Hilbert-Geo: Solving Solid Geometric Problems by Neural-Symbolic Reasoning

Supplementary Material

A. Solid Geometry Formal Language

A.1. Formal Geometry Representation

In the domain of solid geometry, simple geometric bodies serve as fundamental units that are combined into complex geometries via combinatorial strategies. This section provides a formal proof that the solid construction operation constitutes a commutative semigroup.

Previous work on Geometry Formalization Theory has shown that the formal representation of plane geometry is a semigroup (Section. 3.1). In solid geometry, we elevate this representation to a set of faces. The plane geometric category is encapsulated as an entity, which serves as the foundational reference for constructing solid geometric models.

Let the formal representation of a simple polyhedron $\mathcal{P}$ be a set R_P (Here, the polyhedron refers to a figure that is topologically homeomorphic to a $\mathbb{R}^3$ sphere):

$$R_P = \{f_1^{(P)}, f_2^{(P)}, \ldots, f_m^{(P)}\} \tag{2}$$

where each face $f_i^{(P)}$ is an ordered list of vertices (following the right-hand rule with outward normal vectors) describing the boundary of that face:

$$f_i^{(P)} = (v_1, v_2, \ldots, v_k) \tag{3}$$

Definition of Composition Operation ($\oplus_{3D}$): This is the foundation of the combined strategy, let polyhedra A and B be joined via a shared interface. Let S_A denote the shared face in A, and S_B denote the corresponding shared face in B (note: geometrically, the vertex order of S_A is the reverse of S_B). The solid composition operation is defined as the set union minus the symmetric intersection (the internal contact faces), as shown in Eq. 4:

$$R_A \oplus_{3D} R_B = (R_A \setminus \{S_A\}) \cup (R_B \setminus \{S_B\}) \tag{4}$$

Theorem A.1. *The formal representation of solid geometry is closed under the operation $\oplus_{3D}$.*

Proof. By definition, a valid object in the solid formal system is a "set of faces bounding a closed 3D space".

1. Let R_A and R_B be face sets of two valid polyhedra.
2. Execute the operation $R_{result} = R_A \oplus_{3D} R_B$.
3. This operation removes the internal contact faces (S_A and S_B) and retains all external surfaces.
4. According to the generalization of Euler's formula for manifolds, when two closed manifolds are glued along a simply connected face and that face is removed, the remaining surface still constitutes a closed 2-manifold (i.e., the boundary of the new polyhedron).
5. Therefore, R_{result} remains a set of faces describing a closed solid.

$$\forall R_A, R_B \in \mathbb{S}, \quad (R_A \oplus_{3D} R_B) \in \mathbb{S} \tag{5}$$

□

Theorem A.2. $R_A \oplus_{3D} R_B = R_B \oplus_{3D} R_A$.

Proof. Let R_A contain m faces and R_B contain n faces. Based on Eq. 4:

$$R_A \oplus_{3D} R_B = \{f \mid f \in R_A \cup R_B, f \neq S_A, f \neq S_B\} \tag{6}$$

Now consider the reverse operation $R_B \oplus_{3D} R_A$:

$$R_B \oplus_{3D} R_A = (R_B \setminus \{S_B\}) \cup (R_A \setminus \{S_A\}) \tag{7}$$

According to set algebra, the union operation is commutative, i.e., $X \cup Y = Y \cup X$. Therefore:

$$(R_A \setminus \{S_A\}) \cup (R_B \setminus \{S_B\}) = (R_B \setminus \{S_B\}) \cup (R_A \setminus \{S_A\}) \tag{8}$$

Substituting back into Eq. 6 and Eq. 7, we obtain:

$$R_A \oplus_{3D} R_B = R_B \oplus_{3D} R_A \tag{9}$$

This indicates that the construction of solid figures is independent of the input order of components. □

Theorem A.3. $(R_A \oplus_{3D} R_B) \oplus_{3D} R_C = R_A \oplus_{3D} (R_B \oplus_{3D} R_C)$.

Proof. setup:

1. Polyhedra A and B share interface faces (S_{AB}, S_{BA}).
2. Polyhedra B and C share interface faces (S_{BC}, S_{CB}).
3. R_A, R_B, R_C are their respective face sets.

Left Hand Side: Let $R_{AB} = R_A \oplus_{3D} R_B$.

$$R_{AB} = (R_A \cup R_B) \setminus \{S_{AB}, S_{BA}\} \tag{10}$$

Next, calculate $R_{AB} \oplus_{3D} R_C$. The contact interface involves B and C (i.e., S_{BC} and S_{CB}):

$$(R_A \oplus_{3D} R_B) \oplus_{3D} R_C = (R_{AB} \cup R_C) \setminus \{S_{BC}, S_{CB}\} \tag{11}$$

$$= ((R_A \cup R_B) \setminus \{S_{AB}, S_{BA}\} \cup R_C) \setminus \{S_{BC}, S_{CB}\} \tag{12}$$

Since S_{AB}, S_{BA} exist only between A and B and do not affect C, the formula simplifies to the total union minus all interface faces:

$$= (R_A \cup R_B \cup R_C) \setminus \{S_{AB}, S_{BA}, S_{BC}, S_{CB}\} \tag{13}$$

Right Hand Side: Let $R_{BC} = R_B \oplus_{3D} R_C$.

$$R_{BC} = (R_B \cup R_C) \setminus \{S_{BC}, S_{CB}\} \tag{14}$$

Next, calculate $R_A \oplus_{3D} R_{BC}$. The contact interface involves A and B:

$$R_A \oplus_{3D} (R_B \oplus_{3D} R_C) = (R_A \cup R_{BC}) \setminus \{S_{AB}, S_{BA}\} \tag{15}$$

$$= (R_A \cup ((R_B \cup R_C) \setminus \{S_{BC}, S_{CB}\})) \setminus \{S_{AB}, S_{BA}\} \tag{16}$$

Similarly, this simplifies to the total union minus all interface faces:

$$= (R_A \cup R_B \cup R_C) \setminus \{S_{AB}, S_{BA}, S_{BC}, S_{CB}\} \tag{17}$$

Comparing Eq. 13 and Eq. 17, the results are identical:

$$(R_A \oplus_{3D} R_B) \oplus_{3D} R_C = R_A \oplus_{3D} (R_B \oplus_{3D} R_C) \tag{18}$$

□

A.2. Predicate Library and Theorem Bank

A.2.1. Predicate Library

The Solid Predicate Library encompasses 120 predicates, comprising 35 fundamental predicates (Table 3) natively integrated into the solver, along with 20 entity types (Table 4), 35 entity relationships (Table 5), and 30 attribute descriptors (Table 6) defined via a dedicated predicate.

Table 3. fundamental predicates

id	type	name	examples
1	Construction	Coplanar	Coplanar(ABCD)
2	Construction	Cospherical	Cospherical(O,ABC)
3	Construction	Collinear	Collinear(ABC)
4	Construction	Cocircular	Cocircular(O,ABC)
5	BasicEntity	Plane	Plane(U)
6	BasicEntity	Sphere	Sphere(O)
	.....	.....	.....

Table 4. Entities defined for solid geometry using the predicate

id	type	examples
1	Entity	Cube(ABCDEFGH)
2	Entity	Cuboid(ABCDEFGH)
3	Entity	Cone(O,P)
4	Entity	Cylinder(P,Q)
5	Entity	Prism(P,Q)
6	Entity	TriangularPrism(ABC,DEF)
	.....	.....

The detailed statements for defining a predicate is as shown in the Tab. 7, including the predicate name and point variable declaration, validity check declaration, multiple representations, and automatic expansion.

Table 5. Relations defined for solid geometry using the predicate

id	type	examples
1	Relation	ParallelBetweenPlane(U,V)
2	Relation	PerpendicularBetweenPlane(U,V)
3	Relation	PerpendicularBetweenLineAndPlane(AB,U)
4	Relation	IsDiameterOfSphere(AB,O)
5	Relation	IsTangentOfSphere(PA,O)
6	Relation	IsCentreOfSphere(P,O)
	.....	.....

Table 6. Attributions defined for solid geometry using the predicate

id	type	examples
1	Attribution	HeightOfCone(O,P)
2	Attribution	HeightOfCylinder(P,Q)
3	Attribution	HeightOfPrism(P,Q)
4	Attribution	HeightOfTriangularPrism(ABC,DEF)
5	Attribution	RadiusOfSphere(O)
6	Attribution	DiameterOfSphere(O)
	.....	.....

A.2.2. Theorem Bank

As shown in Table 8, this section presents some representative theorems from the theorem library, demonstrating the formal representation of geometric knowledge used in the automated reasoning system.

These theorems focus on line-plane relationships and solid geometry: plane-to-plane relationships (transitivity), line-to-plane relationships (perpendicular and parallel judgments), sphere formulas (area and volume), and cylinder properties (volume and height judgments). The theorem library contains over 200 such theorems covering comprehensive geometric knowledge from basic properties to advanced relationships.

Table 7. example for defining a predicate

name	item	content
IsMidpointOfLine(M,AB)	check	Point(M)
		Line(AB)
		Collinear(AMB)
	multi	M,BA
	extend	Equal(LengthOfLine(AM),LengthOfLine(MB))

Table 8. Examples from the theorem library.

theorem	**premise**	**conclusion**
transitivity_between_plane_and_plane	ParallelBetweenPlane(U,V) & ParallelBetweenPlane(V,W)	ParallelBetweenPlane(U,W)
perpendicular_judgment_between_line_and_plane	PerpendicularBetweenLine(AD,BD) & PerpendicularBetweenLine(AD,CD) & Coplanar(U,BCD) & ∼Coplanar(U,AD)	PerpendicularBetweenLineAndPlane(AD,U)
perpendicular_judgment_between_plane_and_plane	PerpendicularBetweenLineAndPlane(AB,U) & Coplanar(V,AB)	PerpendicularBetweenPlane(U,V)
parallel_judgment_between_line_and_plane	ParallelBetweenLine(AB,CD) & Coplanar(U,CD) & ∼Coplanar(U,AB)	ParallelBetweenLineAndPlane(AB,U)
sphere_area_formula	Sphere(O)	Equal(AreaOfSphere(O),Mul(4,pi,RadiusOfSphere(O), RadiusOfSphere(O)))
sphere_volume_formula	Sphere(O)	Equal(VolumeOfSphere(O),Mul(4/3,pi, RadiusOfSphere(O),RadiusOfSphere(O), RadiusOfSphere(O)))
cylinder_volume_formula_common	Cylinder(P,Q)	Equal(VolumeOfCylinder(P,Q),Mul(AreaOfCircle(P), HeightOfCylinder(P,Q)))
height_of_cylinder_judgment	Cylinder(P,Q) & Coplanar(P,A) & Coplanar(Q,B) & PerpendicularBetweenLineAndPlane(AB,P)	Equal(LengthOfLine(AB),HeightOfCylinder(P,Q))
.....	.....	.....

B. Dataset

B.1. Data and Annotation

SolidFGeo2k (Fig. 13) presents examples of geometric problems under this benchmark, covering various scenarios and tasks related to solid geometry; PlaneFGeo3k (Fig. 14) focuses on the field of plane geometry; MathVerse-solid (Fig. 15) is a representative resource among existing geometric benchmarks, providing abundant supplementary data support for solid geometry tasks. Together, the three constitute a multi-scenario and multi-level dataset system for geometric tasks.

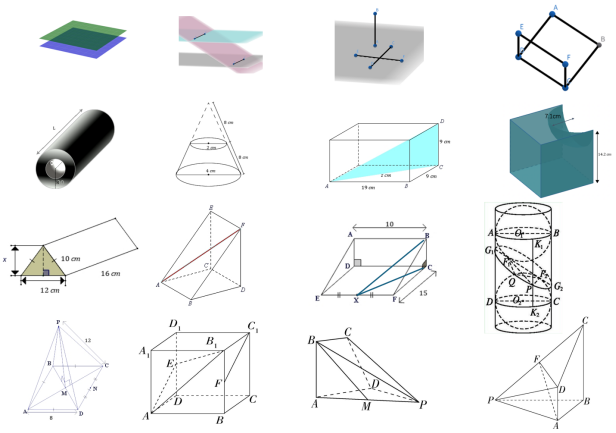

Figure 13. Examples from SolidFGeo2K

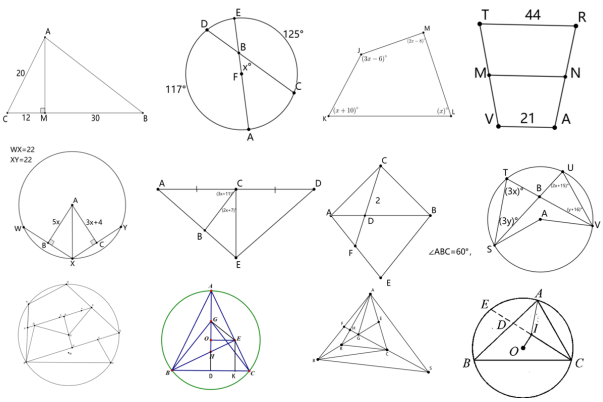

Figure 14. Examples from PlaneFGeo3k

To convert raw problem content into actionable resources for geometric formalization and model training, datasets are systematically annotated manually by experts. This annotation work adheres to standardized protocols and uses professional tools. Meanwhile, formal language encoding is adopted to convert the annotated information into Geometric Description Language. As shown in Figure 16, the geometric description language and the standard answers

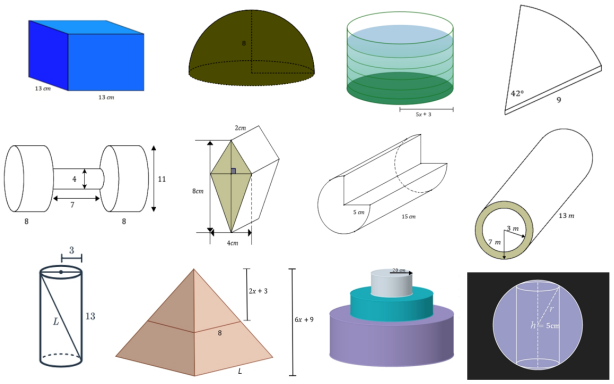

Figure 15. Examples from Mathverse-solid

Figure 16. Data Samples from Mathverse-solid and SolidFGeo2k

are annotated based on the original text and images, which serve as the ground truth.

B.2. Data Construction

In our dataset SolidFGeo2k, we collect solid geometric problems in existing datasets except SolidGeo (containing other datasets) as shown in Fig 17. In addition, we add new samples (31.1%) for the comprehensive evaluation. For each problem, we ask **two undergraduat students** to give detailed annotations including parsing results, reasoning process, and correct answer. For your reference, we show differenes to existing datasets in Fig 17.

Attribute	MathVision	CMMatH	DynaMath	MathVerse	OlympiadBench	GeoEval	MV-Math	SolidGeo	SolidFGeo2k (ours)
Size	203	122	118	296	448	63	59	2200	1908
Fine-grained Label	Subject	Knowledge Point	Question Type	Info Type	Task Type	Eval Subset	Question Type	Sub-domain	Sub-domain
Parsing Annotation	✗	✗	✗	✗	✗	✗	✗	✗	✓
Reasoning Process	✓	✓	✗	✗	✓	✗	✗	✗	✓
LLM Generation	✓	✓	✗	✗	✗	✓	✗	✓	✗

Figure 17. Comparison of Datasets for Solid Geometry Problems

B.3. Fine-grained Subject categorization

The SolidFGeo2k and Mathverse-solid dataset incorporates fine-grained subject categorization, covering the full spectrum of spatial reasoning skills required. Figure 18 categorizes SolidFGeo2k's content into themes: Composite Solid Structure, Spatial Metric Relations, Solid Shape Identification, and Measurement of Solid Geometric Forms, demon-

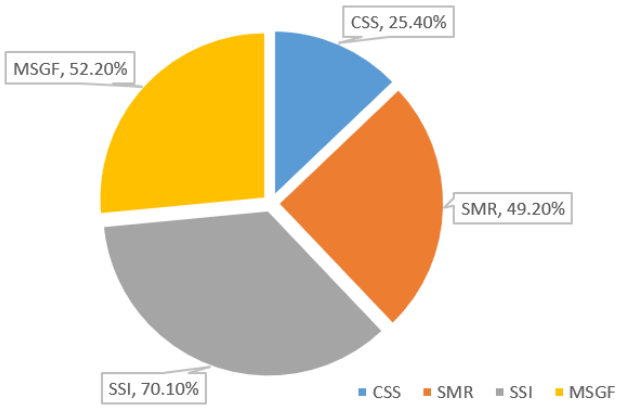

Figure 18. Proportion of different subjects in SolidFGeo2k, note that a single question may match multiple different subjects (Fig. 19)

strating the dataset's coverage of core solid geometry topics. Figure 18's pie chart quantifies the proportion of different subjects in SolidFGeo2k: SSI (Solid Shape Identification) accounts for 70.10%, SMR (Spatial Metric Relations) for 49.20%, CSS (Composite Solid Structure) for 25.40%, and MSGF (Measurement of Solid Geometric Forms) for 52.20%. Note that a single problem may belong to multiple subjects, reflecting the interdisciplinary nature of geometric reasoning tasks.

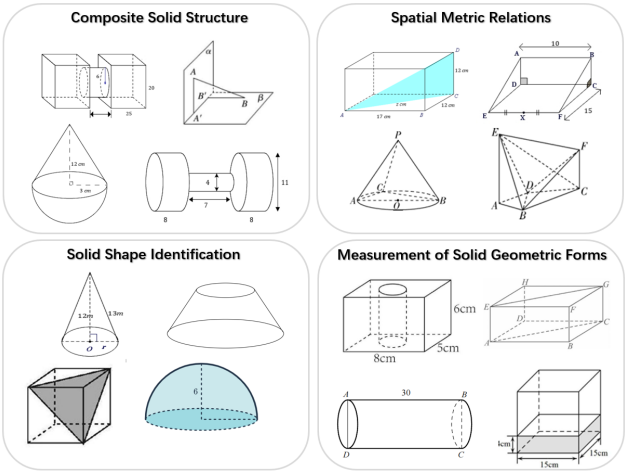

Figure 19. Examples from different subjects

C. Multimodal Formalization Parser (M2FP) Supplementary Information

C.1. M2FP Pipeline Architecture

C.1.1. Overview and Objectives

The Multimodal Formalization Parser (M2FP) translates natural language geometric problem descriptions and visual diagrams into Conditional Description Language (CDL), enabling precise computational reasoning. It integrates information from text and images, converts descriptions into predicate-based CDL format, and ensures completeness and consistency across modalities.

C.1.2. Output Structure

The parser produces a structured CDL representation with four components:

1. **`construction_cdl`**: Geometric construction predicates defining fundamental structure (e.g., `Shape(AB,BC,CD,DA)`, `Collinear(PABQ)`, `Cocircular(O)`)
2. **`text_cdl`**: Predicates extracted from natural language (e.g., `Equal(LengthOfLine(A,B),5)`, `ParallelBetweenLine(A,B,C,D)`)
3. **`image_cdl`**: Predicates derived from visual analysis (e.g., `PerpendicularBetweenLine(AB,BC)`, `Equal(RadiusOfCircle(O),3)`)
4. **`goal_cdl`**: A single canonicalized predicate representing the problem's objective (e.g., `Value(VolumeOfCone(O,P))`)

The system ensures entities in `text_cdl` and `image_cdl` are declared in `construction_cdl`, maintaining cross-modal consistency.

C.1.3. Pipeline Components

Stage 1: Text Parsing The text parsing module extracts entities (points, lines, shapes), maps relations (e.g., "parallel" → `ParallelBetweenLine(A,B,C,D)`), parameterizes attributes (e.g., "length of AB is 5" → `Equal(LengthOfLine(A,B),5)`), identifies goals (e.g., "Find the volume" → `Value(VolumeOfCone(O,P))`), and normalizes the output format.

Stage 2: Image Parsing The image parsing module detects primitives (points, lines, shapes), recognizes visual relations (right-angle marks, parallel indicators, intersections), parses numeric annotations from diagram labels, aligns entities with text mentions, resolves conflicts, and encodes visual information into CDL format consistent with text-derived predicates.

Stage 3: Output Packing The output packing stage consolidates all information into a JSON structure with four CDL components. Numeric values are formatted without units (supporting expressions like `36*pi`), entities use uppercase letters, and formatting follows strict rules (no extra spaces). The system validates that all entities are declared in `construction_cdl` and all predicates conform to the official vocabulary.

C.2. Fuzzy Matching Evaluation Metrics

We employ fuzzy set-based metrics to evaluate CDL parsing outputs, allowing partial credit for semantically similar predicates that may differ in variable naming or formatting. This section provides complete definitions enabling full reproducibility of the evaluation metrics.

C.2.1. Element Normalization Process

Before computing similarity scores, all CDL elements must be normalized to eliminate differences caused by variable naming conventions. Given a raw CDL set $P = \{p_1, p_2, \ldots, p_n\}$ (predicted) or $G = \{g_1, g_2, \ldots, g_m\}$ (ground truth), we apply a normalization function Normalize$(\cdot)$ to obtain normalized sets $P' = \{\text{Normalize}(p) \mid p \in P\}$ and $G' = \{\text{Normalize}(g) \mid g \in G\}$.

The normalization process follows these steps:

1. **Remove whitespace**: Strip all spaces from the element string
2. **Replace variables**: Replace all variable names (single uppercase letters like A, B, C, O, P, etc.) with the placeholder `_V_`
3. **Preserve numeric values**: Keep all numeric values unchanged
4. **Process nested structures**: Recursively apply normalization to nested predicate structures
5. **Normalize variable-only parameters**: If all parameters are variables, replace with `(_V_)`

Examples:

- `Equal(LengthOfLine(A,B),5)` → `Equal(LengthOfLine(_V_),5)`
- `Equal(HeightOfCone(O,P),12)` → `Equal(HeightOfCone(_V_),12)`
- `Shape(AB,BC,CD,DA)` → `Shape(_V_,_V_,_V_,_V_)`
- `Collinear(PABQ)` → `Collinear(_V_)`

The complete normalization algorithm is provided in Algorithm 2.

C.2.2. Fuzzy Matching Score Function

The core fuzzy matching score function $S(p', g') \in [0, 1]$ quantifies the similarity between a normalized predicted element $p' \in P'$ and a normalized ground truth element $g' \in G'$. This function is formally defined as:

$$S(p', g') = \begin{cases} 1.0 & \text{if } p' = g' \text{ (exact match)} \\ 0.8 & \text{if predicate\_name}(p') = \text{predicate\_name}(g') \\ & \quad \wedge \text{numbers}(p') \cap \text{numbers}(g') \neq \emptyset \\ 0.5 & \text{if predicate\_name}(p') = \text{predicate\_name}(g') \\ & \quad \text{only} \\ 0.3 & \text{if numbers}(p') \cap \text{numbers}(g') \neq \emptyset \\ & \quad \text{only} \\ 0.0 & \text{otherwise} \end{cases} \tag{19}$$

where:

- predicate_name(x) extracts the predicate name (the substring before the first parenthesis) from normalized element x
- numbers(x) extracts all numeric values from normalized element x using regular expressions
- The intersection numbers$(p') \cap$ numbers$(g') \neq \emptyset$ means at least one numeric value appears in both elements

Examples:

- $S($`Equal(_V_,5)`$,$ `Equal(_V_,5)`$) = 1.0$ (exact match)
- $S($`Equal(_V_,5)`$,$ `Equal(_V_,5.0)`$) = 0.8$ (predicate match + number intersection)
- $S($`Equal(_V_,5)`$,$ `Equal(_V_,3)`$) = 0.5$ (predicate match only)
- $S($`Equal(_V_,5)`$,$ `LengthOf(_V_,5)`$) = 0.3$ (number intersection only)
- $S($`Equal(_V_,5)`$,$ `LengthOf(_V_,3)`$) = 0.0$ (no match)

C.2.3. Fuzzy Jaccard Similarity

Given normalized sets P' and G' (obtained from raw sets P and G via the normalization process in Section C.2.1), and using the scoring function $S(p', g')$ defined in Section C.2.2, the Fuzzy Jaccard Similarity extends the standard Jaccard index by replacing exact set intersection with a fuzzy intersection. The fuzzy Jaccard score is computed as:

$$\text{Score}(P, G) = \frac{\sum_{p' \in P'} \max_{g' \in G'} S(p', g')}{|P'| + |G'| - \sum_{p' \in P'} \max_{g' \in G'} S(p', g')} \tag{20}$$

where:

- The numerator $\sum_{p' \in P'} \max_{g' \in G'} S(p', g')$ represents the **fuzzy intersection**: the sum of best-match scores for each predicted element
- The denominator $|P'|+|G'|-\sum_{p' \in P'} \max_{g' \in G'} S(p', g')$ represents the **fuzzy union**: the sum of set sizes minus the intersection

Algorithm 2 CDL Element Normalization

```
function NormalizeCDLElement(element)
    t ← strip spaces(element)
    if no parentheses in t then
        if is number(t) then
            return t
        else
            return replace all variables with _V_
        end if
    else
        Process innermost parentheses recursively
        parts ← split by commas within parentheses
        for each part in parts do
            if is number(part) then
                Keep numerical value
            else if contains parentheses then
                part ← NormalizeCDLElement(part)
            else
                part ← _V_
            end if
        end for
        if all parts are _V_ then
            return (_V_)
        else
            return concatenate processed parts
        end if
    end if
end function
```

- P' and G' are obtained by applying normalization to raw sets P and G as described in Section C.2.1

The complete computation algorithm is provided in Algorithm 3.

C.2.4. Boundary Case Handling

Boundary cases are handled as follows: if both sets are empty, Jaccard = 1.0; if one set is empty, Jaccard = 0.0; division by zero is prevented with explicit checks.

C.2.5. Evaluation Algorithms

The evaluation framework implements three core algorithms: normalization (Algorithm 2), Fuzzy Jaccard calculation (Algorithm 3), and score matrix construction (Algorithm 4).

C.2.6. Computational Efficiency

The algorithm computes a $|P'| \times |G'|$ score matrix with $O(|P'| \cdot |G'|)$ time complexity, providing a balance between matching quality and efficiency.

Algorithm 3 Fuzzy Jaccard Similarity Calculation

function FUZZYJACCARD(P, G)
  $P' \leftarrow \{\text{NORMALIZECDLELEMENT}(p) \mid p \in P\}$
  $G' \leftarrow \{\text{NORMALIZECDLELEMENT}(g) \mid g \in G\}$
  if boundary case detected **then**
    return boundary Jaccard value (as described in Section C.2.4)
  end if
  $S \leftarrow \text{BUILDSCOREMATRIX}(P', G')$
  $row\_max[p'] \leftarrow \max_{g' \in G'} S[p', g']$ for each $p' \in P'$
  $intersection \leftarrow \sum_{p' \in P'} row\_max[p']$
  $union \leftarrow |P'| + |G'| - intersection$
  $jaccard \leftarrow intersection / union$
  return $jaccard$
end function

Algorithm 4 Matching Score Matrix Construction

function BUILDSCOREMATRIX(P', G')
  Initialize S as $|P'| \times |G'|$ matrix
  for each $p' \in P'$ and each $g' \in G'$ **do**
    if $p' = g'$ **then**
      $S[p', g'] \leftarrow 1.0$
    else
      $pred\_name \leftarrow$ predicate name(p')
      $gt\_name \leftarrow$ predicate name(g')
      $pred\_nums \leftarrow$ extract numbers(p')
      $gt\_nums \leftarrow$ extract numbers(g')
      $name\_equal \leftarrow (pred\_name = gt\_name)$
      $num\_intersect \leftarrow$ intersects($pred\_nums$, gt_nums)
      if $name\_equal$ and $num\_intersect$ **then**
        $S[p', g'] \leftarrow 0.8$
      else if $name\_equal$ **then**
        $S[p', g'] \leftarrow 0.5$
      else if $num\_intersect$ **then**
        $S[p', g'] \leftarrow 0.3$
      else
        $S[p', g'] \leftarrow 0.0$
      end if
    end if
  end for
  return S
end function

C.3. Parsing Performance Across Models and Sample Sizes

This section presents comprehensive experimental results evaluating the parsing performance of various Multimodal Large Language Models (MLLMs) under different few-shot sample configurations. The evaluation employs fuzzy matching metrics as described in Section **??** to assess the quality of CDL parsing across construction predicates, condition predicates (merged text and image), and goal identification.

C.3.1. Parse Score: Comprehensive Performance Metric

The **Parse Score** serves as a comprehensive performance metric that aggregates the parsing quality across three critical components of geometric problem formalization: **Construction**, **Condition**, and **Goal**. The Parse Score is computed as the expectation (average) of the Jaccard similarity scores across these three components:

$$\text{Parse Score} = \frac{1}{3} \sum_{c \in \mathcal{I}} \text{Jaccard}_c \tag{21}$$

where Jaccard_c denotes the Jaccard similarity score for component c, $\mathcal{I} = \{\text{Construction}, \text{Condition}, \text{Goal}\}$

C.3.2. Component-wise Parsing Performance

To provide detailed insights into the parsing performance, we examine each component individually. The following figures present the Jaccard similarity scores for Construction, Condition, and Goal parsing, respectively, allowing for granular analysis of model strengths and weaknesses across different aspects of geometric formalization.

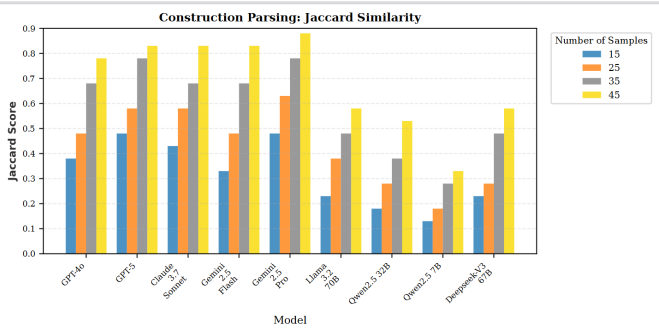

Figure 20. Construction Parsing Performance: Jaccard similarity scores across different models and sample sizes. Construction parsing evaluates the accuracy of extracting geometric structure predicates (e.g., `Shape`, `Collinear`, `Cocircular`). All metrics use fuzzy matching as described in Section **??**.

These three components collectively contribute to the Parse Score, which is computed as their average.

C.3.3. Performance Analysis

The experimental results (Figures 20, 21, and 22) show that Parse Score generally improves as sample size increases from 15 to 45 across all models. Construction parsing achieves the highest scores due to the structured nature of construction predicates, while goal parsing remains the most challenging task. Closed-source models consistently

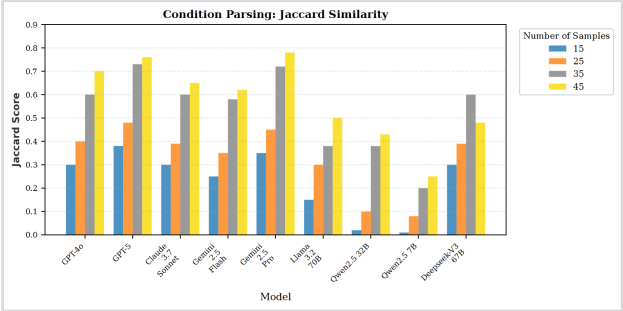

Figure 21. Condition Parsing Performance: Jaccard similarity scores across different models and sample sizes. Condition parsing measures the quality of extracting geometric constraints from problem descriptions and visual diagrams (merged text and image). All metrics use fuzzy matching as described in Section **??**.

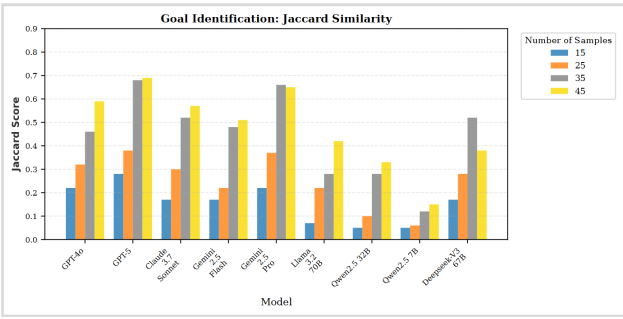

Figure 22. Goal Parsing Performance: Jaccard similarity scores across different models and sample sizes. Goal parsing assesses the precision of identifying the problem's objective. All metrics use fuzzy matching as described in Section **??**.

outperform open-source models, with larger models showing better performance within the same model family. The Parse Score effectively captures overall parsing quality by equally weighting the three components, ensuring comprehensive parsing capability is rewarded.

C.4. Prompt Templates

This section introduces the specific prompt templates used in the M2FP system for CDL parsing and direct problem solving. The M2FP system uses prompt engineering to guide large language models: (1) task description with role definition and schema compliance, (2) strict predicate vocabulary constraints from GDL, (3) few-shot examples covering all predicate types, (4) rule-based guidance for source separation and formatting, and (5) error prevention warnings. This approach balances flexibility with strict constraints to ensure output quality.

C.4.1. CDL Parsing Prompt Template

The following prompt template is used for **CDL parsing**, which guides large language models to convert natural language geometric problem descriptions and visual diagrams into formal CDL representations. Parsing step starts with a targeted prompt engineering approach: specialized prompts are constructed using expertly designed example sets that cover all basic predicates. These examples can express predicates and formal language without involving complex geometric relationships, and each includes geometric descriptions and their corresponding formalizations.

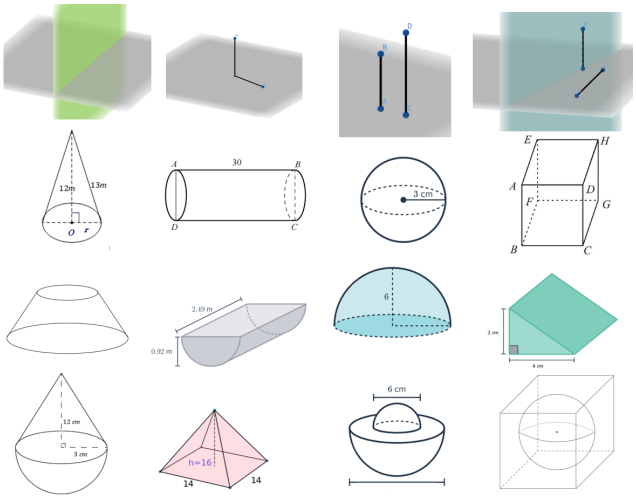

Figure 23. Examples from designed samples

The complete prompt template is shown below:

```
You are an expert in geometry, logic, and
computer science. Your task is to precisely
convert a geometry problem (with natural
language and an image) into a JSON object
following the provided JSON Schema. You
must strictly follow the schema and output
a complete JSON object.

Rule 0: Predicate Compliance
(MOST IMPORTANT)
- All CDL predicates you generate
  (e.g., Equal, Cone, LengthOfLine) MUST
  be strictly chosen from the official
  list below.
- Using any predicate that does not
  appear in this list is strictly
  forbidden.

--- Official Predicate List ---
[valid_predicates_str]
--- End of Official Predicate List ---

Core Rules and Constraints:
```

```
1) Information Source Separation:
   - text_cdl MUST include only facts
     extracted from the natural language
     description.
   - image_cdl MUST include only facts
     directly observable from the image
     (e.g., length labels, right-angle
     marks, shape recognition).
   - If a fact appears in both text and
     image, include it in both fields.

2) construction_cdl - Geometric
   construction predicates (IMPORTANT):
   construction_cdl defines basic
   construction for entities, and MUST
   include the following types where
   applicable:
   - Shape predicates: define
     edges/segments of shapes
     * For segments/edges:
       Shape(AB,BC,CD,DA) or Shape(OP,PO)
       or Shape(PQ,QP)
     * For points (spheres etc.):
       Shape(O) or Shape(P)
     * Example: rectangles require
       Shape(AB,BC,CD,DA); cylinders
       require Shape(PQ,QP)
   - Collinearity/Cocircular/Coplanar/
     Cospherical:
     * Collinear(PABQ) - P, A, B, Q are
       collinear
     * Cocircular(O) - O is on a circle
       (for cone/cylinder base center)
     * Coplanar(U,ABCD) - U coplanar with
       ABCD
     * Cospherical(O) - O is on a sphere
       (for spheres)
   Important:
   - Carefully analyze the image to
     identify all necessary
     edges/segments/relations
   - Most problems require at least one
     Shape(...)
   - Cones/cylinders often need Shape(...)
     and Cocircular(...)
   - Spheres often need Shape(O) and
     Cospherical(O)

3) Answer formatting:
   - problem_answer MUST be a pure number
     or expression (e.g., "10", "36*pi"),
     and MUST NOT contain units or extra
     text.

4) Core predicate logic:
   - Length/Height:
     Equal(LengthOfLine(A,B),5),
     Equal(HeightOfCone(O,P),12)
   - Relations:
     PerpendicularBetweenLine(A,B,C,D),
     ParallelBetweenLine(A,B,C,D)
   - Goal: the requested quantity MUST be
     wrapped by Value(...).

5) Predicate and Operator Legality
   (CRITICAL):
   - Only reuse names from the official
     predicate list; DO NOT invent new
     construction predicates.
   - Quantities allowed in CDL expressions
     are LIMITED to standard forms:
     VolumeOfCone,
     SurfaceAreaOfCylinder,
     AreaOfCircle, LengthOfLine, etc.
   - Only the following algebraic
     operators are allowed: Value, Add,
     Sub, Mul, Div.
   - Formatting: NO extra spaces inside
     any predicate/operator.

6) Completeness Checks:
   - Ensure every entity used by
     text_cdl/image_cdl exists in
     construction_cdl
   - Ensure the target entity in goal_cdl
     exists in the construction as well
   - Self-check after generation: verify
     all predicates/operators are allowed,
     no extra spaces, and no undeclared
     entities are referenced.

Important: Output Requirements
1. You MUST output a complete JSON object
   with all required fields
2. All CDL fields MUST be arrays of
   strings
3. goal_cdl MUST be a string (e.g.,
   "Value(VolumeOfCone(O,P))")
```

C.4.2. Direct Problem Solving Prompt

In addition to CDL generation, the system also supports **direct problem solving** using GPT models for **testing model accuracy**. This approach bypasses formalization and directly generates answers to geometry problems, providing a baseline for comparison with formalized CDL-based reasoning systems.

The direct solving prompt is simpler than CDL generation and focuses on step-by-step reasoning. The prompt template is:

```
You are an expert in geometry and
mathematics. Please solve the following
geometry problem step by step.

Important Instructions:
1. Carefully analyze the problem text and
the accompanying image
2. Show your reasoning process step by step
3. At the end, provide your final answer in
a clear format
4. **Your final answer should be ONLY a
number or mathematical expression (like
"10", "5.5", "12*pi", "36*pi"), without any
units or text**
5. Put your final answer on a line starting
with "FINAL ANSWER: "

Example format:
FINAL ANSWER: 10

or

FINAL ANSWER: 36*pi

Now, please solve this problem:
```

D. SGRE Supplementary Information

D.1. Theorem Search Tree and Search Process

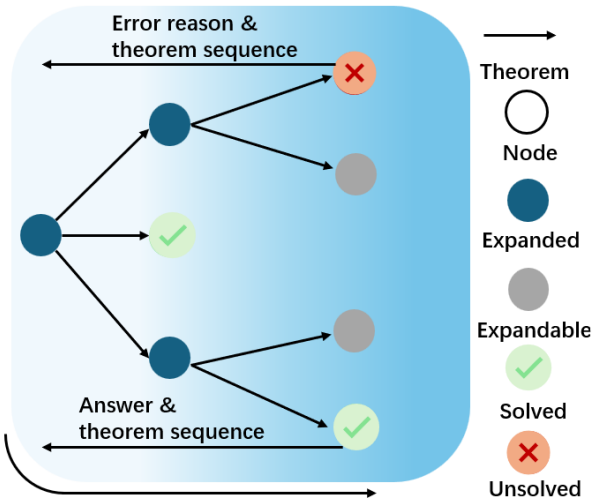

Figure 24. Theorem Search Tree and a inference demonstration is shown in fig. 28

The search process involves constructing a search tree, where nodes represent a set of known conditions, this constitutes a state variable, while arrows denote the applied theorems that supplement the existing 0 condition. As shown in figure 24, starting from the known conditions of the problem, theorems are continuously applied to derive new conditions until the goal is reached. The search tree is traversed using either breadth-first or depth-first search, returning nodes in the "expandable" state. The theorems associated with the current nodes are then repeatedly applied to verify if the problem has been solved. Guided by the known conditions of the current node, the "expand" operation checks the list of applicable theorems and generates new nodes. A simple example of "expand" is as follows: given a cube with a known side length, it is intuitive to expand to other side lengths and various angles. This process requires no additional theorem application and can also validate contradictory conditions resulting from upstream parsing.

D.2. Traditional Search vs. SGRE

Traditional search has various optimization schemes, such as heuristics, but simple pruning of search is not the focus of this paper. The traditional search focuses only on "inferring the target from the known". In contrast, SGRE draws inspiration from SMT (Satisfiability Modulo Theories), which not only determines satisfiability and proactively identifies contradictions but also addresses multi-constraint conflicts (e.g., given a cube with two known distinct side lengths, it returns "unsatisfiable" and provides the cause of the conflict—specifically, the contradictory constraints) and combinations of multiple theories (e.g., hierarchical rapid evaluation of planar and solid properties: the constraints required for problem-solving vary under different theoretical frameworks; see Figure 25). This demonstrates the feasibility of subsequent integration between Hilbert-Geo and large-scale mathematical tools.

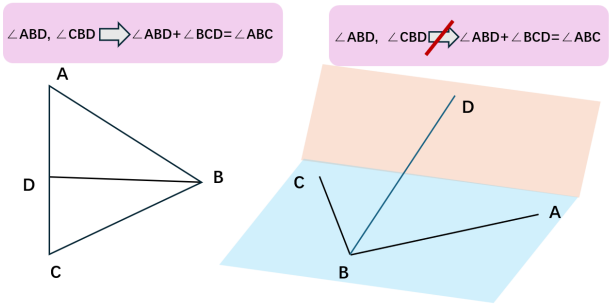

Figure 25. Plane and Solid differ greatly in all aspects, even if the CDL is consistent. SGRE needs to distinguish the Combination of multiple theories

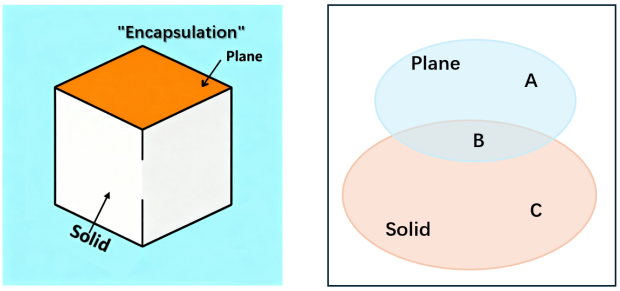

Figure 26. Regarding the encapsulation of the planar theorem, Coplanar(point_seqs) will be treated as a Plane class to apply the theorems and "extend" in set A (See Fig. 27)

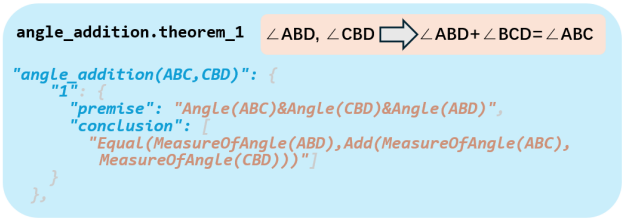

Figure 27. A theorem (as illustrated in Fig. 25) that falls into set A holds true exclusively within the Plane Class

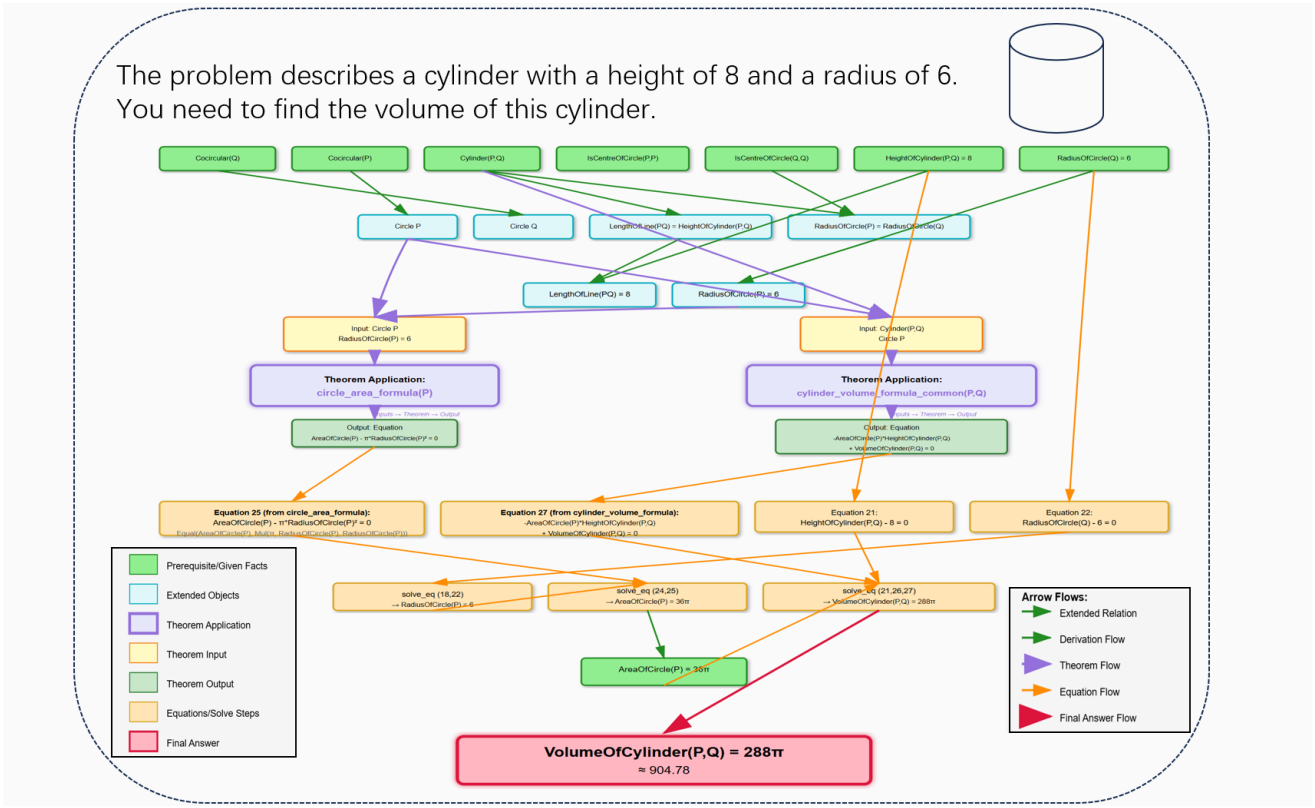

Figure 28. one illustrative example for reasoning based on Hilbert-Geo

E. Experiments

This section supplements some content to the experimental part.

E.1. Mathverse-solid

The findings in Table 9 underscore the inherent challenges of Solid Geometry. GPT-5 emerges as the top-performing model with an overall accuracy of 62.9%, followed by Gemini 2.5 pro at 59.7% and Claude-3.7-Sonnet at 54.7%. Notably, all other models score below 50%, highlighting the significant hurdles that solid geometry reasoning presents even for cutting-edge large language models.

Notable performance disparities persist among state-of-the-art models across fine-grained tasks requiring robust reasoning. GPT-5 leads in SMR with a 67.1% score yet falls short in SSI, managing only 52.9%. Open-source models, exemplified by Deepseek-V3 67B (39.1% in CSS and 34.7% in SSI), consistently underperform relative to closed-source alternatives. Critically, all these models remain far behind human performance, underscoring the current challenges in replicating human-level solid geometry reasoning. Hilbert-Geo, by contrast, delivers a distinct performance profile. Evaluated on Hilbert-Geo with 45 samples, it achieves top-tier results across all key dimensions: 85.9% in CSS, 84.9% in SMR, 80.2% in SSI, 78.3% in MSGF, and an overall accuracy of 84.1%.

Table 9. MLLMs and Hilbert-Geo (Gemini-2.5-pro 45 samples) performances on Mathverse-solid; Four fine grains: Composite Solid Structures (CSS), Spatial Metric Relations (SMR), Solid Shape Identification (SSI), Measurement of Solid Geometric Forms (MSGF); The underline represents the best performance of MLLMs

Model	Overall.Avg	CSS	SMR	SSI	MSGF
Closed-source MLLMs					
GPT-4o	42.4	45.1	36.6	44.2	46.7
GPT-5	<u>62.9</u>	<u>64.9</u>	<u>67.1</u>	52.9	56.1
Claude 3.7 Sonnet	54.7	57.2	55.3	50.6	52.4
Gemini 2.5 Flash	47.5	50.1	47.2	44.5	45.9
Gemini 2.5 Pro	59.7	60.6	61.9	<u>54.1</u>	<u>60.1</u>
Open-source MLLMs					
Llama 3.3 70B	36.0	38.2	36.5	33.8	33.1
Qwen2.5-VL-Instruct-32B	31.0	33.9	29.1	29.8	30.4
Qwen2.5-VL-Instruct-7B	22.9	24.7	20.4	23.5	24.2
Deepseek-V3 67B	35.4	39.1	33.7	34.7	32.2
Human Performances					
Human	92.1	93.0	89.4	97.8	88.5
Hilbert-Geo					
Hilbert-Geo (Ground-Truth) (Ours)	86.3	87.5	87.1	86.9	81.1
Hilbert-Geo (Gemini-2.5-Pro) (Ours)	84.1	85.9	84.9	80.2	78.3

As shown in table 10 and figure 29. Based on Hilbert-Geo, the reasoning success rates of Gemini 2.5 Pro and GPT-5 rise sharply as the number of samples increases, approaching 90% when the sample size reaches 45. In contrast, open-source models such as Llama 3.2 70B exhibit a modest upward trend in performance, which also remains significantly lower than that of closed-source counterparts overall.

Table 10. Performance of Different Models on Hilbert-Geo (45 Samples): Accuracy, Average Time per Solved Problem, and Reasoning Steps; The underline represents the best performance of Hilbert-Geo

Model	Overall.Avg	CSS	SMR	SSI	MSGF
Closed-source MLLMs					
Hilbert-Geo (GPT-4o)	62.1	65.1	59.6	64.2	59.1
Hilbert-Geo (GPT-5)	80.5	84.2	77.1	<u>82.9</u>	77.5
Hilbert-Geo (Claude 3.7)	73.2	76.2	69.8	74.6	72.4
Hilbert-Geo (Gemini 2.5 Flash)	69.1	70.1	67.5	71.5	67.9
Hilbert-Geo (Gemini 2.5 Pro)	<u>84.1</u>	<u>85.9</u>	<u>84.9</u>	80.2	<u>78.3</u>
Open-source MLLMs					
Hilbert-Geo (Llama 3.3 70B)	52.0	54.6	46.5	53.8	56.1
Hilbert-Geo (Qwen2.5-Vl-32B)	38.7	43.9	36.1	35.8	36.4
Hilbert-Geo (Qwen2.5-Vl-7B)	29.4	30.4	26.9	33.7	28.2
Hilbert-Geo (Deepseek-)	46.6	49.1	44.7	44.7	47.2
Formal Language Ground Truth					
Hilbert-Geo (CDL)	86.1	87.5	87.1	86.9	81.1

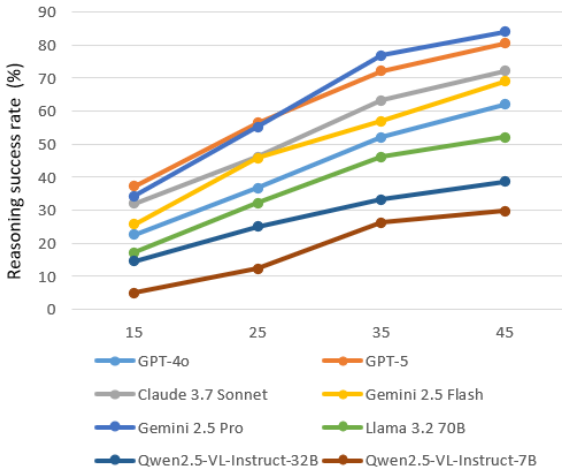

Figure 29. Reasoning performance of MLLMs under different numbers of samples based on Hilbert-Geo

E.2. PlaneFGeo3k

Table 11. MLLMs and Hilbert-Geo (Gemini-2.5-pro 45 samples) performances on PlaneFGeo3k

Model	Accuracy (%)
GPT-4o	47.2
GPT-5	68.1
Claude 3.7 Sonnet	60.0
Gemini 2.5 Flash	55.6
Gemini 2.5 Pro	72.3
Llama 3.2 70B	33.3
Qwen2.5-VL-Instruct-32B	24.8
Qwen2.5-VL-Instruct-7B	17.5
Deepseek-V3 67B	30.4
human	87.9
Ground Truth	84.1
Hilbert-Geo	80.2

As shown in table 11, GPT-5 emerges as the top performer with an overall accuracy of 72.3%, followed by Gemini 2.5 Pro at 68.1% and Claude 3.7 Sonnet at 60.0%. Notable performance disparities persist among leading models. Open-source models like Llama 3.2 70B consistently underperform compared to closed-source ones. Critically, all these models remain far behind human performance, highlighting the current difficulties in replicating human-level plane geometry reasoning. Hilbert-Geo, by contrast, delivers a distinct performance profile. Evaluated on Plane-Geo with 45 samples, it achieves excellent results across all key areas, with an overall accuracy of 80.2%.