symmetry

MDPI

Article

FGeo-GCG: Hybrid Validation-Enhanced Geometric Data Synthesis with Human-like Proof

Cheng Qin [1], Xiaokai Zhang [2,*], Yuchang Yang [2], Zhenhai Sun [2], Yang Li [2], Zhengyu Hu [2] and Tuo Leng [1,2,*]

[1] School of Future Technology, Shanghai University, Shanghai 200444, China; qincheng@shu.edu.cn
[2] School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China; yangyuchang@shu.edu.cn (Y.Y.); slim@shu.edu.cn (Z.S.); leeyoung628@shu.edu.cn (Y.L.); huzhengyu@shu.edu.cn (Z.H.)
* Correspondence: xiaokaizhang@shu.edu.cn (X.Z.); tleng@shu.edu.cn (T.L.)

Abstract

Euclidean plane geometry problem solving is a challenging benchmark for artificial intelligence because it requires complex diagram understanding, symbolic deduction, and multi-step reasoning. Constructing effective datasets for this task requires geometric instances that are realizable, non-degenerate, structurally diverse, and paired with human-like proofs. However, existing random or template-based generation pipelines often produce redundant, singular, or infeasible candidates, causing substantial computation to be spent before useful reasoning trajectories can be extracted. To address these limitations, we present FGeo-GCG, a hybrid geometric data synthesis framework built on the FormalGeo-V2 deductive engine. It formulates Geometric Configuration Generation as an incremental linear construction process that decomposes global constraint satisfaction into local construction steps, thereby pruning invalid branches during the generation process. To improve reliability and efficiency, FGeo-GCG combines two validation stages: a safe stochastic Jacobian-rank filter estimates whether local candidate constraints contribute independent algebraic restrictions, and progressive geometric validation checks whether the resulting partial construction remains realizable and non-degenerate. By encoding incidence-, metric-, and symmetry-related dependencies within unified constraint graphs, the framework also connects geometric data synthesis with structural symmetry analysis. Validated constraint graphs are then converted into problem instances through forward deduction, goal decomposition, and multi-dimensional complexity filtering, producing proof targets without manual annotation. Experiments show that the full validation pipeline reduces the failure rate for highly constrained instances. The resulting FGeo-GCG dataset contains more than 50,000 formally validated plane geometric configurations and provides engine-derived reasoning traces and targets for future training and evaluation of neuro-symbolic geometry problem-solving systems.

Keywords: formal mathematics; geometric configuration generation; formal verification; data synthesis

Check for updates

Academic Editor: Tamas Kiss

Received: 19 May 2026
Revised: 29 May 2026
Accepted: 12 June 2026
Published: 15 June 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Euclidean plane geometry problem solving (GPS) is difficult for current AI systems because the diagram, the formal premises, and the proof search all constrain one another. A solver must keep incidence, metric, and auxiliary-construction information consistent while deriving multi-step conclusions under an axiomatic system. Classical automated geometry

theorem proving addressed this problem with symbolic engines such as Wu's method [1] and Gröbner-basis-based reasoning [2]. These methods are logically rigorous, but the search space can grow quickly when many auxiliary constructions are possible. Recent Large Language Models (LLMs) [3] and neuro-symbolic systems have changed the empirical landscape, with systems such as AlphaGeometry 2 [4], GenesisGeo [5], InternGeometry [6], TongGeometry [7], and Seed-Prover [8–10] reporting strong results on IMO-style geometry without human demonstrations.

This progress has also shifted attention to synthetic data generation. Multimodal Large Language Models (MLLMs) [11] require examples in which formal premises, diagrams, and proof steps refer to the same underlying construction. Many recent pipelines therefore sample randomized geometric configurations and use them to build deductive databases [12]. In this setting, the premise generator is not a minor preprocessing component. If it produces degenerate, unrealizable, or nearly duplicate configurations, the later stages inherit those defects before any theorem or diagram is exported.

Existing generators often validate correctness and non-degeneracy only after a random sample has been formed. This late filtering creates a practical trade-off. Templates keep the search controlled but restrict the range of constructions; unconstrained random sampling explores a wider space but produces many candidates that are rejected only after symbolic processing. At scale, exhaustive algebraic checks and graph-isomorphism-based deduplication can become expensive, and exploring deductive closure via forward chaining [13] is NP-hard in general.

We address this issue with an algorithmic framework for Automated Geometric Configuration Generation (GCG) that combines formal reasoning with lightweight numerical validation. A central idea of our approach is to validate relations during construction. We embed abstract geometric configurations into a differentiable numerical space and use this representation to connect formal descriptions with numerical computation as the configuration is being built.

This leads to a Linear Construction Paradigm, in which configurations are expanded one relation at a time and checked locally as they grow. The check is based on Numerical Jacobian-Rank Verification: local topological constraints are written as differentiable parameterized equations, and the rank of the constraint Jacobian is used to estimate local Degrees of Freedom (DOFs) and detect singular configurations. By evaluating the Jacobian under random floating-point substitutions, the method obtains a lightweight probabilistic test for local independence. Relations that fail this test can be discarded early, leaving symbolic checks for fewer and better-conditioned candidates.

The paper is organized as follows. Section 2 compares the method with neuro-symbolic geometry solvers, synthetic data generation pipelines, and geometric constraint solving. Section 3 gives the generation algorithm and validation mechanism. Section 4 describes dataset construction, dataset analysis, and pipeline evaluation.

The main contributions of this paper are as follows:

1. We present a generation algorithm that combines formal logic with numerical kinematic checks to synthesize algebraically well-posed, non-degenerate geometric configurations while maintaining topological variation.
2. We integrate Numerical Jacobian Verification into an incremental generation loop to detect singular or locally redundant relations before invoking heavier constructive checks.
3. We show how the generated configurations can be converted into formally verified geometry problems with aligned symbolic statements, diagrams, and proof traces.

https://doi.org/10.3390/sym18061035

2. Related Work

Our work lies between three lines of research: neuro-symbolic geometry theorem proving, synthetic data generation for geometric reasoning, and geometric constraint solving. The point of departure is the stage at which validation is performed. Rather than improving inference after a complete problem has been written down, we ask whether a partially sampled premise is already algebraically non-redundant, geometrically realizable, and suitable for downstream deduction.

2.1. Neuro-Symbolic Geometry Problem Solving

Automated geometry reasoning has evolved from classical symbolic approaches, such as Wu's method [1], the area method [14], and rule-based theorem proving [15], to modern neuro-symbolic systems that combine learned heuristics with formal deduction. Representative recent systems include AlphaGeometry [16], which couples a language model with a deductive database and algebraic reasoning engine to propose auxiliary constructions, and AlphaGeometry 2 [4], which extends the formal language and substantially improves performance on Olympiad-style problems. GenesisGeo [5] further improves symbolic inference efficiency, while InternGeometry [6] explores agentic search strategies for difficult geometry tasks.

AlphaGeometry-style solvers assume that a theorem instance has already reached a form on which auxiliary-construction search and deductive closure are meaningful. FGeo-GCG targets the preceding failure mode: during sampling, many partial premises are already over-constrained, degenerate, or algebraically redundant before they can support any useful proof search. Improving the downstream prover therefore does not remove the need to reject such branches while the configuration is still being assembled.

2.2. Large-Scale Synthetic Data Generation for GPS

The demand for training data in geometric problem solving has led to several automated synthesis pipelines. A common strategy is to sample abstract geometric structures and then invoke symbolic checks to retain admissible instances. For example, the AlphaGeometry pipeline uses randomized graph generation followed by logical filtering; this design can still spend considerable computation on over-constrained, degenerate, or unrealizable candidates before they are rejected. Recent datasets and generation frameworks have broadened the target format of synthetic data: GeoGen [17] combines FormalGeo [18] with stepwise reasoning traces; GeoGPT4V [19] emphasizes consistency between diagrams and textual descriptions; SDE-GPG [20] introduces controllable generation via checking functions; and GeoFM [21] perturbs metric conditions to improve robustness against visual hallucination.

Trace-generation datasets such as GeoGen are most useful after a solvable formal instance can be replayed as supervised reasoning steps, while controllable generation pipelines such as SDE-GPG still rely on checking functions to decide whether completed candidates satisfy the desired constraints. FGeo-GCG moves this decision into the incremental construction loop: a sampled relation is tested before the generator commits to extending it into a full problem. This matters because rejected branches are cheaper to discard before diagram rendering, proof-target selection, or natural-language packaging.

Table 1 makes this stage distinction explicit. The relevant comparison is not only whether a system can solve or generate geometry problems, but when invalid configurations are detected and what artifact is available after validation. This distinction separates solver-side progress from generator-side pruning, which is the bottleneck addressed here.

Table 1. System-level comparison with representative geometry reasoning and synthesis pipelines.

Method/ System	Primary Goal	Output	Validation Strategy	Proof Trace	Scalability Focus
FormalGeo [18]	Formal geometry representation	Formalized problems and predicates	Formal parser, solver, and verifier	Formal deduction traces	Extensible formal infrastructure
AlphaGeometry/ AG2 [4,16]	Olympiad solving	Solved theorem instances	Deductive database and algebraic reasoning	Solver derivations	Search over auxiliary constructions
SDE-GPG [20]	Controllable problem generation	Solvable geometry problems	Deduction-engine checking functions	Available after solving	Controlled synthesis
GeoGen [17]	Data and reasoning traces	Formal/NL training examples	FormalGeo-based replay and checking	Stepwise traces	Dataset construction
GenesisGeo [5]	Efficient geometry solving	Formal solved problems	Symbolic inference engine	Solver derivations	Solver-side inference efficiency
FGeo-GCG (ours)	Configuration synthesis	Validated configurations, diagrams, GDL, proof targets	Jacobian pre-screen + constructive validation + replay	Engine-derived action traces	High-throughput generation and early pruning

2.3. Geometric Constraint Solving and Numerical–Symbolic Fusion

Our numerical filter is also related to geometric constraint solving (GCS). Classical GCS and automated generation systems often use graph-theoretic tools, including Laman-style sparsity analysis [22,23] and canonical labeling or graph-isomorphism software such as Nauty and Bliss [24,25], to analyze rigidity or remove topologically duplicate structures. Such filters are principled and often exact for their target structures, but FGeo-GCG needs a lighter online decision: after each newly sampled relation, the generator must decide whether the current partial premise has gained an independent constraint or has merely become redundant or singular. We therefore encode the current formal constraints as differentiable equations and use the Jacobian rank as a local pre-screening signal. Unlike neural relational models [26–28], the output is not a learned proof step or relation prediction, but an accept-or-reject signal that reduces the load on constructive validation and symbolic proof replay.

3. Geometric Configuration Generation Algorithm

This section gives the computational details of Automated Geometric Configuration Generation (GCG). The generator adds entities one at a time and treats each accepted relation as a constraint over continuous parameters. A Jacobian-rank-based algebraic filter is applied before the constructive geometric checks, so many locally redundant candidates are rejected before they reach the formal backend.

The generator separates numerical pre-screening from constructive validation. For each newly sampled entity, candidate relations are first tested against the local Jacobian rank, and only rank-increasing candidates are checked by FormalGeo-V2 against the committed partial configuration. The overall process is illustrated in Figure 1.

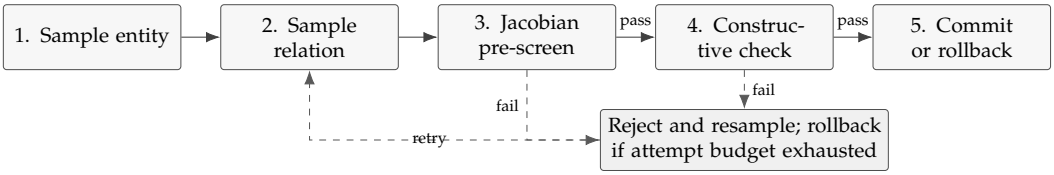

Figure 1. High-level flow of the GCG validation loop. Candidate relations are first screened by the stochastic Jacobian-rank test and then checked by the FormalGeo-V2 constructive backend. Accepted local relations are accumulated until the current entity is fully constrained; otherwise, the temporary branch is rolled back when the attempt budget is exhausted.

3.1. Problem Formulation and Definitions

Geometric Configuration (GC):

A geometric configuration is represented as a tuple $\mathcal{C} = \langle \mathcal{E}, \mathcal{R} \rangle$, where:

- $\mathcal{E}$ is a set of geometric entities (e.g., points, lines, and circles). Each entity $e \in \mathcal{E}$ has a Degree of Freedom (DOF), denoted by D_e, and is parameterized by a vector of continuous variables $\mathbf{P}_e \in \mathbb{R}^{D_e}$.
- $\mathcal{R}$ is a set of geometric relations or constraints (e.g., `PointOnLine` and `Perpendicular`). Each relation $r \in \mathcal{R}$ is mapped to one or more linear or nonlinear algebraic equations $\mathbf{f}_r(\mathbf{P}) = 0$ over the parameters of the entities involved in r.

The Generation Task:

The objective is to sample and construct a configuration $\mathcal{C}$ such that the induced global constraint system

$$\Phi(\mathbf{P}) := \{\mathbf{f}_r(\mathbf{P}) = 0 \mid r \in \mathcal{R}\} \tag{1}$$

is solvable, non-degenerate, and free of obvious redundant constraints.

3.2. Incremental Linear Construction Paradigm

To improve computational tractability, the generation process adopts an Incremental Linear Construction Paradigm [12]. Instead of solving the entire global constraint system at once—which is NP-hard in general—the configuration is built incrementally, entity by entity.

At step t, a new target entity e_t is introduced. Crucially, the parameters of all previously generated entities in the subgraph $\mathcal{E}_{<t} = \{e_1, \ldots, e_{t-1}\}$ are held fixed and treated as constants. The only active free variables in this step are the local parameters of the current target entity, denoted by $\mathbf{v} = \mathbf{P}_{e_t}$.

This construction keeps algebraic evaluation local to the parameters of the current target entity. The per-step cost is dominated by the number of Jacobian operations in dimension $|\mathbf{v}|$, which is typically small compared with the number of parameters in the full configuration.

This local strategy does not constitute a complete global satisfiability proof: each step is validated only with respect to the committed partial configuration. To guard against accumulated inconsistencies, the final serialized construction is replayed in FormalGeo-V2 before reasoning data are extracted.

3.3. FormalGeo-V2 as the Constructive Validation Backend

For FGeo-GCG, FormalGeo-V2 is used as a construction checker rather than as a full theorem-proving loop. Each call receives the committed partial configuration and one tentative local relation, then attempts to realize that relation under the GDL specification and the current object values. The call returns a feasibility flag without committing to the tentative relation; candidates reported as inconsistent or degenerate are rejected.

This check is deliberately placed after the Jacobian-rank filter. The rank test only asks whether the candidate adds an independent local restriction with respect to $\mathbf{P}_{e_t}$; FormalGeo-V2 then tests whether that candidate is constructible under the already-committed-to geometric state. The outline of FGeo-GCG is illustrated in Figure 2.

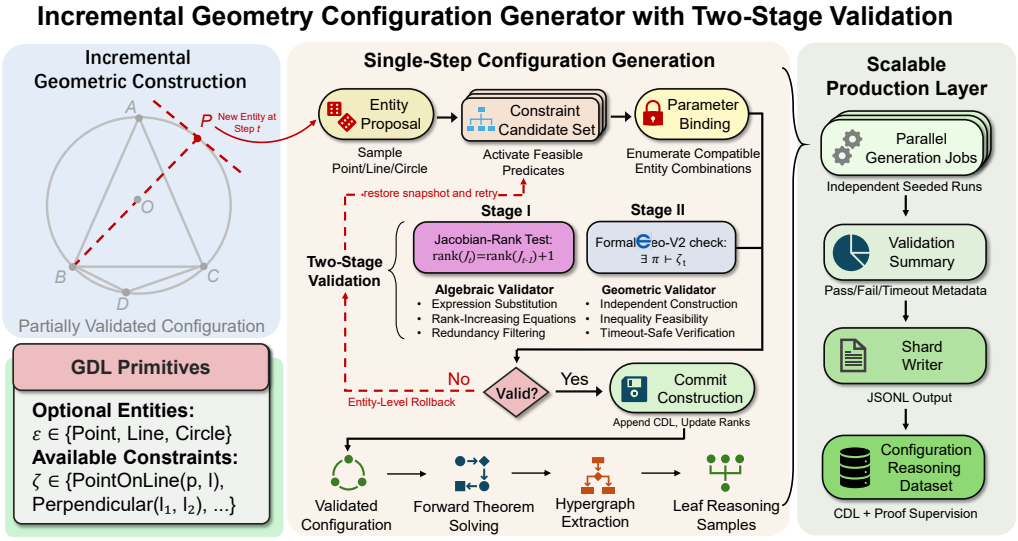

Figure 2. System architecture of the GCG pipeline. The generator constructs one entity at a time. Candidate relations are first screened by a local Jacobian-rank filter and then checked by FormalGeo-V2. Failed branches restore a lightweight snapshot of the mutable generation state.

3.4. Two-Stage Validation

Incremental construction needs to maintain algebraic non-redundancy and geometric validity without calling a full symbolic procedure on every sampled relation. We therefore use two validation stages.

3.4.1. Stage 1: Fast Algebraic Filter via Safe Stochastic Jacobian Check

Let $\mathbf{v} = \mathbf{P}_{e_t}$ be the free variables of e_t. Suppose a candidate relation r_{cand} yields local equations $\mathbf{f}_{cand}(\mathbf{v}, \mathbf{P}_{<t}) = 0$. We assess whether this relation contributes an independent restriction by evaluating the numerical rank of the local Jacobian matrix.

To evaluate the Jacobian robustly under randomly sampled parameter values, we use a randomized substitution operator $\xi : \mathbf{P} \to \Omega_B$. The bounded sampling domain Ω_B excludes values that make any denominator in the evaluated constraint expressions too close to zero:

$$\Omega_B = \left\{ \mathbf{p} \in [-10, 10]^d \mid \min_{q \in \mathcal{D}} |q(\mathbf{p})| \geq \epsilon \right\}, \tag{2}$$

where $\mathcal{D}$ denotes the set of denominator expressions appearing in the local constraints being evaluated. After substituting random floating-point values, we evaluate the numerical Jacobian matrix. Let $\mathbf{F}_{curr}$ denote the currently accepted local constraints for e_t. We accept r_{cand} only if it increases the local rank:

$$\operatorname{rank}\left(\left[\frac{\partial(\mathbf{F}_{curr} \cup \{\mathbf{f}_{cand}\})}{\partial \mathbf{v}} \right]_{\text{num}} \right) > \operatorname{rank}\left(\left[\frac{\partial \mathbf{F}_{curr}}{\partial \mathbf{v}} \right]_{\text{num}} \right) \tag{3}$$

This test serves as a fast probabilistic filter: it removes locally redundant constraints before invoking more expensive constructive validation.

Numerical Settings and Repeated Rank Estimation:

In the implementation, Safe Stochastic Substitution is used only as a numerical pre-screening step. Each active free variable is sampled independently from a bounded uniform interval, while samples whose distance to the nearest integer is smaller than ϵ are rejected and resampled. The default settings used in our experiments are summarized in Table 2.

Table 2. Default numerical settings for the stochastic Jacobian-rank filter.

Item	Default Setting	Role
Sampling distribution	Independent uniform sampling on $[-10, 10]$	Numerical substitution
Safe exclusion width	$\epsilon = 10^{-3}$	Reject near-zero denominators
Floating-point type	64-bit floating point	Numerical Jacobian evaluation
Rank computation	SVD-based numerical rank	Rank estimation
Rank tolerance	$\tau = \max(m, n)\epsilon_{\text{mach}}\sigma_{\max}$	Small singular-value cutoff
Repetitions	$K = 5$ substitutions per candidate	Robustness to accidental samples
Acceptance rule	Majority rank increase and Stage 2 pass	Candidate retention

For each candidate, the implementation draws $K = 5$ independent substitutions. A candidate is passed to Stage 2 only when the augmented Jacobian has larger numerical rank in at least three trials. Candidates that fail this majority rule are rejected or resampled.

Failure Modes of the Probabilistic Filter:

Rank errors can still occur near singular samples or under poor conditioning. The repeated-substitution rule reduces accidental decisions but does not make the rank test exact; candidates affected by unstable rank estimates may therefore be rejected during sampling.

3.4.2. Stage 2: Progressive Geometric Validation

While algebraic non-redundancy is necessary, it is not sufficient: geometric inconsistencies or degeneracies may still occur. Using the FormalGeo-V2 backend described above, we therefore invoke constructive validation progressively, including intermediate states in which the entity is not yet fully constrained (DOF > 0).

An under-constrained entity corresponds to a set of feasible placements. Stage 2 attempts a provisional construction under the current partial constraints. If these constraints are infeasible—for example, if a point is required to lie on two distinct parallel lines—the engine reports the conflict before the algebraic system reaches zero DOFs.

The final serialized configuration is later replayed during reasoning data extraction, so configurations that cannot be reconstructed from the recorded construction sequence are filtered out before dataset export.

3.5. The Core Generation Loop

The integrated generation loop is formalized in Algorithm 1. To make the procedural logic explicit, we first summarize its execution flow.

The loop uses the local rank k as the stopping criterion for the current entity. Since previously committed entities are fixed, accepted relations only need to increase rank with respect to $\mathbf{P}_{e_t}$. If k reaches D_{target}, the temporary relations are committed; otherwise they are discarded and the previously committed configuration is left unchanged.

Algorithm 1 Stochastic Geometric Configuration Generation via two-stage validation.

Require: Target complexity budget N_{target}, Max attempts T_{max}
Ensure: A valid Geometric Configuration $\mathcal{C} = \langle \mathcal{E}, \mathcal{R} \rangle$
Initialize $\mathcal{E} \leftarrow \emptyset$, $\mathcal{R} \leftarrow \emptyset$
while $|\mathcal{E}| < N_{target}$ **do**
  Sample new entity e_t, let $\mathbf{v} = \mathbf{P}_{e_t}$ and $D_{target} = \text{DOF}(e_t)$
  Initialize local rank $k \leftarrow 0$, $\mathbf{F}_{temp} \leftarrow \emptyset$, $T \leftarrow 0$
  while $k < D_{target}$ **and** $T < T_{max}$ **do**
    Sample valid candidate relation r_{cand} producing $\mathbf{f}_{cand}$
    // Stage 1: Safe Stochastic Jacobian Check
    Compute repeated numerical rank estimates over K safe substitutions using domain Ω, and set k_{new} by the majority rule
    if $k_{new} > k$ **then**
      // Stage 2: Progressive Geometric Validation (Early Pruning)
      if FormalGeo.ConstructiveCheck($\mathcal{E}, \mathcal{R} \cup \mathbf{F}_{temp} \cup \{\mathbf{f}_{cand}\}$) passes **then**
        Accept constraint: $\mathbf{F}_{temp} \leftarrow \mathbf{F}_{temp} \cup \{\mathbf{f}_{cand}\}$, $k \leftarrow k_{new}$
      else
        Reject r_{cand} (Early Pruned: Geometric Conflict)
      end if
    else
      Reject r_{cand} (Algebraically Redundant)
    end if
    $T \leftarrow T + 1$
  end while
  if $k == D_{target}$ **then**
    Commit e_t to configuration: $\mathcal{E} \leftarrow \mathcal{E} \cup \{e_t\}$, $\mathcal{R} \leftarrow \mathcal{R} \cup \mathbf{F}_{temp}$
  else
    Backtrack and discard e_t (Target DOF unattainable within T_{max})
  end if
end while
return $\mathcal{C}$

3.6. Reasoning Data Extraction from Generated Configurations

The generated configuration $\mathcal{C}$ is converted into reasoning supervision by reconstructing the serialized construction in FormalGeo-V2 and running forward deduction under the corresponding GDL specification. The resulting proof state is represented as a directed hypergraph, where nodes denote geometric facts and hyperedges denote deduction operations.

Candidate goals are selected from newly derived facts. For each goal, the module traces the proof hypergraph backward to the initial facts, topologically orders the required theorem-level operations, and stores the resulting goal–action pair. This stage provides the interface between GCG and dataset construction: GCG supplies realizable configurations, while proof extraction turns them into verified reasoning instances used in Section 4.

3.7. Implementation Optimization: Atomic State Snapshot and Transactional Rollback

Before sampling relations for a target entity, GCG records the mutable state needed to undo that entity-level attempt: the current entity count, cached local ranks, available relation pools, and the random-number-generator state. These fields are stored by `create_snapshot()` before the constraint-assignment loop starts.

If the attempt budget is exhausted before k reaches D_{target}, `restore_snapshot()` restores these fields. The tentative relations sampled for the current entity are discarded, while the previously committed entities and relations remain unchanged.

In practice, rollback stores incremental mutable states rather than full-constraint-graph copies. Its overhead is therefore proportional to the number of local mutable records touched by the current branch, not to the full size of the already validated configuration.

3.8. Implementation Optimization: Asynchronous Concurrency

In the parallel generator, each worker owns an independent random seed and runs the complete GCG loop locally. A worker outputs serialized construction records after they pass validation; it does not write JSONL shards directly.

Finished records are pushed to an IPC queue and consumed by a separate writer process. The writer batches records and appends them to persistent JSONL shards, keeping file flushing outside the inner sampling and Jacobian-validation loops.

4. FGeo-GCG Dataset Construction and Evaluation

The valid configurations $\mathcal{C}$ generated in Section 3 are stored as undirected constraint graphs before proof extraction. We convert these graphs into geometry problems with explicit premises and proof targets. This section describes a Hybrid Data Synthesis procedure that formulates problem extraction as a Target Constraint Satisfaction Problem and applies multi-dimensional complexity filtering to remove trivial instances. We then analyze the resulting FGeo-GCG dataset and evaluate the efficiency and reliability of the GCG pipeline used to produce it.

4.1. Hybrid Synthesis and Deductive Problem Extraction Framework

Template-based geometric data augmentation is limited by the constructions encoded in human-authored scripts. FGeo-GCG instead uses the FormalGeo-V2 deductive engine to derive candidate conclusions from generated configurations and extract proof-valid premise–target pairs through backward tracing of proof dependencies.

The generation process is formalized as a Target Constraint Satisfaction Problem (Target-CSP). For a given configuration graph $\mathcal{C} = \langle \mathcal{E}, \mathcal{R} \rangle$, the relations $\mathcal{R}$ form the initial premise constraints. We use a dual-directional reasoning mechanism:

1. Forward Deduction (FD): Starting from $\mathcal{R}$, the engine derives additional geometric conclusions using its axiom library (e.g., concyclicity, collinearity, proportionality) that are implied by the configuration but not explicitly stated in $\mathcal{R}$.
2. Goal Decomposition (GD): For a candidate target conclusion T, the engine decomposes the goal into prerequisite subgoals and then performs a backward search through the dependency structure to identify a subset of $\mathcal{R}$. This subset is used as the premises required to deduce T.

By mapping configurations to deductive Directed Acyclic Graphs (DAGs), we obtain the candidate set $\mathcal{P}_{cand}$. The overall pipeline is illustrated in Figure 3.

4.2. Complexity Evaluation and Filtering

Not all extracted premise–target pairs require multi-step reasoning. We define a quantifiable, multi-dimensional geometric complexity function $F_{complexity}(\mathcal{P})$ and use it to filter out trivial instances. The score is a heuristic ranking criterion and is not intended to approximate human olympiad difficulty.

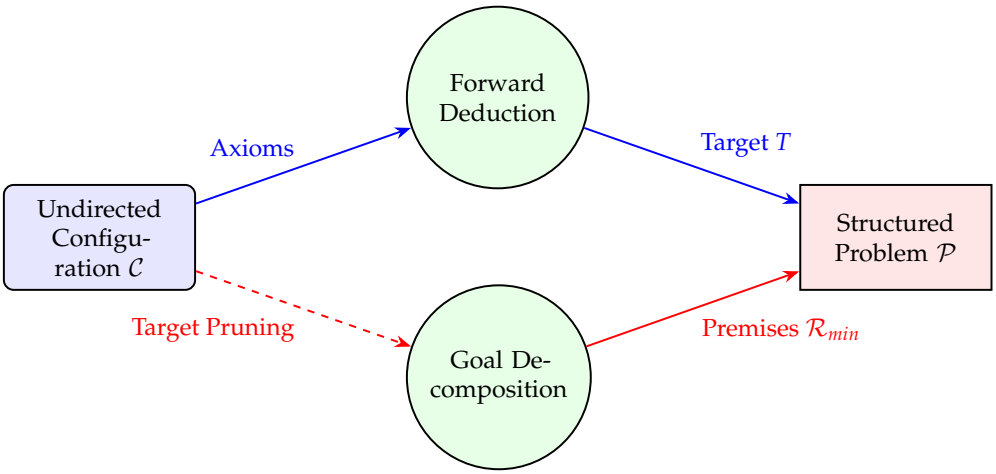

Figure 3. The hybrid synthesis and deductive problem extraction pipeline. An undirected geometric constraint graph $\mathcal{C}$ (left) is processed in two directions. Forward deduction (FD) derives candidate conclusions (top, blue arrows), while goal decomposition (GD) decomposes the target and performs a backward search to obtain a premise set (bottom, red arrows). The extracted instance (right) is a structured geometry problem.

The complexity metric incorporates three core dimensions:

- Number of Entities and Constraints (N_{ent}, N_{con}): A configuration with more geometric entities (e.g., points, lines, circles, and segments) and more relations among them generally induces a larger constraint graph. Higher entity and constraint counts mean that more objects participate in explicit geometric relations.
- Number of Newly Derived Facts (N_{fact}): Starting from the initial premise constraints, FormalGeo-V2 performs forward deduction and derives additional facts implied by the configuration. A larger number of newly derived facts leaves more intermediate statements available for subsequent target selection.
- Minimal Proof Depth (D_{proof}): This is defined as the length of the shortest path within the deductive DAG connecting the premise nodes to the target conclusion. A shallow path (e.g., $D_{proof} < 5$) indicates a direct application of basic axioms, while higher depths signify multi-step sequential logic.

For each candidate problem, the three dimensions are normalized by robust corpus-level reference values computed from the candidate pool:

$$\widehat{x} = \min\left(1, \frac{x}{q_{95}(x)}\right), \tag{4}$$

where $q_{95}(x)$ denotes the 95th percentile of the corresponding statistic among generated candidates. The final complexity score is then computed as

$$F_{complexity}(\mathcal{P}) = 0.25\widehat{N}_{ent} + 0.25\widehat{N}_{con} + 0.20\widehat{N}_{fact} + 0.30\widehat{D}_{proof}. \tag{5}$$

The largest weight is assigned to proof depth because the benchmark is primarily intended to evaluate multi-step deduction, while entity and constraint counts prevent the filter from selecting deep but structurally narrow traces. The weights are heuristic and are not tuned on a downstream test set; they bias the ranking toward proof depth while retaining structural terms from entities, constraints, and derived facts. We set the filtering threshold τ_{oly} to the empirical 70th percentile of $F_{complexity}$ on the candidate pool, and additionally require $D_{proof} \geq 6$ to remove direct one- or two-rule derivations. This cutoff keeps the upper-complexity subset of generated candidates without imposing a fixed, dataset-independent numerical threshold.

We apply the filter ($F_{complexity} \geq \tau_{oly}$) and remove candidates below the threshold. This produces a set of problems with denser constraint structures, more derived facts, and longer deductive chains without relying on manual annotation.

Auxiliary-Construction Extraction is treated as a lightweight annotation step rather than an independent optimization module. During proof-trace replay, if an executable deduction requires an intermediate geometric object that is not part of the visible premise set, the object is marked as a potential auxiliary construction and linked to the corresponding dependency in the proof DAG. This allows the generated dataset to include reasoning instances with auxiliary-construction cues, while leaving full auxiliary-construction planning and optimization to future system support.

4.3. Dataset Construction and Multi-Format Alignment

A geometry benchmark is easier to reuse when the symbolic record, text, diagram, and verification metadata refer to the same construction. We therefore provide aligned representations of each filtered problem for Large Language Models and Vision–Language Models (VLMs).

The internal representation of the filtered problem $\mathcal{P}$ is translated into parallel formats:

1. Natural Language (NL): A rule-based method converts premises $\mathcal{R}$ and target T into standardized natural-language text.
2. FormalGeo GDL: The symbolic entities, relations, and continuous parameter vectors $\mathbf{P}$ are serialized into FormalGeo Geometric Description Language statements and JSON-serialized states.
3. Diagram Rendering: The same parameterized geometric state is used to generate visual diagrams, such as TikZ-based figures or rasterized images, for vision–language evaluation.

This aligned multi-modal representation, illustrated in Figure 4, supports evaluation of both symbolic deduction and visual-text grounding.

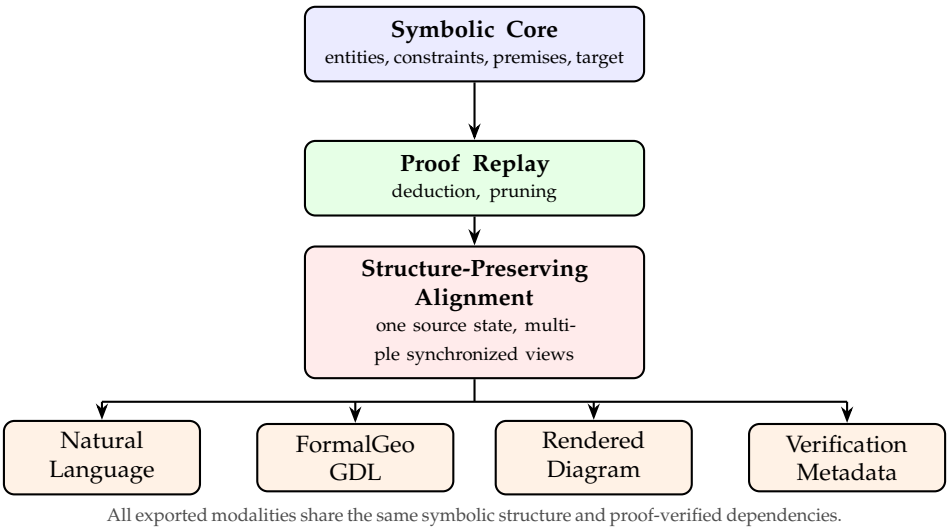

Figure 4. Multi-modal alignment subsystem. A filtered symbolic problem is replayed through the formal proof engine and then exported into aligned natural-language, FormalGeo GDL, diagram, and metadata views.

4.4. FGeo-GCG Dataset Distribution Characteristics

Using the proposed pipeline, we build the FGeo-GCG dataset. In this subsection, we give a quantitative overview and a descriptive comparison with existing benchmarks such as Geometry3K, GeoQA, FormalGeo7K, and IMO-AG. Unless otherwise stated, dataset-size statistics for prior benchmarks are taken from their original papers. Proof-depth values are

used only when proof traces or published annotations make a step count available; under this convention, one theorem, axiom, or solver rule application counts as one deductive step. For FGeo-GCG, proof depth is computed from the replayed deductive hypergraph after premise minimization. We count theorem-level `apply` and `decompose` actions, but exclude low-level algebraic simplification, numerical construction bookkeeping, orientation checks, and free-object declarations.

4.4.1. Dataset Overview

FGeo-GCG contains more than 50,000 formally validated problems in the current-generation snapshot after duplicate symbolic states and candidates below the complexity threshold are removed. For FGeo-GCG, D_{proof} is the shortest dependency path from the retained premises to the target conclusion in the replayed deductive DAG. As shown in Table 3, the snapshot is larger than several commonly used geometry benchmarks and has longer replayed proof traces under the counting convention stated above. Its topological structures are less concentrated on a small set of simple configurations, such as equilateral or isosceles triangles. The algebraic filter in Section 3 helps remove redundant constraints before proof extraction, which increases variation in the resulting graph structures.

Table 3. Comparison of large-scale geometric reasoning datasets. Volumes are taken from the cited dataset papers. Depth statistics use the number of deductive rule applications when proof traces are available; values marked with ≈ or inequalities are approximate summary values and are not exact cross-dataset measurements.

Dataset	Generation Method	Total Volume	Avg. Depth	Max Depth	Hidden Auxiliaries
Geometry3K [29]	Web Scraping + Heuristic	3002	≈3.2	8	No
GeoQA [30]	Web Scraping + Templates	5010	≈2.8	6	No
PGPS9K [31]	Web Scraping + Augmented	9022	≈4.1	11	Limited
FormalGeo7K [18]	Formal Annotation + Solver	7000	≈5.2	19	No
IMO-AG [16]	Hybrid Deductive	∼30	>15	>20	Yes
FGeo-GCG (Ours)	Hybrid Deductive + CSP	>50,000	>10.5	>25	Yes

4.4.2. Multi-Dimensional Comparative Analysis

The comparison is descriptive because the prior datasets do not all release proof traces in a directly comparable format. Figure 5 is therefore an illustrative schematic rather than a measured kernel density estimate. It summarizes the qualitative trend represented by the reported or estimated statistics: scraped datasets tend to concentrate at shallow proof depths, while complexity filtering shifts FGeo-GCG toward deeper replayed traces.

The dataset also reduces reliance on manual traces from scraped textbooks. Scraped data often reflects the frequency of common textbook proof patterns. In FGeo-GCG, targets are sampled from facts derived by the engine, so the selected problems follow the implemented axiom and theorem schemas rather than the distribution of a fixed textbook source.

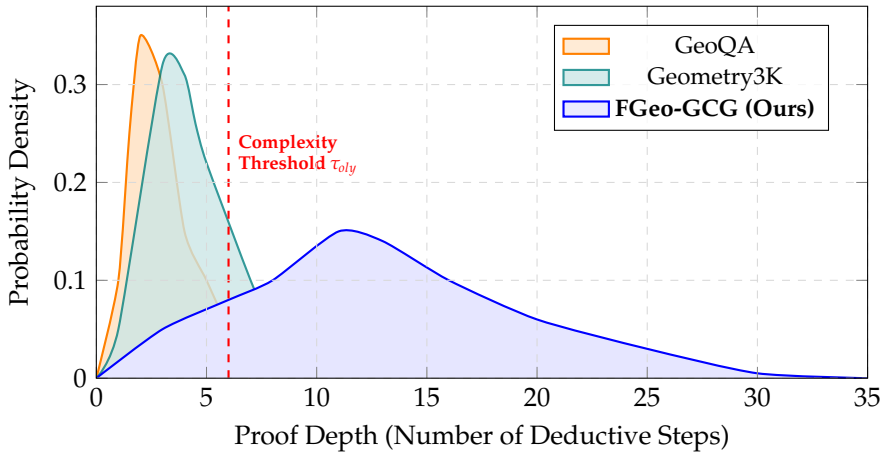

Figure 5. Illustrative proof-depth distribution schematic. This figure is not a measured KDE from released benchmark traces. It schematically summarizes the qualitative trend that scraped datasets tend to contain shallower proof traces, while complexity filtering shifts FGeo-GCG toward deeper proofs and a longer tail.

4.5. Generated Dataset Analysis

Beyond dataset-scale comparisons, we check four possible failure modes in the generated instances: collapse to a small set of entity types, concentration on a few predicates, weak growth in constraint density, and shallow proof supervision. These checks are meant to show what the generator actually produces after filtering, not only how many problems remain.

4.5.1. Entity and Relation Composition

Figure 6 checks whether the generator collapses to a small set of entity types as configurations grow. Across the reported entity-count bins, points, lines, and circles remain present in the generated instances rather than disappearing in larger configurations. The more visible change with scale is in the relation families, which shift from initialization-heavy relations in small cases toward a broader mix of incidence-, metric-, and circle-related constraints.

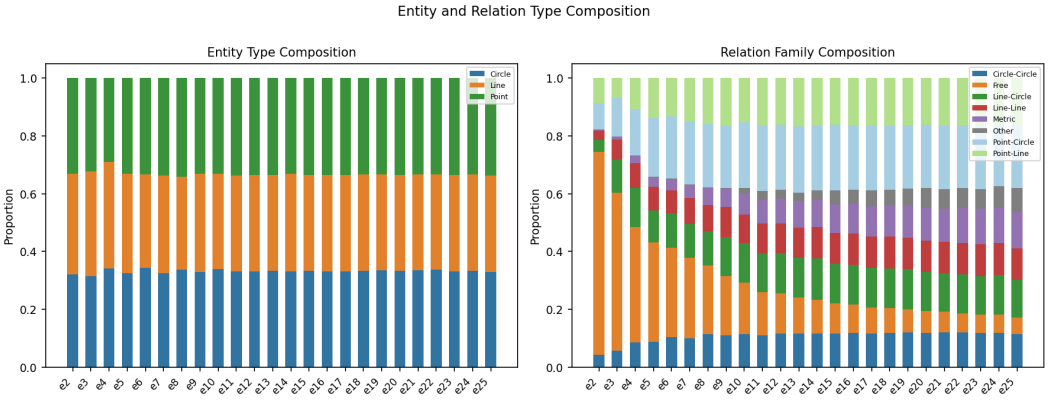

Figure 6. Entity and relation type composition. The generated instances maintain stable proportions of points, lines, and circles, while relation families become more diverse as entity count increases.

Small configurations contain a larger share of free constructions and simple line–circle or line–line relations, which are needed to initialize valid geometric objects. As the entity count increases, the share of free relations decreases, while metric, point–circle, point–line, circle–circle, and other relation families become more prominent. This pattern suggests that larger instances add constrained relationships, rather than simply appending isolated entities to an existing construction.

4.5.2. Predicate Coverage and Long-Tail Structure

Figure 7 reports the top constraint predicates and the rank–frequency curve. The highest-frequency predicates are construction scaffolds such as PointOnLine, LineTangentToCircle, and PointOnCircle; these predicates are common because they create the support objects used by later constraints. The frequent set is not limited to incidence predicates, however: it also includes parallelism, perpendicularity, circle concurrence, segment equality, midpoint, and free-line/free-circle constructions.

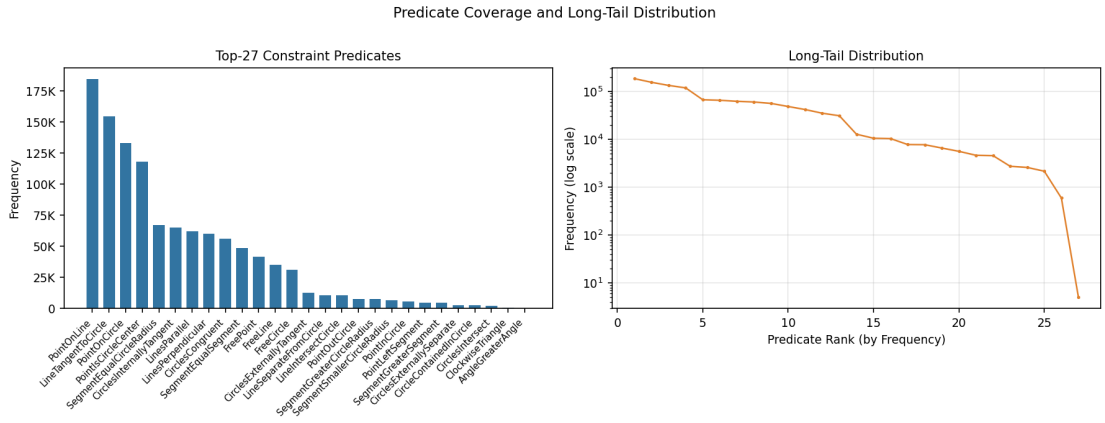

Figure 7. Predicate coverage and long-tail distribution. The generated dataset covers a broad set of geometric predicates. High-frequency incidence predicates provide construction scaffolds, while lower-frequency metric, angular, and intersection predicates form a long tail of harder constraints.

The log-scale rank plot shows a long-tail pattern: predicate frequency decreases gradually over several orders of magnitude rather than collapsing immediately after the most common relations. FGeo-GCG therefore contains both common construction primitives and less frequent specialized constraints, such as circle intersection, clockwise-triangle orientation, and angle comparison predicates. This reduces the chance that evaluation is dominated only by the top few construction templates.

4.5.3. Complexity Scaling with Entity Count

Figure 8 shows how generated configurations change as the number of entities increases. The number of constraints grows almost monotonically with entity count, rising from only a few constraints in small cases to roughly fifty constraints for the largest configurations. Together with the relation-family shift above, this shows that additional entities are usually integrated through new incidence, metric, and relational constraints.

The constraints-per-entity ratio also increases with entity count and approaches a dense regime near two constraints per entity. Meanwhile, the free-entity ratio decreases sharply: small configurations may contain many unconstrained or weakly constrained entities, whereas larger configurations leave only a small fraction of entities free. The validation-duration plot shows the computational cost of this denser structure: median validation time remains manageable, but larger configurations have a wider spread and more high-cost outliers.

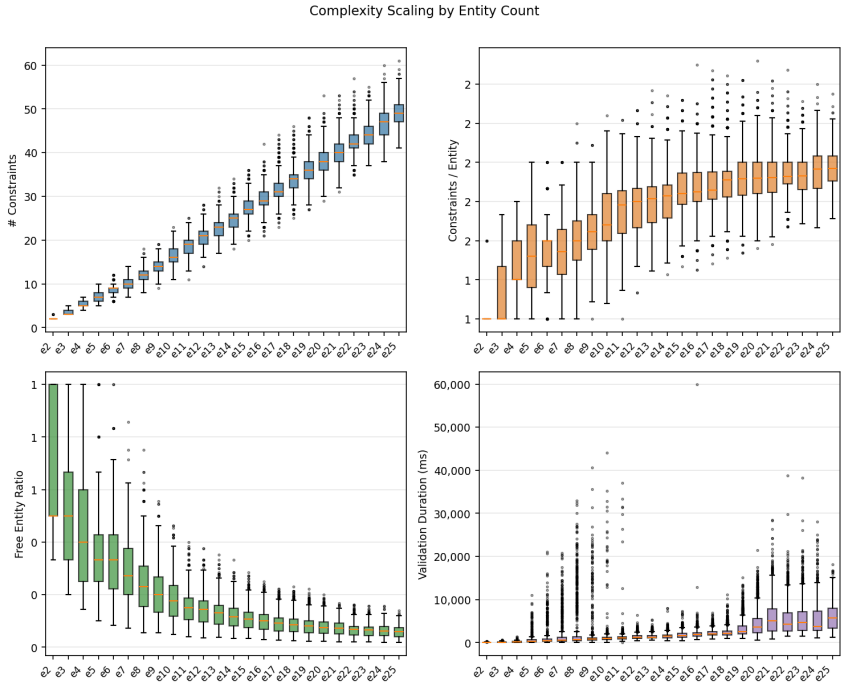

Figure 8. Complexity scaling by entity count. Larger generated configurations contain more constraints, higher constraint density, and fewer free entities, and have longer validation times, indicating that entity growth is accompanied by increasing relational and computational complexity.

4.5.4. Reasoning Trace Characteristics

Figure 9 analyzes the reasoning data extracted after FormalGeo-V2 replay. Unlike the structural statistics above, which describe the explicit constraint graphs, these plots use the proof supervision attached to candidate goals. The extracted goal–action pairs contain both short derivations and longer dependency chains, so a single valid configuration can contribute problems at different deductive depths.

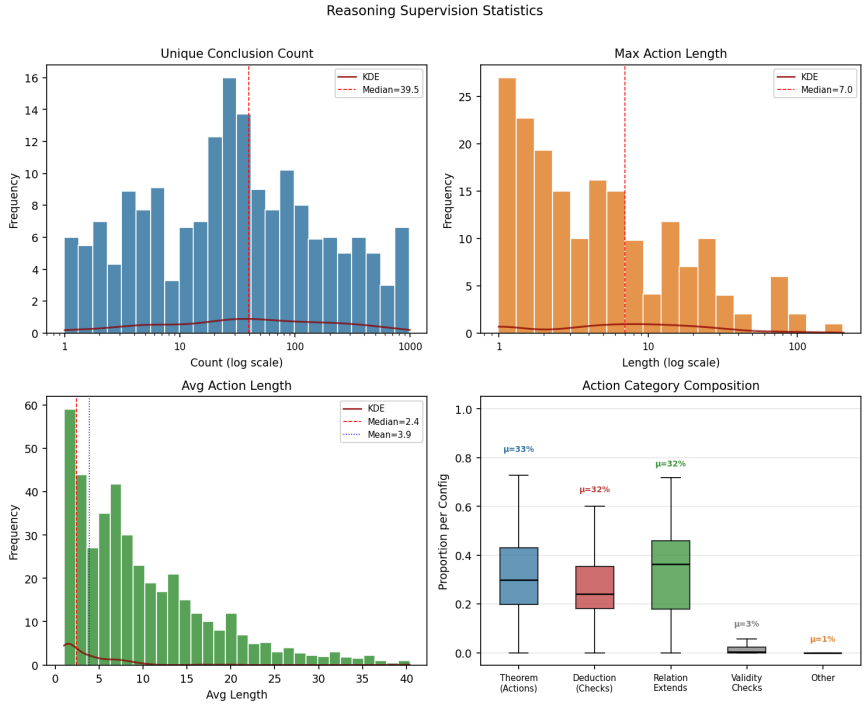

Figure 9. Reasoning data characteristics. The batch-solving stage replays each generated configuration in FormalGeo-V2, extracts proof-hypergraph leaves as candidate goals, and records theorem-level action sequences for proof supervision.

These proof traces provide the supervision format needed for downstream solver training and evaluation. The present analysis uses them to characterize dataset validity and structure, not to claim solver-level performance gains.

4.6. GCG Pipeline Efficiency Evaluation

We evaluate the GCG components introduced in Section 3 with two focused experiments: concurrency scalability and ablation of the validation stages.

4.6.1. Concurrency and Throughput Scalability

As discussed in Section 3.8, generation can be constrained by blocking I/O. The asynchronous writer decouples computation from disk writes and improves parallel throughput in this setting.

We deployed the generation pipeline on a high-performance compute node featuring a 64-core AMD EPYC processor. Because the absolute number of valid configurations generated per minute depends on disk bandwidth, geometric primitive mix, and validation settings, Figure 10 reports the normalized speedup ratio relative to the single-core baseline. In our logs, raw throughput is computed as the number of formally valid configurations written to disk divided by wall-clock generation time, excluding initialization and final serialization cleanup.

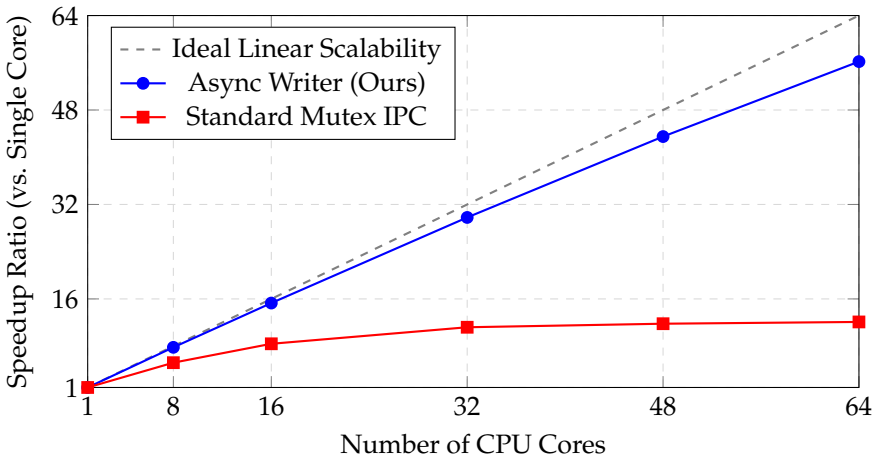

Figure 10. Concurrency scalability and speedup ratio. The plot reports normalized speedup relative to the single-core baseline, not absolute throughput. An asynchronous writer architecture improves scalability with available CPU cores and outperforms a mutex-based pipeline that serializes I/O beyond moderate core counts.

The results suggest that mutex-based IPC plateaus due to synchronized disk flushes, while the asynchronous design scales better by reducing I/O contention.

Scaling begins to flatten beyond roughly 32 cores. Profiling logs attribute this mainly to relation sampling and local equation construction, Jacobian-rank evaluation, FormalGeo-V2 constructive validation, and serialization/I/O through the writer queue. The asynchronous writer reduces serialized disk flushing, but validation and queue contention still limit scaling once enough workers are active.

4.6.2. Ablation Study: Effect of Two-Stage Validation

The two-stage validation pipeline (Section 3) consists of Stage 1 (Jacobian-based algebraic filter) and Stage 2 (progressive geometric validation). We perform an ablation study by disabling these components during generation of highly constrained (DOF $= 0$) instances ($N_{target} = 20$ entities). To account for stochastic variation, the reported failure

rates and generation times are aggregated across five random seeds; confidence intervals are computed across the same seed-level measurements.

We tracked the Configuration Failure Rate (the percentage of stochastic initialization attempts that terminate in unrecoverable mathematical contradictions before reaching the target complexity) and the Mean Generation Time. For each $(\text{pipeline}, N_{target}, T_{max}, \text{seed})$ setting, failure rate is computed as failed attempts divided by total attempts, and the 95% confidence interval is computed across seed-level failure rates as $\bar{x} \pm t_{0.975,n-1} s/\sqrt{n}$. Table 4 reports the main ablation setting and a compact robustness check over different N_{target} and T_{max} values.

Table 4. Ablation and robustness analysis of the two-stage validation pipeline. Results are aggregated across five random seeds. The first two rows report the main setting, including confidence intervals for failure rate; the remaining rows provide compact sensitivity checks.

Setting	Metric	Full Pipeline	w/o Stage 2	w/o Stage 1
$N_{target} = 20, T_{max} = 30$	Failure Rate (%, mean ± std; 95% CI)	**3.2 ± 0.5**; [**2.6**, **3.8**]	34.9 ± 2.8; [31.4, 38.4]	55.2 ± 4.6; [49.5, 60.9]
$N_{target} = 20, T_{max} = 30$	Expected Attempts per Success	**1.03**	1.54	2.23
$N_{target} = 20, T_{max} = 30$	Mean Time (s, mean ± std)	**58.2 ± 2.7**	43.8 ± 3.1	52.6 ± 4.4
$N_{target} = 10, T_{max} = 30$	Failure Rate (%, mean ± std)	**1.1 ± 0.4**	17.4 ± 2.0	27.6 ± 2.8
$N_{target} = 15, T_{max} = 30$	Failure Rate (%, mean ± std)	**2.3 ± 0.7**	27.8 ± 2.4	42.7 ± 3.7
$N_{target} = 20, T_{max} = 10$	Failure Rate (%, mean ± std)	**9.6 ± 1.2**	45.8 ± 3.4	65.4 ± 5.2
$N_{target} = 20, T_{max} = 50$	Failure Rate (%, mean ± std)	**1.7 ± 0.4**	29.8 ± 2.5	48.6 ± 4.0

Table 4 separates the failure modes of the two validation stages. Without Stage 1, redundant constraints more often lead to numerical singularities. Without Stage 2, infeasible partial constructions are rejected later in the search. The expected attempts row is computed from the measured failure rate as $1/(1 - \text{failure rate})$; under the main setting, the full pipeline gives 1.03 expected attempts per success, compared with 1.54 without Stage 2 and 2.23 without Stage 1.

The ablated variants can have lower mean runtimes on completed attempts because they perform fewer validation checks, but their higher failure rates reduce effective generation reliability. The naive random-walk baseline lacks both validation stages; in this dense constrained setting, its failure rate exceeds 99.9%, corresponding to more than 1000 expected attempts per success under the same definition. The sensitivity rows show the same ordering across the tested N_{target} and T_{max} settings.

4.7. Summary

Overall, the analyses show how the generated dataset and the validation pipeline behave under the reported settings. FGeo-GCG contains balanced entity and relation composition, a long-tailed predicate distribution, increasing constraint density with entity count, and proof traces with varied deductive lengths. The concurrency experiment shows that the asynchronous writer improves generation throughput under multi-core execution, while the ablation study shows that both algebraic filtering and progressive geometric validation reduce failures on highly constrained instances. The resulting dataset provides proof-verified premise–target pairs for future downstream solver training and evaluation, although a controlled comparison against prior training datasets is outside the scope of this study.

5. Conclusions

This paper presented FGeo-GCG, a geometric data synthesis framework for generating, validating, and organizing plane geometry configurations. Its central component is the Geometric Configuration Generation (GCG) algorithm, which formulates configuration generation as an incremental linear construction process. A stochastic Jacobian-rank filter screens locally redundant or singular relations, and progressive constructive validation

rejects unrealizable partial configurations before they become complete problem instances. This design reduces unnecessary symbolic validation while preserving the realizability and non-degeneracy required for downstream formal reasoning.

Built on the FormalGeo-V2 deductive engine, FGeo-GCG converts validated constraint graphs into proof-oriented and calculation-oriented targets through forward deduction, goal decomposition, and complexity filtering. The resulting dataset contains more than 50,000 formally validated plane geometry configurations, together with engine-derived reasoning traces. In the reported experiments, the full validation pipeline reduces failures in highly constrained generation, the asynchronous writer improves multi-core throughput, and the generated configurations show long-tailed predicate coverage, increasing constraint density, and balanced entity and relation composition. The proof traces and aligned formal representations provide the material needed for downstream solver-training experiments, while the effect on solver performance remains to be tested under a controlled evaluation protocol.

Limitations

FGeo-GCG currently targets Euclidean plane geometry under the object and relation vocabulary supported by FormalGeo-V2. In practice, this means that the generated instances are centered on points, lines, circles, and the corresponding GDL predicates and theorems. Geometries that require different primitives or semantics, such as spatial incidence relations, non-Euclidean models, algebraic curves, or dynamic constructions, would need separate parameterizations and validation procedures rather than a direct reuse of the present pipeline.

The Jacobian-rank test should also be interpreted as a numerical filter, not as a symbolic proof of constraint independence. It is useful for removing many locally redundant or singular candidates before expensive validation, but it can still be sensitive near degenerate configurations or ill-conditioned constructions. The subsequent constructive validation step mitigates this issue, yet some valid candidates may be rejected conservatively and some near-redundancies may only be detected later in the pipeline.

The reasoning traces are further limited by the current GDL predicate set and theorem library. They do not cover auxiliary constructions or diagram interpretations that fall outside the FormalGeo-V2 engine. Therefore, this paper does not claim a solver-performance gain from training on FGeo-GCG. Such a claim would require a separate study with a shared formal interface, controlled data budgets, and comparable solver settings.

These findings distinguish FGeo-GCG from template expansion: the generator tests candidate relations during construction and records proof-replay information after validation. Future work will first use this dataset in controlled solver-training experiments, and then study how the generation process can incorporate auxiliary constructions, intermediate lemmas, and geometric domains beyond the present Euclidean plane setting.

Author Contributions: Conceptualization, C.Q., X.Z. and T.L.; methodology, C.Q., X.Z., Y.Y. and Y.L.; software, C.Q., X.Z., Y.Y. and Z.S.; validation, C.Q., X.Z. and T.L.; writing—original draft preparation, C.Q.; writing—review and editing, C.Q., X.Z., Z.H. and T.L.; visualization, C.Q.; supervision, T.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by National Natural Science Foundation of China (NSFC) grant 12071282.

Data Availability Statement: The project is available at https://github.com/Vench115/FGeo-GCG, accessed on 20 May 2026.

Acknowledgments: The authors would like to thank all contributors who participated in this research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wu, W.T. Basic Principles of Mechanical Theorem Proving in Elementary Geometries. *J. Autom. Reason.* **1986**, *2*, 221–252. [CrossRef]
2. Buchberger, B. Applications of Gröbner Bases in Non-Linear Computational Geometry. In *Mathematical Aspects of Scientific Software*; Weinberger, H., Miller, W., Rice, J.R., Eds.; Springer: New York, NY, USA, 1988; Volume 14, pp. 59–87. [CrossRef]
3. Zhao, W.X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. A Survey of Large Language Models. *arXiv* **2025**. [CrossRef]
4. Chervonyi, Y.; Trinh, T.H.; Olšák, M.; Yang, X.; Nguyen, H.; Menegali, M.; Jung, J.; Verma, V.; Le, Q.V.; Luong, T. Gold-Medalist Performance in Solving Olympiad Geometry with AlphaGeometry2. *arXiv* **2025**. [CrossRef]
5. Zhu, M.; Wang, Z.; Ji, S.; Du, Z.; Ke, J.; Deng, X.; Yin, Z.; Huang, X.; Wang, H.; Chen, W. GenesisGeo: Technical Report. *arXiv* **2025**. [CrossRef]
6. Zhao, H.; Shen, J.; Zhang, Y.; Gao, S.; Liu, K.; Ma, T.; Zheng, F.; Lin, D.; Zhang, W.; Chen, K. Achieving Olympia-Level Geometry Large Language Model Agent via Complexity Boosting Reinforcement Learning. *arXiv* **2026**. [CrossRef]
7. Zhang, C.; Song, J.; Li, S.; Liang, Y.; Ma, Y.; Wang, W.; Zhu, Y.; Zhu, S.C. Proposing and Solving Olympiad Geometry with Guided Tree Search. *Nat. Mach. Intell.* **2026**, *8*, 84–95. [CrossRef]
8. Chen, L.; Gu, J.; Huang, L.; Huang, W.; Jiang, Z.; Jie, A.; Jin, X.; Jin, X.; Li, C.; Ma, K.; et al. Seed-Prover: Deep and Broad Reasoning for Automated Theorem Proving. *arXiv* **2025**. [CrossRef]
9. Chen, J.; Chen, W.; Du, J.; Hu, J.; Jiang, Z.; Jie, A.; Jin, X.; Jin, X.; Li, C.; Shi, W.; et al. Seed-Prover 1.5: Mastering Undergraduate-Level Theorem Proving via Learning from Experience. *arXiv* **2025**. [CrossRef]
10. Zhou, Y.; Zhao, J.; Zhang, Y.; Wang, B.; Wang, S.; Chen, L.; Wang, J.; Chen, H.; Jie, A.; Zhang, X.; et al. Solving Formal Math Problems by Decomposition and Iterative Reflection. *arXiv* **2025**. [CrossRef]
11. Yin, S.; Fu, C.; Zhao, S.; Li, K.; Sun, X.; Xu, T.; Chen, E. A Survey on Multimodal Large Language Models. *arXiv* **2024**. [CrossRef]
12. Chou, S.C.; Gao, X.S.; Zhang, J.Z. A Deductive Database Approach to Automated Geometry Theorem Proving and Discovering. *J. Autom. Reason.* **2000**, *25*, 219–246. [CrossRef]
13. Nevins, A.J. Plane Geometry Theorem Proving Using Forward Chaining. *Artif. Intell.* **1975**, *6*, 1–23. [CrossRef]
14. Chou, S.C.; Gao, X.S.; Zhang, J.Z. Automated Production of Traditional Proofs for Constructive Geometry Theorems. In Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science, Montreal, QC, Canada, 19–23 June 1993; pp. 48–56. [CrossRef]
15. Chou, S.C. *Mechanical Geometry Theorem Proving*; Number 41 in Mathematics and Its Applications <Dordrecht>; Reidel: Dordrecht, The Netherlands, 1988.
16. Trinh, T.H.; Wu, Y.; Le, Q.V.; He, H.; Luong, T. Solving Olympiad Geometry without Human Demonstrations. *Nature* **2024**, *625*, 476–482. [CrossRef]
17. Pan, Y.; Zhang, Z.; Hu, P.; Ma, J.; Du, J.; Zhang, J.; Liu, Q.; Gao, J.; Ma, F. Enhancing the Geometric Problem-Solving Ability of Multimodal LLMs via Symbolic-Neural Integration. *arXiv* **2025**. [CrossRef]
18. Zhang, X.; Zhu, N.; He, Y.; Zou, J.; Huang, Q.; Jin, X.; Guo, Y.; Mao, C.; Li, Y.; Zhu, Z.; et al. FormalGeo: An Extensible Formalized Framework for Olympiad Geometric Problem Solving. *arXiv* **2024**. [CrossRef]
19. Cai, S.; Bao, K.; Guo, H.; Zhang, J.; Song, J.; Zheng, B. GeoGPT4V: Towards Geometric Multi-modal Large Language Models with Geometric Image Generation. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Miami, FL, USA, 12–16 November 2024; pp. 750–766. [CrossRef]
20. Jiang, Z.; Zhang, T.; Peng, P.; Chen, J.; Xun, Y.; Zhang, H.; Li, L.; Li, Y.; Zhang, S. Towards Generating Controllable and Solvable Geometry Problem by Leveraging Symbolic Deduction Engine. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*; Association for Computational Linguistics: Vienna, Austria, 2025; pp. 1378–1398. [CrossRef]
21. Zhang, Y.; Hu, D.; Yu, T.; Liu, H.; Liu, Y. GeoFM: Enhancing Geometric Reasoning of MLLMs via Synthetic Data Generation through Formal Language. *arXiv* **2025**. [CrossRef]
22. Streinu, I.; Theran, L. Combinatorial Genericity and Minimal Rigidity. In Proceedings of the Twenty-Fourth Annual Symposium on Computational Geometry, College Park, MD, USA, 9–11 June 2008; pp. 365–374. [CrossRef]
23. Capco, J.; Gallet, M.; Grasegger, G.; Koutschan, C.; Lubbes, N.; Schicho, J. The Number of Realizations of a Laman Graph. *SIAM J. Appl. Algebra Geom.* **2018**, *2*, 94–125. [CrossRef]
24. McKay, B.D.; Piperno, A. Practical Graph Isomorphism, II. *J. Symb. Comput.* **2014**, *60*, 94–112. [CrossRef]
25. Liu, J.; Yang, S.; Liu, W.; Ni, F.; Zhu, C. Practical Canonical Labeling of Multi-Digraphs via Computer Algebra. *Symmetry* **2024**, *16*, 1638. [CrossRef]

26. Feng, Y.; You, H.; Zhang, Z.; Ji, R.; Gao, Y. Hypergraph Neural Networks. *Proc. Aaai Conf. Artif. Intell.* **2019**, *33*, 3558–3565. [CrossRef]
27. Cai, D.; Song, M.; Sun, C.; Zhang, B.; Hong, S.; Li, H. Hypergraph Structure Learning for Hypergraph Neural Networks. In Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, Vienna, Austria, 23–29 July 2022; pp. 1923–1929. [CrossRef]
28. Li, M.; Zhang, Y.; Li, X.; Zhang, Y.; Yin, B. Hypergraph Transformer Neural Networks. *ACM Trans. Knowl. Discov. Data* **2023**, *17*, 1–22. [CrossRef]
29. Lu, P.; Gong, R.; Jiang, S.; Qiu, L.; Huang, S.; Liang, X.; Zhu, S.C. 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)*; Association for Computational Linguistics: Vienna, Austria, 2021. [CrossRef]
30. Chen, J.; Tang, J.; Qin, J.; Liang, X.; Liu, L.; Xing, E.; Lin, L. GeoQA: A Geometric Question Answering Benchmark Towards Multimodal Numerical Reasoning. In Proceedings of the Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, Online, 1–6 August 2021; pp. 513–523. [CrossRef]
31. Hao, Y.; Zhang, M.; Yin, F.; Huang, L. PGDP5K: A Diagram Parsing Dataset for Plane Geometry Problems. In Proceedings of the 2022 26th International Conference on Pattern Recognition (ICPR), Montréal, QC, Canada, 21–25 August 2022; pp. 1763–1769.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.