_\star$ has the stored replay target. Consequently, answer verification is model-independent: the evaluated model is compared against the value obtained by replaying the stored graph, not against an LLM-written solution or judge. This proves the representation property used by the benchmark: if the theorem executors and predicates are sound for their typed domains, then replay-valid instances have auditable, reproducible ground-truth answers.

## J.2 Limitations

VDE is a framework for replay-verifiable dynamic evaluation rather than a complete replacement for all mathematical reasoning benchmarks. The present instantiation covers analytic geometry and number theory; other areas such as algebraic combinatorics, calculus, probability, and proof-oriented olympiad mathematics would require additional type systems, theorem libraries, renderers, and validation filters. The quality of generated instances depends on the soundness and coverage of the theorem library. We mitigate this by manually filtering theorem schemas and by replay-validating every released instance, but the library may still bias the benchmark toward constructions that are easy to encode.

The framework assumes that the stored executable graph is the source of ground truth. This makes final-answer verification reliable, but it does not by itself certify that every natural-language rendering is optimally clear or pedagogically natural. We therefore combine automatic replay checks with a stratified human inspection protocol, although the audit is still not an exhaustive review of every generated instance. The evaluated model set is also a snapshot of currently available systems and does not establish conclusions about all future model families. Finally, dynamic generation reduces contamination risk but does not eliminate all benchmark-specific adaptation: once a generator is public, models may be trained to exploit its distribution.

## J.3 Reproducibility and Open Artifacts

The submission package is organized to support three levels of reproducibility:

1. **Instance verification:** each generated problem stores its typed value–theorem graph, givens, theorem-node records, injected constraints, presentation graph, rendered statement, and

replay-derived target. Re-running deterministic replay from the stored graph must recover the target and satisfy all predicates.

2. **Benchmark regeneration:** the generator specifies the domain, theorem library, random seed, construct-theorem budget $|T_c|$, constraint budget $|T_k|$, branching-node count $B$, numeric filters, and rendering prompt. These fields are sufficient to regenerate the reported Easy/Medium/Hard distributions up to nondeterminism from LLM-based surface rendering.
3. **Model evaluation:** the evaluation prompt in Appendix H.2 fixes the zero-shot solving protocol and final-answer format. Accuracy is computed by comparing extracted final answers against deterministic replay targets, with exact symbolic matching when available and numeric tolerance only for real-valued geometry quantities.

The supplementary artifact should contain: source code for graph generation and replay validation; theorem-library specifications for analytic geometry and number theory; rendered benchmark splits; raw model responses or answer-extraction records when redistribution is allowed by provider terms; scripts for computing the tables and figures; and a README with the software environment, command sequence, and expected output files. Closed-source model outputs may be subject to provider terms; when full response redistribution is restricted, the artifact should include hashes, extracted answers, aggregate scores, and the exact query protocol.

### J.4 Artifact Contents and New Assets

The submission introduces new benchmark assets rather than a trained model. The supplementary artifact contains six VDE split files under `data/release_corpus`: `geometry_easy.json` (500 instances, 5.94 MiB), `geometry_medium.json` (500, 7.65 MiB), `geometry_hard.json` (500, 8.52 MiB), `number_theory_easy.json` (500, 4.97 MiB), `number_theory_medium.json` (500, 7.72 MiB), and `number_theory_hard.json` (496, 11.20 MiB). The released corpus therefore contains 2,996 replay-validated instances and occupies 45.99 MiB as JSON. The number-theory hard split contains 496 accepted instances because four generated candidates failed the final validation/filtering requirements and were not backfilled after the release freeze.

Each instance records the domain, difficulty metadata, generation seed or used seed, theorem-library references, constructive and constraint budgets, branching setting, full executable graph, presentation graph, rendered problem statement, replay-derived target, and validation status. The code asset contains the graph-construction, constraint-insertion, forward-replay, presentation, and serialization modules. The prompt assets are reproduced in Appendix H.1–H.3. No personal data or human-subject records are included in these new assets; consent is therefore not applicable to the generated corpus. The release README should include the license chosen for the new code and generated data, dependency versions, and exact commands for replay validation and table regeneration.

### J.5 Experimental Details

No model is trained or fine-tuned in this work. The main experiments evaluate existing models in a zero-shot setting using a shared prompt, with no task-specific demonstrations. Each VDE domain is split into Easy, Medium, and Hard buckets using the same tiering variables described in §4: construct depth $|T_c|$, constraint budget $|T_k|$, and branching-node count $B$. Unless otherwise stated, each VDE domain–difficulty bucket contains 500 generated instances. Static benchmark comparisons use the public evaluation subsets described in the experiment section and are evaluated with the same final-answer extraction protocol where applicable.

For answer checking, symbolic quantities are canonicalized before comparison. Numeric geometry answers are compared after deterministic replay with a fixed tolerance chosen to absorb floating-point formatting differences, not mathematical slack. Invalid model responses, missing final-answer lines, or outputs that cannot be normalized to the required answer type are counted as incorrect. Error analysis is performed only on incorrect cases and uses the replay-grounded categories in Appendix H.3.

**Cross-seed evaluation.** We generated three independent VDE snapshots using the same theorem library, generation configuration, and difficulty definitions, with 100 instances per domain–difficulty bucket. Across the three snapshots, all evaluated models preserved the Easy-to-Hard accuracy

degradation, and the aggregate model ordering remained stable. Individual buckets showed minor local ranking fluctuations without changing the main conclusions.

### J.6 Statistical Uncertainty

Reported accuracies are proportions over finite evaluation sets. For a bucket with $n$ examples and observed accuracy $\hat{p}$, we compute an approximate 95% binomial confidence interval using

$$\hat{p} \pm 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}. \tag{10}$$

This interval is used as descriptive uncertainty for sampling variability over the evaluated problem set; it does not account for provider-side model nondeterminism or future model updates. For VDE buckets with $n = 500$, the worst-case half-width occurs near $\hat{p} = 0.5$ and is about 4.4 percentage points. For AIME-style sets with $n = 30$, the corresponding worst-case half-width is about 17.9 percentage points, so small differences on that benchmark should be interpreted cautiously. The Easy-to-Hard degradation trends in VDE are substantially larger than this uncertainty band for most evaluated models.

### J.7 Compute Resources

VDE generation and replay validation are CPU-oriented symbolic and numeric workloads. GPU resources are not required for graph construction, replay, answer checking, or table generation. The release-preparation run used a commodity x86-64 Windows workstation with 20 logical CPU threads and 32 GB RAM. The software environment used Python 3.12.7 with SymPy 1.13.3, NumPy 1.26.4, pandas 2.2.2, and OpenAI Python SDK 2.33.0. The released corpus contains 2,996 accepted instances across six JSON split files (45.99 MiB total). Per candidate, the generator uses the stored tier configuration, including a maximum of 120 resampling attempts, 28 skeleton/forward attempts, and a 6.0 second timeout for each forward attempt before rejecting the candidate or moving to a new seed. Peak memory use is below 16 GB for the reported release run.

The CPU time is modest compared with LLM querying. Generating and replay-validating the six released splits requires 14 CPU-hours in total on the 20-thread workstation, corresponding to 2.1 wall-clock hours with parallel split-level jobs. Geometry easy/medium/hard take 10/18/25 minutes, and number-theory easy/medium/hard take 12/25/45 minutes, with the hard number-theory split slower because more candidates are rejected by branch and predicate filters. Replay validation and answer-key serialization require an additional 18 minutes for the full released corpus.

LLM API calls dominate monetary cost. The primary released rendering requires one renderer call per accepted instance (2,996 calls). The main zero-shot VDE evaluation requires one solver call per model–instance pair; for the six models in Table 2, a full VDE-only evaluation uses 17,976 solver calls before retries. We use at most one retry for transient provider failures; the main evaluation uses 18,214 solver requests including retries. Including rendering, main evaluation, answer extraction, multi-renderer robustness, tool-use baselines, and replay-grounded error labeling, the reported study uses 27,600 total LLM/API requests. The total API cost is US$1,800 for the model mix used here. No model training or fine-tuning is performed.

### J.8 Broader Impacts and Safeguards

The main positive impact of VDE is to improve transparency and robustness in mathematical reasoning evaluation. Replay-verifiable dynamic benchmarks can reduce reliance on contaminated static test sets and on unverifiable model-written solutions, helping researchers identify failures in composition, consistency, and symbolic calculation. The framework may also support education-oriented diagnostics by generating controlled families of problems.

Potential negative impacts are mostly benchmark-governance risks rather than direct safety risks. Public generators may encourage overfitting to the generator distribution, leading to inflated benchmark progress without corresponding gains in broader mathematical reasoning. Synthetic problems can also underrepresent mathematical styles that are difficult to encode as executable graphs. If used for educational placement or automated assessment, errors in rendering, filtering, or answer normalization could unfairly affect users. We mitigate these risks by releasing replay traces, generation

metadata, human-audit guidelines, and clear scope limitations. VDE does not release a trained model, personal data, scraped sensitive content, or a high-risk generative system; therefore, no additional model-access safeguards are required.

The benchmark should not be used as a sole decision mechanism for student placement, employment screening, or high-stakes certification. Its intended use is research evaluation and diagnostic analysis of mathematical reasoning systems. Because the corpus is synthetic, the main fairness concern is not demographic representation but coverage of mathematical styles: domains not encoded in the theorem library may be underrepresented. We therefore report the covered domains explicitly, separate generated assets from external benchmark references, and provide replay metadata so downstream users can audit or filter instances for their own evaluation setting.

### J.9 Asset Licenses and Terms

The work uses several categories of existing assets. Public benchmark references such as GSM8K, MATH, AIME-style problem sets, OlympiadBench, GeoQA, and related geometry resources are cited in the main text; users of the released artifact should consult the original dataset pages for their licenses and redistribution terms. We do not claim ownership of these external datasets and do not relicense them. Closed-source models are accessed through their providers under the providers' terms of service; the paper reports aggregate evaluation results and cites provider documentation or technical reports where available. Open-source model names and versions are reported in the experiment section and should be accompanied by their original model licenses in the released README.

New assets introduced by this work include theorem-library specifications, generation code, replay-verification code, generated VDE instances, prompts, and evaluation scripts. Their structure and released split sizes are documented in Appendix J.4. These assets should be released with an explicit permissive license when allowed by institutional policy and third-party dependencies. Each released generated instance includes or should include provenance metadata: domain, difficulty tier, random seed or generation identifier, theorem-library version, graph specification, replay target, renderer prompt version, and validation status. Since the benchmark is synthetic and generated from authored theorem libraries rather than human-subject data, consent is not applicable.

### J.10 Human Inspection, Ethics, and LLM Use

The manual quality inspection reported in the paper was conducted as an internal expert review of generated mathematical statements. Annotators judge statement correctness, solvability, and hidden leakage using the full instruction text in Appendix I. The audit covers 120 instances: 20 instances from each domain–difficulty bucket across two domains and three difficulty tiers. The review interface consists of text fields showing the problem statement and, when needed, the target query; there is no separate graphical interface or image stimulus, so screenshots are not applicable. Annotators are internal researchers/authors or collaborators performing expert quality control as part of the research workflow. No external crowd workers are hired, and no separate piece-rate compensation scheme is used. No private personal data are collected, and no participant intervention or behavioral study is conducted; therefore, IRB-style human-subject review is not applicable.

LLMs are used as part of the benchmark construction and evaluation pipeline in three ways: rendering selected presentation graphs into natural-language problem statements, solving problems during model evaluation, and assisting replay-grounded error labeling for selected incorrect responses. The renderer is not trusted as a source of mathematical truth: every released instance is checked against the executable graph by deterministic replay, and rendered statements are filtered by automatic and human inspection. LLMs used for writing assistance, editing, or formatting do not affect the scientific claims unless explicitly described in the method.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's core contributions, namely a verifiable dynamic evaluation framework, constraint-based difficulty control, and empirical validation; these claims are supported by the method and experiments sections, with scope limitations documented in Appendix J.2.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix J.2 discusses domain scope, theorem-library dependence, rendering risks, dynamic-generator overfitting, and the limited breadth of the empirical study.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The paper provides the formal assumptions behind the typed value–theorem bipartite graph representation, including typed domains, executable theorem schemas, acyclic constructive dependencies, unique producers for non-given values, and predicate constraints. Appendix J.1 gives the corresponding proof argument for deterministic replay and replay-grounded answer verification.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper specifies the generation pipeline, controllable difficulty factors, evaluation setup, model families, and appendix prompts; Appendix J.3 and Appendix J.5 further document artifact contents, replay validation, answer checking, and regeneration metadata.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The project repository is `https://github.com/LuYanjie12/VDE`. Appendix J.3 and Appendix J.4 document the code, data, replay traces, generated splits, scripts, and reproduction instructions.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The work does not train new models, and the paper describes the generated evaluation sets, difficulty buckets, evaluation protocol, and the frontier models that were queried for the main comparisons. Additional answer-checking and split details are given in Appendix J.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Appendix J.6 defines binomial confidence intervals for reported accuracies, states the assumptions, and gives representative half-widths for VDE and AIME-sized evaluation sets.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix J.7 provides a concrete compute profile: CPU-only generation/replay on a 20-logical-thread workstation with 32 GB RAM, Python/package versions, 2,996 accepted instances across 45.99 MiB of JSON, per-candidate timeout/resampling settings, 14 CPU-hours (2.1 wall-clock hours with parallel split jobs), and renderer/evaluator API-call counts and cost.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

Justification: The work evaluates models on synthetic, verifiable math problems, does not involve deceptive practices or sensitive human data, and Appendix J.8 and Appendix J.10 discuss responsible release, risks, and safeguards.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Appendix J.8 discusses positive impacts, negative benchmark-governance risks, possible misuse in educational or high-stakes assessment, mathematical-coverage fairness limitations, and mitigations through replay traces, metadata, audits, scope limitations, and usage cautions.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a high-risk generative model or scraped dataset; it introduces a synthetic evaluation benchmark for mathematical reasoning. Appendix J.8 explains why additional model-access safeguards are not required.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Prior models and external assets are cited, and Appendix J.9 documents the categories of existing assets, required license/terms handling, provider terms for closed-source models, and README requirements for released artifacts.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Appendix J.4 and Appendix J.9 document the new assets introduced by the work, including the six released split files, 2,996 replay-validated instances, theorem-library specifications, generation and replay code, prompts, evaluation scripts, provenance metadata, license expectations, and consent considerations.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

    Justification: Yes. The paper includes the full internal expert-review instruction text in Appendix I. Appendix J.10 gives the sample size and stratification, explains that the review interface consists only of text fields so screenshots are not applicable, and states that annotators are internal researchers/collaborators with no external crowd-worker or piece-rate compensation scheme.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve human-subject experimentation requiring IRB review; Appendix J.10 clarifies that the manual quality inspection is an internal expert review of generated statements.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The method and appendix explicitly describe the use of LLMs to render executable presentation graphs into natural-language problems, evaluate models, and support selected error labeling; Appendix J.10 states how these uses are validated and bounded.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.