GeoBuildBench: A Benchmark for Interactive and Executable Geometry Construction from Natural Language

Anonymous ACL submission

Abstract

We introduce **GeoBuildBench**, a benchmark designed to evaluate whether large language models and multimodal agents can ground informal natural-language plane geometry problems into executable geometric constructions. Unlike existing geometry benchmarks that focus on answer correctness or static diagram interpretation, GeoBuildBench treats geometry diagram as an interactive construction task: given a textual problem, an agent must generate a domain-specific language (DSL) program to produce a diagram satisfying explicitly specified geometric objects and verifiable constraints. The benchmark features 489 Chinese textbook-style problems, curated through automated filtering and human validation to ensure text-complete, constructible problem specifications. We evaluate several state-of-the-art multimodal models in a bounded iterative setting and show that, despite reasonable success rates, models frequently exhibit structural hallucinations, missing objects, and failures to satisfy geometric constraints, with limited ability to exploit visual and constraint-based feedback for self-correction. These results highlight geometry construction as a rigorous testbed for grounded, executable reasoning beyond textual or visual plausibility. Our benchmark and code are released at https://anonymous.4open.science/r/GeoBuildBench-3032.

1 Introduction

Large language models (LLMs) and multimodal LLMs (MLLMs) have shown strong performance on geometry-focused benchmarks such as GeoQA and Geometry3K, which evaluate multimodal reasoning over textbook-style geometry problems and diagrams (Chen et al., 2021; Lu et al., 2021). In parallel, neuro-symbolic systems such as AlphaGeometry achieve near–Olympiad-level performance by combining language models with formal deduction engines (Trinh et al., 2024). Despite this progress, existing approaches largely assume formalized representations or given diagrams, leaving open a fundamental question: can these models transform informal, natural-language descriptions into consistent, executable geometric structures?

Geometry provides a uniquely stringent testbed for grounded reasoning. Textbook problems describe points, lines, and geometric relations in natural language, implicitly requiring the construction of a consistent diagram before symbolic reasoning. Errors in this process are rarely ambiguous: incorrect or missing constructions typically lead to impossible or visibly invalid diagrams, making geometry well suited for studying structural hallucination.

Prior work has often treated diagram construction as a one-shot preprocessing step, using relation-extraction to map text into solver-specific representations (Gan and Yu, 2018) or numerical optimization (Krueger et al., 2021) to produce coordinates. More recent text-to-diagram methods emphasize coordinate accuracy and formal constraints (Hu and Zhong, 2023; Wang et al., 2025; Cheng et al., 2025), but their evaluations rely on visual similarity or parser-dependent checks, and do not assess whether agents can incrementally build, inspect, and repair geometric structures under explicit semantic constraints. Although agentic frameworks like ReAct and Reflexion (Yao et al., 2023; Shinn et al., 2024) have shown promise in interleaving reasoning with actions, they have primarily been tested in textual environments. Geometry differs fundamentally: every action must satisfy strict spatial constraints, and failures, such as intersecting parallel lines or failing a tangency requirement, are immediately observable through the environment.

To address this gap, we introduce **GeoBuildBench**, a benchmark and interactive environment for evaluating agents that translate natural-language plane geometry problems into executable geomet-

ric constructions. We define a compact, execution-oriented geometry Domain Specific Language (DSL), together with a Python interpreter and Matplotlib renderer. Instead of evaluating against a single "gold" diagram, for each problem GeoBuildBench specifies a set of required objects and explicit verifiable geometric constraints. Agents iteratively generate DSL programs, execute them, and use visual and constraint-based feedback to correct errors, forming a closed agent–environment loop. Figure 1 illustrates this agent–environment loop, including the geometry DSL action space, execution and rendering, and constraint-based verification.

This setting enables us to evaluate models beyond textual correctness or visual plausibility. We measure (i) *executability* of constructions, (ii) *object coverage*, (iii) *geometric constraint satisfaction*, e.g., parallelism and tangency, and (iv) *structural hallucinations*, such as infeasible constructions or references to undefined objects.

Our contributions are threefold:

1. **An Interative Environment:** An executable geometry construction environment with a minimal DSL and deterministic rendering for grounded agent interaction.

2. **The GeoBuildBench Dataset:** A benchmark of 489 Chinese textbook-style geometry problems, annotated with constraints to support evaluation without assuming a single gold construction.

3. **Empirical Analysis of MLLMs:** An evaluation of state-of-the-art models revealing that current MLLMs struggle with structural correctness and iterative self-repair despite their strong performance on existing geometry QA benchmarks.

2 Related Work

2.1 Geometry Problem Solving Benchmarks and Formal Representations

Geometry problem solving has long been studied as a multimodal reasoning task over natural-language descriptions, diagrams, and geometric rules. Early systems such as GEOS combine text parsing and diagram interpretation to answer standardized geometry questions (Seo et al., 2015). More recent benchmarks significantly scale up supervision and emphasize structured intermediate representations. GeoQA introduces a large collection of geometry problems paired with executable programs, enabling interpretable multimodal reasoning (Chen et al., 2021). Geometry3K further provides densely annotated geometry problems and supports symbolic reasoning through the Inter-GPS framework, which translates textual and diagrammatic inputs into a formal geometry language for theorem-guided inference (Lu et al., 2021). In parallel, prior work has explored formalizing natural-language geometry problems by extracting relations and mapping them into solver-compatible representations (Gan and Yu, 2018). While these datasets and systems advance geometry question answering and symbolic reasoning, they primarily focus on predicting answers or formal programs, and do not directly evaluate whether a model can construct a geometrically consistent diagram from natural language under strict executability and constraint satisfaction.

2.2 Text-to-Diagram Generation and Constraint-Based Geometry Construction

A complementary line of work addresses diagram construction from textual descriptions. Geometry Model Builder (GMB) and its domain-specific language GMBL represent constructions and constraints explicitly, generating diagrams via numerical optimization over geometric constraints (Krueger et al., 2021). Hu and Zhong propose a training-free text-to-diagram method that optimizes point coordinates to produce accurate textbook-style diagrams (Hu and Zhong, 2023). More recently, MagicGeo and MagicGeoBench combine LLM-based text formalization with a formal geometry solver to generate constraint-consistent diagrams (Wang et al., 2025). However, their evaluation relies primarily on CLIP-based visual similarity, which measures visual-textual alignment but does not verify whether geometric relations such as parallelism, perpendicularity, or incidence are semantically satisfied. GeoUni explores a unified neural model for diagram generation and geometric reasoning (Cheng et al., 2025), but its evaluation pipeline depends on a geometry parser with limited relational coverage, making it difficult to detect violations of unrecognized constraints. In contrast to these approaches, our work focuses on evaluating geometry construction as an executable, constraint-checked process, and on analyzing whether general-purpose LLM-based agents can iteratively build, inspect, and repair geometric structures from natural

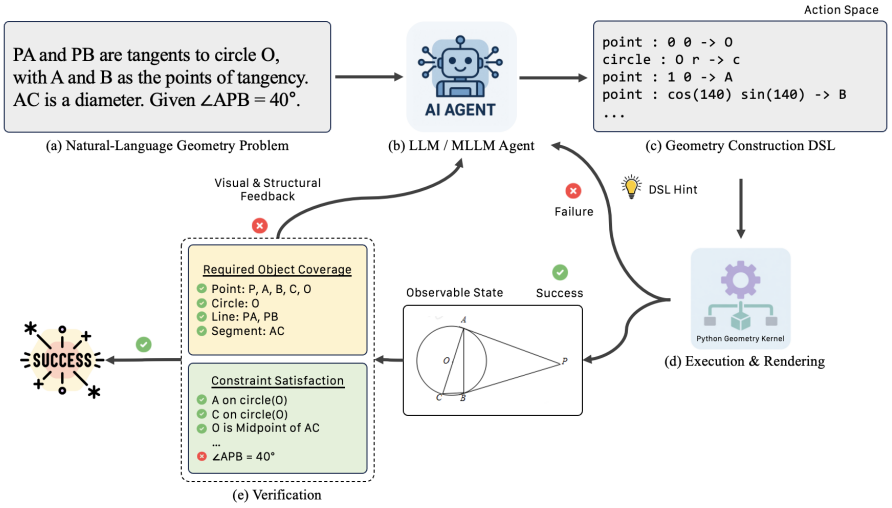

Figure 1: Overview of the GeoBuildBench environment. An agent translates a natural-language geometry problem into executable DSL programs, executes them to obtain rendered diagrams, and verifies the result against required object coverage, construction validity, and geometric constraints. Violation feedback is returned to the agent for iterative repair, while the interaction terminates once all required objects are present and all constraints are satisfied.

language.

3 GeoBuildBench Environment

3.1 Task Definition

GeoBuildBench evaluates an agent's ability to ground a natural-language plane geometry problem into an executable geometric construction. Each task consists of three components: (i) a natural-language problem description, (ii) a set of *required geometric objects* that must appear in the final construction, and (iii) a list of *verification conditions* encoding the geometric semantics of the problem.

The agent's goal is to generate a construction program that, when executed, produces a geometric configuration satisfying all required objects and verification conditions. Importantly, GeoBuildBench does not assume a single gold construction or canonical diagram. Multiple constructions are considered valid as long as they are executable and semantically accurate. This design explicitly encourages the use of *auxiliary constructions* such as additional helper points, lines, or circles introduced to satisfy distance or incidence constraints, mirroring standard human geometric practice.

A task is considered successful if the generated construction (1) executes without runtime errors, (2) contains all required geometric objects in the final state, and (3) satisfies all verification conditions. These criteria define a binary notion of task success, while partial metrics can be computed to analyze intermediate failure modes. Tasks terminate either upon success or when a predefined interaction budget is exceeded.

We do not assume that the verification conditions exhaustively cover all possible geometric relations that may appear in natural-language problems. Instead, each task is specified using a finite and predefined set of condition types that capture common, executable geometric semantics and admit reliable numerical verification.

3.2 Geometry Construction DSL

To support precise and deterministic evaluation, GeoBuildBench introduces a compact, execution-oriented domain-specific language (DSL) for plane geometry construction. The DSL defines the agent's action space and is designed to be minimal, explicit, and fully executable.

The DSL provides primitives for constructing geometric entities (e.g., `point`, `line`, `segment`, `circle`) and operators for common geometric constructions (e.g., intersections, parallel and perpendicular lines, midpoints, rotations). Each command deterministically updates the geometric state and returns named objects that may be referenced by subsequent commands. Crucially, the DSL is a *construction language* rather than a declarative constraint language: relations such as parallelism, perpendicularity, or angle values cannot be asserted directly, but must be realized through explicit constructions.

The complete DSL syntax, supported commands, and execution semantics are specified in Appendix A. In this section, we focus on the DSL's role as an executable action interface rather than enumerating its full grammar.

3.3 Execution and Verification

The generated DSL programs are executed by a geometry kernel that maintains an explicit geometric state, including object coordinates and structural relations. Execution failures, such as referencing undefined objects or attempting geometrically infeasible operations (e.g., intersecting parallel lines), are explicitly detected and reported.

After execution, a verification module assesses the result across three dimensions:

- **Executability** checks whether the program runs to completion without errors.
- **Required Object Coverage** verifies that all geometric objects specified in the task definition are present in the final construction, regardless of any auxiliary objects introduced.
- **Constraint Satisfaction** checks if all verifiable conditions like incidence, parallelism, metric constraints, and angular requirements are met.

Violations of these checks reveal different forms of *structural hallucination*, such as references to non-existent objects, geometrically infeasible constructions, or failures to satisfy required relations. Formal definitions of all verification condition types and their numerical tolerances are provided in Appendix B. This explicit verification enables semantic evaluation beyond textual correctness or visual similarity, without reliance on proxy image-based metrics.

3.4 Agent–Environment Interaction Protocol

As shown in Figure 1, GeoBuildBench is structured as an interactive agent–environment loop. At each step, the agent observes the problem description and previous environment feedback to emit or modify a DSL program without assuming a fixed granularity of updates. The environment executes the program, renders a diagram, and returns visual output together with structured feedback regarding execution errors, missing required objects or violated constraints.

This interaction continues iteratively, allowing the agent to inspect and repair its construction based on feedback. The interaction terminates upon success or when the interaction budget is reached.

4 GeoBuildBench Dataset

The benchmark consists of 489 Chinese plane-geometry problems curated from GeoQA (Chen et al., 2021) and additional online textbook repositories. Each instance is represented by its natural-language text paired with a canonicalized set of required objects and a list of verification conditions that encode the semnatic constraints. We do not claim to cover every possible relation type that may appear in plane geometry. Instead, the verification conditions are defined over a practical closed set of relation types that frequently occur in GeoQA-style problems and admit reliable numerical checking (see Appendix B.2 for the type list and semantics).

4.1 Curation Pipeline

Building a construction-centric benchmark poses challenges beyond those in standard geometry QA datasets. Many problems rely on information that is only recoverable from an accompanying diagram, contain ambiguous references, or specify internally inconsistent constraints. To ensure that each instance is solvable *from text alone as a geometric construction task*, we apply a three-stage curation pipeline combining LLM-based processing with human verification.

Stage 1: Text-based constructibility filtering and cleaning. We first use GPT-4.1 to assess whether a problem is suitable for geometric construction from text alone. The model is prompted to (i) reject problems whose geometric meaning depends on diagram-only cues (e.g., numbered angles such as "$\angle 1$"), undefined point references, or incomplete constraints, and (ii) remove non-constructive content, including answer queries, multiple-choice options, score annotations, and boilerplate phrases such as "as shown in the figure". The output of this stage is a *cleaned text* that retains only geometric setup conditions, e.g., incidences, parallelism, angle relations, and metric constraints.

Notably, a problem may appear well-specified in natural language yet still be inconstructible due to hidden geometric inconsistencies. Such cases highlight that geometric constructibility is nontrivial and cannot be determined from surface form alone. We provide concrete examples of both constructible and inconstructible cases in Appendix F.

Stage 2: Task representation extraction. For problems that pass Stage 1, we apply a second GPT-4.1 prompt to extract a structured task representation. This stage converts the cleaned text into (i) `required_objects`, e.g., points, segments/lines, circles, and polygons, (ii) `verification_conditions`, e.g., point-on-segment, angle values, parallelism, length relations, and (iii) auxiliary metadata such as category labels and an estimated construction difficulty. The prompt enforces a fixed schema and a closed set of supported condition types, ensuring consistency across instances.

Importantly, this stage is *not* intended to solve the geometry problem: it only maps surface geometric descriptions into pre-defined condition templates. As a result, the extraction can succeed even when the model's geometric reasoning is weak, because it primarily requires schema-constrained mapping rather than multi-step deduction. Moreover, although the same geometric relation may be expressed in diverse natural-language forms, the output is explicitly canonicalized into our finite condition-type set, which later enables unambiguous constraint-based evaluation and hallucination diagnosis.

Stage 3: Human verification. Finally, all retained instances undergo human verification. Annotators check that (i) the extracted task representation reflects the cleaned text, (ii) required objects and conditions are neither missing nor malformed, and (iii) the constraint set is geometrically realizable, i.e., a valid construction exists in principle. Crucially, annotators are *not* asked to re-interpret the full problem or invent new constraints; their role is primarily to validate that the LLM output conforms to the fixed schema and intended semantics, and to filter out rare cases of inconsistency (e.g., mutually incompatible length and perimeter constraints). Problems that are syntactically well-formed but geometrically inconsistent are removed at this stage.

Annotators and consistency. Human verification was conducted by two graduate-level annotators with formal training in mathematics. Both annotators followed a unified annotation guideline specifying the allowed object types, verification condition schema, and rejection criteria, and were instructed to assess geometric realizability strictly under the given constraints, without introducing unstated assumptions.

To ensure consistency, all instances were reviewed by at least one annotator, and a subset of cases were independently checked by a second annotator. Disagreements were rare and were resolved through discussion. The verification process focuses on checking schema conformity and geometric realizability, and does not require solving the problems.

4.2 Task Representation

Each verified problem is represented as a construction task defined by `required_objects` and `verification_conditions`. Importantly, GeoBuildBench does not assume a single gold diagram or construction procedure. Multiple constructions are considered valid as long as they introduce all required objects and satisfy all verification conditions. This design explicitly allows auxiliary constructions (e.g., helper points, lines, or circles), reflecting standard human geometric practice. An example task instance is shown in Appendix F.

4.3 Dataset Statistics

GeoBuildBench covers a broad range of plane-geometry topics commonly found in textbook-style problems, with a majority of instances involving circle- and triangle-based constructions. The benchmark also includes problems featuring angle relationships, geometric transformations, metric relations, polygons, and similarity or congruence.

Each problem is annotated with a construction difficulty level ranging from 1 (very easy) to 4 (hard), reflecting the complexity of *constructing* the geometric configuration rather than solving a downstream numerical or proof-based question. Most instances fall into moderate difficulty levels, making the benchmark well suited for evaluating multi-step construction and error recovery.

Detailed category and difficulty distributions are provided in Appendix E.

5 Benchmark Evaluation and Analysis

5.1 Experimental Setup

We evaluate GeoBuildBench using a set of modern multimodal large language models capable of reasoning and geometry construction with visual input support. For each model, we adopt a fixed interaction budget of **maximum 5 iterations**, where at each step the agent can emit a DSL program to be executed by our geometry kernel and receive structured feedback.

All experiments use the same execution environment, ensuring fairness across models. Each problem instance is processed in *vision-enabled mode*, whereby the agent observes both the textual problem description and a rendering of the current geometric state.

We test the following models:

- **GPT-5.1** (OpenAI): state-of-the-art reasoning and multimodal LLM (OpenAI, 2025).
- **Gemini-3-Flash** (Google) — low-latency reasoning model (Google Cloud, 2025).
- **Qwen3-VL-235B-A22B-Instruct** (Alibaba): advanced multimodal vision-language model (Bai et al., 2025).
- **LLaMA-3.2-90B-Vision-Instruct** (Meta): open-weight vision reasoning model (Meta AI, 2024).

During evaluation, each model is prompted to construct the geometry in an iterative loop. We do not finetune models; all evaluations use zero-shot or few-shot prompting strategies consistent across models.

5.2 Evaluation Metrics

To assess model performance on GeoBuildBench, we define the following metrics:

Success Rate The fraction of problems where the agent successfully constructs a diagram satisfying all required objects and verification conditions within the fixed interaction budget.

Iteration Steps We report the distribution of steps used to reach successful solutions, including the minimum, average, and maximum number of iterations.

Hallucination Analysis We count hallucination events: cases where the model references nonexistent objects, emits syntactically invalid DSL, or returns mismatched program output. We also analyze the average number of hallucinations per problem and recovery rates.

Missing Objects We track the total number of geometric objects that were required by a task but not produced by the model's construction program, broken down by type (e.g., points, segments, circles).

Failed Conditions We measure the total number of verification conditions that remain unsatisfied after program execution. We classify failures by condition type (e.g., `angle_value`, `parallel`, etc.).

These metrics together provide a comprehensive view of a model's ability to ground textual geometry problems into executable constructions.

5.3 Overall Results

Table 1 reports the overall performance of four state-of-the-art vision-language models on GeoBuildBench. A construction is considered successful only if it executes without error, covers all required geometric objects, and satisfies all verification conditions.

We observe substantial variation across models. GPT-5.1 and Gemini-3-Flash achieve the highest success rates (78.9% and 75.3%, respectively), indicating that current frontier MLLMs can often construct valid geometric configurations under explicit semantic constraints. However, these two models exhibit different error profiles. Gemini-3-Flash requires fewer steps on average (1.55 vs. 1.87) and produces the lowest number of hallucinations per problem (0.34), suggesting more stable geometric construction behavior, while GPT-5.1 achieves a slightly higher overall success rate.

In contrast, Qwen3-VL-235B and LLaMA-3.2-90B-Vision perform significantly worse, with success rates of 42.2% and 21.3%, respectively. Both models exhibit substantially higher hallucination rates (above 2.3 per problem on average), along with large numbers of missing required objects and failed geometric constraints. These failures indicate difficulty in maintaining consistent geometric state and respecting relational constraints over multi-step construction.

Across all models, we find that unsuccessful cases are dominated by structural errors rather than execution failures. In particular, missing geometric objects and violations of angle, incidence, and metric constraints account for the majority of failures, highlighting that GeoBuildBench primarily probes semantic grounding and structural reasoning, rather than surface-level syntax or program execution.

5.4 Structural Hallucinations

We first analyze *structural hallucinations*, which capture failures in maintaining a coherent geometric state during construction. We define a structural hallucination as an error that introduces un-

Table 1: Overall performance on GeoBuildBench. All models are evaluated with a maximum of 5 interaction steps. Lower is better (↓) for steps, hallucinations, missing objects, and failed constraints.

Model	Success Rate (%)	Avg. Steps ↓	Hallucinations / Prob. ↓	Missing Objects ↓	Failed Constraints ↓
GPT-5.1	**78.9**	1.87	0.40	939	1119
Gemini-3-Flash	75.3	**1.55**	**0.34**	**329**	**932**
Qwen3-VL-235B	42.2	2.30	2.30	2042	1817
LLaMA-3.2-90B-Vision	21.3	2.23	2.38	1823	1584

Table 2: Structural hallucination frequency and recovery behavior on GeoBuildBench. Lower is better (↓) for all metrics.

Model	Hallucinations per Problem ↓	Recovery Steps per Hallucination ↓	Avg. Success Steps ↓
GPT-5.1	0.40	1.29	1.87
Gemini-3-Flash	**0.34**	**1.17**	**1.55**
Qwen3-VL-235B	2.30	1.74	2.30
LLaMA-3.2-90B-Vision	2.38	1.79	2.23

defined geometric references, syntactically invalid DSL programs, or mismatches between declared and produced objects.

Table 2 shows that hallucination frequency varies substantially across models. Gemini-3-Flash and GPT-5.1 exhibit low hallucination rates (0.34 and 0.40 per problem, respectively), while Qwen3-VL-235B and LLaMA-3.2-90B-Vision produce more than two hallucinations per problem on average.

Across all models, the most frequent hallucination type is *undefined reference*, where a DSL command refers to a geometric object that has not been instantiated. This trend holds consistently across models (see Appendix G.2), with weaker models exhibiting an order-of-magnitude increase in such errors. This indicates that most failures arise from difficulties in tracking and reusing previously constructed geometric objects, rather than from superficial syntax issues. The prevalence of such errors highlights the challenge of maintaining a consistent internal object graph over multi-step geometric construction.

A detailed quantitative breakdown of hallucination types, missing objects, and failed verification conditions is provided in Appendix G.4.

5.5 Feedback Sensitivity and Error Recovery

GeoBuildBench explicitly exposes agents to visual renderings and structured constraint feedback after each construction attempt. This enables analysis not only of how often models make structural errors, but also of how effectively they recover from them.

We quantify recovery behavior using the average number of steps required to resolve a hallucination once it occurs. As shown in Table 2, frontier models recover quickly from structural errors. Gemini-3-Flash achieves the fastest recovery (1.17 steps per hallucination), followed by GPT-5.1 (1.29 steps), indicating that these models often correct errors immediately after receiving feedback.

In contrast, Qwen3-VL-235B and LLaMA-3.2-90B-Vision exhibit substantially weaker recovery behavior, requiring 1.74 and 1.79 steps per hallucination, respectively. Despite being allowed the same interaction budget, hallucinations in these models frequently persist across iterations rather than being resolved.

Importantly, hallucination frequency and recovery speed capture complementary aspects of agent behavior. Some models exhibit frequent structural errors but are nevertheless able to interpret feedback and repair them efficiently. Others fail both to maintain a stable geometric state and to exploit feedback for correction. These results demonstrate that iterative interaction alone is insufficient: effective geometry construction requires sensitivity to explicit structural feedback and the ability to update internal representations accordingly.

5.6 Auxiliary Constructions and Geometric Flexibility

A distinctive feature of GeoBuildBench is that it permits auxiliary geometric objects and constructions that are not explicitly mentioned in the problem statement. This design mirrors standard human geometric practice, where additional points, circles,

Table 3: Vision ablation on open-weight models.

Model	Setting	Success Rate (%)	Avg. Steps ↓	Missing Objects ↓	Failed Constraints ↓	Halluc. ↓
Qwen3-VL-235B	with vision	42.2	2.30	2042	1817	928
	no vision	39.1	2.08	131	100	85
LLaMA-3.2-90B-Vision	with vision	21.3	2.23	1823	1584	819
	no vision	23.1	2.19	2408	2165	1108

or perpendiculars are often introduced to satisfy metric or angular constraints.

We observe that successful models frequently exploit this flexibility. Both GPT-5.1 and Gemini-3-Flash regularly introduce helper constructions, such as circles to enforce length constraints or perpendicular lines to realize right angles. These auxiliary objects are retained in the construction state but ignored by the verification module as long as all required objects and constraints are satisfied.

In contrast, weaker models rarely make effective use of auxiliary constructions. Instead, they tend to omit required relations or repeatedly attempt to satisfy constraints without constructing the necessary geometric support. This behavior suggests a more superficial treatment of geometric relations and an inability to flexibly leverage geometric properties.

By allowing auxiliary constructions, GeoBuildBench evaluates not whether a model reproduces a canonical diagram, but whether it can *use geometry as a toolkit* to realize constraints. This capability is largely invisible to benchmarks based on visual similarity or single-shot diagram generation.

5.7 Ablation Study

We conduct a controlled vision ablation on the two open-weight models. For each model, we toggle image inputs (*with vision* vs. *no vision*) while keeping the agent–environment loop, interaction budget, and prompting template fixed. Table 3 summarizes the ablation outcomes. Overall, enabling vision produces *small* changes in headline success rates, but *large* shifts in failure-mode profiles (missing objects, failed conditions, and structural hallucinations).

For Qwen3-VL, enabling vision yields a modest success increase (42.2% vs. 39.1%), but substantially increases all verifier-diagnosed error signals: missing required objects (2042 vs. 131), unsatisfied conditions (1817 vs. 100), and structural hallucinations (928 vs. 85). This suggests that vision can help Qwen identify viable construction ideas more often, but it simultaneously destabilizes symbol binding and state tracking (e.g., referencing objects not yet created or not present in the current construction state).

For LLaMA-3.2-Vision, enabling vision does not improve success (21.3% vs. 23.1%), but it consistently reduces structural errors: missing objects (1823 vs. 2408), failed conditions (1584 vs. 2165), and hallucinations (819 vs. 1108). This pattern indicates that vision primarily improves output cleanliness and object referencing correctness, while the dominant remaining bottleneck is geometric planning and constraint satisfaction.

5.8 Discussion

Taken together, our results show that performance on GeoBuildBench is governed primarily by a model's ability to maintain a coherent geometric state, recover from structural errors using explicit feedback, and flexibly employ auxiliary constructions. High success rates correlate with low hallucination frequency and fast recovery, rather than with raw model size or general vision-language capacity.

A key implication is that many failure modes revealed by GeoBuildBench would likely be missed by evaluation protocols that rely only on visual plausibility or answer correctness. A diagram can look reasonable to a human observer while violating essential geometric relations, and an agent may reach the correct answer despite an internally inconsistent or incomplete construction. In contrast, GeoBuildBench makes these errors explicit by verifying object existence and constraint satisfaction.

Overall, these findings suggest that current MLLMs—despite strong performance on conventional geometry tasks—still struggle with robust, executable geometry construction. GeoBuildBench offers a controlled testbed for diagnosing such limitations and for supporting future work on structurally grounded, agentic reasoning for geometry.

Additional breakdowns of hallucination categories, missing-object cases, and violated verification conditions are reported in the Appendix.

Limitations

GeoBuildBench is designed to benchmark grounded, executable geometry construction, but it does not fully cover the space of geometric language understanding or geometric reasoning. The benchmark scope is intentionally constrained. Tasks focus on *Chinese* plane-geometry problems and a practical, closed set of required-object types and verification condition templates. As a result, performance on GeoBuildBench may not transfer directly to (i) other languages (e.g., English problem statements), (ii) broader Euclidean topics that rely on relations outside our condition-type set, or (iii) settings that require richer mathematical context (e.g., analytic geometry, 3D geometry, or non-Euclidean variants). Relatedly, the action space is defined by our Geometry Construction DSL. While it is expressive for common constructions, it remains a DSL abstraction rather than a full-featured dynamic geometry system, and agent behaviors may differ under alternative tooling or interaction primitives. Moreover, the reported model results are contingent on a particular interactive protocol (vision-enabled feedback, fixed iteration budget, and a specific prompting and harness design). Different step budgets, feedback granularity, or prompting strategies may change absolute success rates and error profiles. We therefore recommend interpreting the reported numbers primarily as evidence about failure modes and relative robustness under our controlled setting, rather than as definitive rankings across all geometry-construction regimes.

References

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, and *et al.* 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

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

J. K. Cheng and 1 others. 2025. Geouni: A unified model for generating geometry diagrams and reasoning. *arXiv preprint arXiv:2504.10146*.

Wei Gan and Xiao Yu. 2018. Automatic understanding and formalization of natural language geometry problems using syntax–semantics models. *Artificial Intelligence Review*.

Google Cloud. 2025. Gemini 3 flash model overview. Accessed 2025.

Zhen Hu and Xin Zhong. 2023. A precise text-to-diagram generation method for elementary geometry. In *ICCWAMTIP*.

Robert Krueger, Jun M. Han, and Daniel Selsam. 2021. Automatically building diagrams for olympiad geometry problems. In *Lecture Notes in Computer Science*.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. 2021. Inter-gps: Interpretable geometry problem solving with formal language and symbolic reasoning. *Preprint*, arXiv:2105.04165.

Meta AI. 2024. Llama-3.2-90b-vision-instruct model card. `https://huggingface.co/meta-llama/Llama-3.2-90B-Vision`. Accessed 2025.

OpenAI. 2025. Introducing GPT-5.1 for developers. Accessed 2025.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1466–1476, Lisbon, Portugal. Association for Computational Linguistics.

Noah Shinn and 1 others. 2024. Reflexion: Language agents with verbal reinforcement learning. In *International Conference on Learning Representations*.

Tuan Anh Trinh and 1 others. 2024. Solving olympiad geometry without human demonstrations. *Nature*.

Jincheng Wang and 1 others. 2025. Magicgeo: Training-free text-guided geometric diagram generation. *arXiv preprint*.

Shunyu Yao and 1 others. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

A GeoDSL Quick Reference

Syntax. Each command follows:

```
command : inputs -> outputs
```

Commands are executed sequentially. Outputs are named objects that can be referenced by later commands.

Numeric literals and expressions. GeoDSL supports inline numeric literals (integers and floats) and arithmetic/trigonometric expressions for coordinates, radii, and angles. Supported operators/functions include `+,-,*,/,( )` and `sin`, `cos`, `tan`. Angles can be specified in degrees (e.g., `90°`, `45deg`) or radians (e.g., `1.5708rad`, `1.5708r`); unitless angles default to degrees.

Design note. GeoDSL is a *construction* language: it provides commands to build geometric objects rather than to assert constraints. Auxiliary constructions (additional points, lines, or circles) are allowed.

A.1 Primitive Object Constructors

Table 4 summarizes the primitive commands for creating geometric objects in GeoDSL, including points, lines, segments, rays, circles, and angle objects. All primitives deterministically create concrete geometric entities that can be referenced in subsequent commands.

A.2 Construction Operators

Table 5 lists operators for common geometric constructions, including intersections, parallel and perpendicular lines, midpoints, and rotations.

A.3 Measurement and Arithmetic Utilities

Table 6 lists commands for distance measurement and basic arithmetic operations. Most arithmetic can alternatively be expressed via inline expressions.

B Verification Semantics and Failure Modes

This appendix formally specifies the geometric verification semantics used in GeoBuildBench. It defines the scope of supported verification conditions, the object-level coverage checks applied to each construction, and the interpretation of verification failures as distinct forms of structural hallucination.

B.1 Verification Scope and Object Coverage

Each task instance specifies a set of *required geometric objects* and a set of *verification conditions*. Verification proceeds only after executing the generated GeoDSL program and instantiating a concrete geometric state.

GeoBuildBench does not aim to cover all possible geometric relations that may appear in natural-language problems. Instead, verification is defined over a finite and explicitly enumerated set of condition types that (i) frequently occur in textbook-style plane geometry problems and (ii) admit reliable numerical evaluation. This closed design ensures deterministic checking and consistent interpretation across instances.

Before evaluating relational constraints, the verifier first checks *object coverage*, ensuring that all required geometric entities are explicitly present in the final construction. The object types considered include: (i) points, (ii) segments, (iii) lines, (iv) circles, and (v) polygonal objects when specified by the task.

Missing objects are treated as verification failures regardless of whether auxiliary constructions are permitted. This separation allows the benchmark to distinguish failures caused by omitted entities from those arising from incorrect geometric relations.

B.2 Supported Verification Condition Types

GeoBuildBench evaluates a finite and explicitly defined set of verification condition types. Although natural-language geometry descriptions may express the same relation in diverse surface forms, all conditions are normalized into this fixed set during dataset curation (see Section 4.1). As a result, downstream verification and hallucination diagnosis operate on canonicalized geometric semantics rather than on raw linguistic expressions.

The supported condition types cover a broad range of geometric semantics and can be grouped into the following categories.

Incidence and topological relations. These conditions verify whether geometric objects satisfy required incidence or ordering relations, including point-on-line, point-on-segment, point-on-circle, collinearity, concurrency, concyclicity, order-on-line, and related variants (e.g., extensions or arc-based relations).

Table 4: Primitive object constructors in GeoDSL.

Category	Command	Inputs	Outputs / Description
Point	`point`	`-`, `x y`, `line`, `circle`	Creates a point. Coordinates create a fixed point; line/circle inputs sample a point on the object.
Line	`line`	`A B`	Line through points A and B.
Line	`line_bisector`	`A B`	Perpendicular bisector of segment AB.
Segment	`segment`	`A B`	Segment with endpoints A and B.
Ray	`ray`	`A B`	Ray starting at A and passing through B.
Circle	`circle`	`O A`, `O r`	Circle centered at O, defined by a point or radius.
Angle	`angle`	`A B C`	Angle at vertex B formed by rays BA and BC.
Constant	`const int`, `const Measure`	`v`	Creates numeric or length constants.

Table 5: Construction operators in GeoDSL.

Command	Inputs	Outputs	Description
`intersect`	`obj1 obj2`	`P` or `P Q`	Intersection of geometric objects (e.g., line–line, line–circle).
`parallel_line`	`P ref`	`l`	Line through P parallel to a reference object.
`orthogonal_line`	`P line`	`l`	Line through P perpendicular to a given line.
`midpoint`	`A B`	`M`	Midpoint of segment AB.
`rotate`	`P` θ `Center`	`Q`	Rotates point P by angle θ around `Center`.

Metric and angular constraints. Metric conditions enforce equality, sum, or ratio relations among distances, segments, or angles. These include distance equality, segment equality, angle value, angle equality, angle sum, angle ratio, and segment ratio. Specialized constraints such as isosceles triangle, right triangle, and perimeter constraints are treated as derived metric relations.

Directional and structural relations. Parallelism, perpendicularity, perpendicular bisectors, and diameter-based conditions are verified as structural relations that constrain the global configuration of objects. Violations typically indicate geometrically infeasible constructions rather than local numerical error.

Curvature and tangency relations. Tangency-related conditions, including tangent, tangent at point, and line-is-tangent, verify first-order contact between linear and circular objects under numerical tolerance.

Polygonal and higher-level properties. When polygonal objects are specified, the verifier checks polygon type, regular polygon constraints, square constraints, and associated polygon properties. These conditions are evaluated through their induced metric and angular relations rather than through symbolic assertions.

B.3 Numerical Tolerances and Equivalence Handling

All metric and angular conditions are evaluated under fixed numerical tolerances to account for floating-point imprecision. For angular constraints, both the measured angle and its reflex ($360^\circ - \theta$) are treated as equivalent to avoid spurious failures due to orientation ambiguity. Distance-based conditions are normalized to be invariant to global scaling applied during rendering.

B.4 Interpretation of Verification Failures

Verification failures are interpreted as evidence of distinct structural failure modes in the agent–environment loop. Importantly, because all conditions are expressed in a canonicalized form, failure diagnosis is independent of the original linguistic realization of the constraint.

Non-existent object references. Conditions that reference undefined geometric entities indicate hallucinated symbols that were never constructed in the executable geometry.

Table 6: Measurement and arithmetic utilities in GeoDSL.

Command	Inputs	Outputs	Description
distance	A B	d	Distance between points A and B.
sum	a b	c	Addition.
minus	a b	c	Subtraction.
product	a b	c	Multiplication.
ratio	a b	c	Division.

Missing required constructions. Failures of object coverage or incidence relations reflect omissions of required points, segments, lines, or circles, even when auxiliary constructions are allowed.

Geometrically infeasible relations. Violations of parallelism, perpendicularity, tangency, or angle constraints typically arise from internally inconsistent or infeasible configurations that cannot be satisfied simultaneously.

Semantic mismatch under tolerance. Near-satisfied metric or angular constraints that fall outside numerical tolerance indicate partial semantic understanding rather than syntactic or execution failure.

By explicitly separating object coverage, canonicalized relational verification, and numerical evaluation, GeoBuildBench enables semantics-aware diagnostics that go beyond textual correctness or visual similarity and directly probe structured geometric reasoning.

C Full Agent System Prompt

Listing 1: Full system prompt used for all agents in GeoBuildBench.

```
You are a geometry problem-solving agent that
    uses a Domain-Specific Language (DSL) to
    construct geometric figures.

Your task is to read Chinese geometry problems
    and generate DSL code that creates the
    geometric construction described in the
    problem.

## DSL Syntax Overview

Format: command : inputs -> outputs

### Basic Commands
# Points
point : 100 150 -> P

# Mathematical Expressions (NO SPACES!)
point : 50+30 100-20 -> Q
point : cos(30°) sin(30°) -> R
point : 100*cos(45°) 100*sin(45°) -> S
point : (100+100*cos(115°)) (0+100*sin(115°)) ->
     P

# Lines & Segments
line : A B -> line_AB
segment : A B -> seg_AB
ray : A B -> ray_AB

# Circles
circle : O A -> circle_O
circle : O 50 -> circle_O
circle : O 100*sin(60°) -> c

# Angles & Rotation
angle : A B C -> angle_ABC
rotate : P 60 Center -> P_rot
rotate : P 60° Center -> P_rot

# Constructions
midpoint : A B -> M
orthogonal_line : P line -> perp
parallel_line : P line -> para
intersect : line1 line2 -> P
intersect : line circle_O -> P1 P2
line_bisector : A B -> bisector
angular_bisector : A B C -> bis

# Triangle Centers
incenter : A B C -> I
circumcenter : A B C -> O
incircle : A B C -> circle_I
circumcircle : A B C -> circle_O

## CRITICAL RULES
1. Define before use
2. One definition per label
3. NO polygon command
4. Construct, don't assert
5. Use expressions for precision
```

D Task Parsing and Annotation Prompts

This appendix reports the exact prompts used to filter, clean, and annotate Chinese geometry problems for inclusion in GeoBuildBench. These prompts define the construction suitability criteria and the formal extraction of geometric objects and verification conditions.

D.1 Construction Suitability Filtering Prompt

```
Analyze this geometry problem for GEOMETRIC
    CONSTRUCTION suitability.

Problem: {problem_text}

Your task:
1. Determine whether this problem is suitable
    for geometric figure construction
2. Remove non-construction content (questions,
    scores, diagram references, etc.)
3. Keep ONLY the geometric setup conditions

REJECTION CRITERIA (return is_valid: false if
    ANY apply):

1. Undefined Points:
   - "angle E = 40 degrees" (point E is not
    geometrically defined)
   - "angle BDC = 30 degrees" (point D is not
    defined)
   - Valid: "point D lies on segment AB, angle
    BDC = 30 degrees"

2. Ambiguous Angles:
   - "angle D = 26 degrees"
   - Valid: "angle ABC = 50 degrees"

3. Diagram-Dependent Angle Labels:
   - "angle 1 = 30 degrees, angle 2 = 45 degrees
    "

4. Incomplete Constraints:
   - "AB is parallel to CD, angle E = 40 degrees
    " (angle location undefined)

5. Pure Calculation Problems:
   - Problems that only ask for numerical values
     without defining a constructible figure

CLEANING RULES:
- Remove score indicators
- Remove diagram references (e.g., references to
     figures)
- Remove questions or proof requests
- Remove multiple-choice answers
- Remove any text appearing after result or
    query markers
  (e.g., result clauses, questions, or proof
    statements)

KEEP:
- Shape definitions (e.g., triangles,
    quadrilaterals)
- Explicit point positions
- Measurements (lengths, angles)
- Geometric relations (parallel, perpendicular,
    etc.)

Return JSON:
{
  "is_valid": true/false,
  "cleaned_text": "geometry construction
    conditions only",
  "rejection_reason": ""
}

Only return the JSON.
```

D.2 Formal Construction Annotation Prompt

```
Parse this geometry problem and extract
    construction requirements.

Problem: {problem_text}

TASK 1: Extract geometric objects and conditions
TASK 2: Classify into ONE category
TASK 3: Rate construction difficulty (1-5)

Return JSON:
{
  "required_objects": {
    "points": [],
    "segments": [],
    "lines": [],
    "circles": [],
    "polygons": []
  },
  "verification_conditions": [],
  "category": "...",
  "difficulty": 3
}

SUPPORTED CONDITION TYPES:
- parallel, perpendicular, collinear, concurrent
- angle_value, angle_equality, angle_sum,
    angle_ratio
- segment_equality, segment_ratio, perimeter
- point_on_segment, midpoint_of, order_on_line
- point_on_circle, tangent_line, diameter
- triangle_valid, isosceles_triangle,
    right_triangle
- polygon_property, polygon_type, square,
    regular_polygon

CRITICAL RULES:
- All points must be explicitly defined
- Use only supported condition types
- difficulty reflects construction complexity,
    not solving difficulty

Only return JSON.
```

E Dataset Statistics and Annotation Details

This appendix provides detailed statistics and annotation information for GeoBuildBench, complementing the high-level description in the main text. We report the distribution of problem categories and construction difficulty levels, and clarify the principles used during difficulty annotation.

E.1 Category Distribution

GeoBuildBench spans a diverse range of plane-geometry topics commonly appearing in middle- and high-school textbooks. Each problem is assigned to a single high-level category based on its primary geometric structure and constraints.

Table 7 reports the distribution over categories. Circle- and triangle-based constructions constitute

the majority of the dataset, while a substantial portion involves angle relationships, geometric transformations, and metric constraints. Polygonal constructions and similarity or congruence relations are also included, ensuring coverage beyond simple primitive configurations.

Table 7: Category distribution of GeoBuildBench (489 problems).

Category	Count	Ratio (%)
Circle	154	31.5
Triangle	143	29.2
Angle Relationships	80	16.4
Geometric Transformations	41	8.4
Polygon	32	6.5
Metric Relations	25	5.1
Similarity & Congruence	9	1.8
Basic Constructions	5	1.0

E.2 Construction Difficulty Annotation

Each problem in GeoBuildBench is annotated with a discrete construction difficulty level ranging from 1 (Very Easy) to 4 (Hard). Importantly, this difficulty score reflects the complexity of *constructing* the geometric configuration itself, rather than the difficulty of solving any associated numerical or proof-based question.

Difficulty levels are assigned during dataset curation based on the following factors: (i) the number of required geometric objects, (ii) the degree of dependency between constructions (e.g., intersection-defined points or chained auxiliary constructions), and (iii) the precision and interaction of metric and angular constraints. Problems that require multiple auxiliary constructions or tight constraint coordination are rated as higher difficulty.

Table 8: Construction difficulty distribution in GeoBuildBench.

Difficulty Level	Count	Ratio (%)
Level 1 (Very Easy)	54	11.0
Level 2 (Easy)	238	48.7
Level 3 (Medium)	162	33.1
Level 4 (Hard)	35	7.2

E.3 Interpretation and Use in Evaluation

The distribution shows that GeoBuildBench is dominated by Easy and Medium instances, reflecting the structure of typical educational geometry problems. At the same time, the inclusion of a nontrivial fraction of Hard instances enables stress-testing of multi-step construction, auxiliary reasoning, and error recovery under feedback.

During evaluation, difficulty labels are not exposed to the models. Instead, they are used only for analysis, allowing us to examine how construction complexity correlates with success rate, hallucination frequency, and feedback utilization.

F Illustrative Construction Examples

This appendix presents two representative examples to illustrate the nontrivial nature of geometric construction from natural-language descriptions. The first example demonstrates a problem that is constructible but requires non-obvious auxiliary constructions. The second example highlights a problem that appears well-specified in text yet is geometrically inconsistent and therefore inconstructible. Together, these examples clarify both the capabilities and the limitations addressed by GeoBuildBench.

F.1 Example A: Constructible (Tangent–Diameter Configuration)

Problem (translated). *Let PA and PB be tangents to circle O at points A and B. Let AC be a diameter. Given $\angle APB = 40°$.*

Why this example is nontrivial in our setting. GeoDSL is a *construction* language (construct, don't assert): it cannot directly assert "PA is tangent" or "$\angle APB = 40°$". Instead, the agent must explicitly construct a concrete configuration that *realizes* tangency (via perpendicularity to radii) and matches the target angle, potentially introducing auxiliary objects.

A normalized executable instance (one of many valid constructions). This problem does not uniquely determine a single diagram up to coordinates. Therefore, the agent may instantiate any concrete configuration that satisfies the constraints. A convenient choice uses the tangent–tangent angle identity:

$$\angle APB = 180° - \angle AOB.$$

Thus, enforcing $\angle APB = 40°$ can be achieved by setting $\angle AOB = 140°$ in a *normalized coordinate system*. Importantly, this $140°$ is *not* a literal number read from the text; it is an intermediate value introduced to instantiate one valid configuration.

Executable GeoDSL construction

Listing 2: A valid GeoDSL construction for Example A (normalized instance).

```
point : 0 0 -> O
point : 100 0 -> A
point : 100*cos(140°) 100*sin(140°) -> B

circle : O A -> circle_O
line : O A -> line_OA
line : O B -> line_OB

orthogonal_line : A line_OA -> tangent_A
orthogonal_line : B line_OB -> tangent_B
intersect : tangent_A tangent_B -> P

rotate : A 180° O -> C
segment : P A -> PA
segment : P B -> PB
segment : A C -> AC
```

Rendered diagram. Figure 2 shows the diagram rendered by our execution engine.

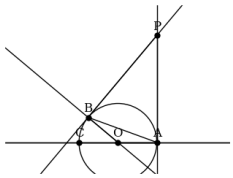

Figure 2: Example A (constructible): a rendered diagram from the GeoDSL program in Listing 2. The coordinates instantiate one normalized configuration; correctness is judged by verified geometric relations (tangency via perpendicularity to radii, and $\angle APB = 40°$), not by matching a unique canonical layout.

F.2 Example B: Well-Formed but Inconstructible

Problem. *In triangle $\triangle ABC$, BD is a median. Given $AB = 5$, $BC = 3$, and the perimeter of triangle $\triangle ABD$ equals 11, determine the perimeter of triangle $\triangle BCD$.*

Claim. There exists no triangle $\triangle ABC$ satisfying all of the following conditions simultaneously:

- BD is a median of $\triangle ABC$ (so D is the midpoint of AC),
- $AB = 5$,
- $BC = 3$,
- the perimeter of $\triangle ABD$ is 11.

Proof. Since BD is a median, point D is the midpoint of AC, and hence

$$AD = DC = \frac{AC}{2}.$$

Let

$$AD = x \quad \text{and} \quad BD = 6 - x,$$

which follows from the perimeter condition of $\triangle ABD$:

$$\begin{aligned} &AB + BD + AD = 11 \\ &\Rightarrow 5 + BD + AD = 11 \\ &\Rightarrow BD + AD = 6 \end{aligned}$$

Applying Apollonius' theorem to triangle $\triangle ABC$ with median BD, we obtain

$$AB^2 + BC^2 = 2\left(BD^2 + \left(\frac{AC}{2}\right)^2\right).$$

Substituting the known values,

$$25 + 9 = 2\left((6 - x)^2 + x^2\right).$$

Simplifying yields

$$2x^2 - 12x + 19 = 0.$$

The discriminant of this quadratic equation is

$$\Delta = (-12)^2 - 4 \cdot 2 \cdot 19 = 144 - 152 = -8 < 0.$$

Since the discriminant is negative, the equation has no real solutions. Therefore, no real values of AD and BD satisfy all given constraints.

Conclusion. No triangle $\triangle ABC$ exists that satisfies the stated conditions. Although the problem appears coherent and complete in natural language, its geometric constraints are internally inconsistent. Such problems are explicitly excluded during the dataset curation process described in Section 4.1.

G Detailed Evaluation Diagnostics

This appendix provides detailed diagnostic analyses that complement the main results in Section 5. While the main text focuses on high-level performance trends, the analyses here expose *how* and *why* models fail, with a particular emphasis on structural hallucinations, missing constructions, constraint violations, and feedback utilization in the agent–environment loop.

Table 9: Overall diagnostics on GeoBuildBench. All runs use a maximum of 5 interaction steps. Lower is better (↓) for hallucinations, recovery steps, missing objects, and failed conditions.

Model	Success (%)	Total Problems	Hallucinations / Prob. ↓	Avg. Halluc. Recovery ↓	Total Missing Objects ↓	Total Failed Conditions ↓	Avg. Success Steps ↓
GPT-5.1	78.9	493	0.401	1.290	939	1119	1.874
Gemini-3-Flash	75.3	489	0.335	1.175	329	932	1.554
Qwen3-VL-235B	42.2	491	2.303	1.743	2042	1817	2.304
LLaMA-3.2-90B-Vision	21.3	494	2.381	1.794	1823	1584	2.229

G.1 Overall Diagnostics

Table 9 summarizes overall diagnostic statistics across all evaluated models. In addition to success rate, we report hallucination frequency, hallucination recovery behavior, and the total number of missing objects and failed constraints. As discussed in Section 5, models with higher success rates also exhibit substantially fewer hallucinations and faster recovery from errors, indicating more effective use of environment feedback.

G.2 Structural Hallucination Types

Table 10 reports the breakdown of structural hallucination types. Across all models, undefined references dominate, indicating difficulty in maintaining a consistent internal object graph during multi-step construction. Syntax errors and output mismatches are comparatively less frequent and rarely constitute the primary failure mode.

G.3 Missing Required Objects

Table 11 shows the number of missing required objects by geometric type. Missing points and segments account for the majority of omissions, suggesting that models often fail to explicitly instantiate all entities described in the text, even when auxiliary constructions are allowed.

G.4 Failed Verification Conditions

To keep the presentation compact, Tables 12 – 15 report the top ten most frequently failed verification conditions for each model. Angle-value constraints, incidence relations (e.g., point-on-segment or point-on-line), and metric constraints dominate across models, reinforcing the observation that failures are primarily semantic rather than syntactic.

Table 12: Top failed verification conditions for GPT-5.1 (counts).

Condition Type	Count
angle_value	304
point_on_circle	135
point_on_segment	131
point_on_line	96
distance_equals	68
segment_equality	66
perpendicular	64
parallel	58
midpoint_of	29
angle_bisector	26

Table 13: Top failed verification conditions for Gemini-3-Flash (counts).

Condition Type	Count
angle_value	332
point_on_segment	110
distance_equals	108
point_on_line	86
segment_equality	49
parallel	42
perpendicular	37
midpoint_of	26
concurrent	23
angle_bisector	19

Table 14: Top failed verification conditions for Qwen3-VL-235B (counts).

Condition Type	Count
angle_value	473
distance_equals	247
point_on_circle	190
point_on_segment	187
perpendicular	99
point_on_line	91
parallel	88
triangle_valid	85
angle_bisector	74
segment_equality	65

Table 10: Breakdown of structural hallucination types (counts).

Model	Undefined Ref.	Syntax Error	Output Mismatch
GPT-5.1	87	33	74
Gemini-3-Flash	95	46	20
Qwen3-VL-235B	554	297	77
LLaMA-3.2-90B-Vision	486	264	69

Table 11: Missing required objects by object type (counts).

Model	Points	Segments	Lines	Circles	Polygons
GPT-5.1	406	286	128	35	84
Gemini-3-Flash	135	96	26	6	66
Qwen3-VL-235B	862	736	244	50	150
LLaMA-3.2-90B-Vision	779	637	225	50	132

Table 15: Top failed verification conditions for LLaMA-3.2-90B-Vision (counts).

Condition Type	Count
angle_value	439
point_on_circle	202
distance_equals	175
point_on_segment	166
parallel	88
perpendicular	79
point_on_line	74
angle_bisector	63
segment_equality	60
triangle_valid	54

G.5 Feedback Utilization Indicators

Finally, Table 16 isolates two indicators of feedback utilization: hallucinations per problem and the average number of steps required to recover from hallucinations. Lower values indicate that a model not only makes fewer structural errors but also more effectively incorporates visual and constraint-based feedback when errors occur.

Table 16: Feedback utilization indicators.

Model	Hallucinations / Prob. ↓	Avg. Halluc. Recovery ↓
GPT-5.1	0.401	1.290
Gemini-3-Flash	0.335	1.175
Qwen3-VL-235B	2.303	1.743
LLaMA-3.2-90B-Vision	2.381	1.794