PGM: AGENTIC SPATIAL REASONING THROUGH PROGRAMMABLE GEOMETRIC MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Spatial reasoning from images requires measuring object relations and evaluating how they change under hypothetical scene modifications. Connecting these computations requires a scene representation that preserves object identity and keeps geometry consistent across successive operations. We introduce the Programmable Geometric Model (PGM), which couples reconstructed scene geometry with executable operations in a structured Python object. PGM organizes object point clouds, oriented bounding boxes, semantic orientations, and camera poses in a shared coordinate system. Its editable object states allow measurements and rendered views to reflect the same scene configuration after each modification. A Program Agent generates Python programs to query, inspect, and modify PGM, then uses execution results and diagnostics to guide subsequent programs. The scene state persists across executions, allowing dependent computations to build on earlier operations. This formulation supports quantitative queries and hypothetical reasoning through explicit geometric computation, with no task-specific training. Experiments on MMSI-Bench, MindCube-1K, and Omni3D-Bench show improvements across multiple agent backbones. Ablation studies further support the contributions of scene editing and iterative program interaction.

1 INTRODUCTION

Spatial reasoning from images requires determining how objects are arranged and how their relations change with viewpoint or object motion. Figure 2 illustrates a representative example: "After the chair nearest the door rotates 180° in place, is the whiteboard to its front-left?" Answering this question requires identifying the relevant chair, updating its orientation while preserving its position, and re-evaluating the whiteboard's location in the chair's updated reference frame. These steps depend on a consistent account of the same objects and their geometry. Given only a limited set of images, the challenge is to recover a scene representation that supports both measurements of the observed arrangement and computations about hypothetical changes.

Recent work has highlighted the difficulty of spatial reasoning for vision-language models, especially when tasks require metric understanding, explicit 3D geometry, or reasoning across limited observations Chen et al. (2024a); Cheng et al. (2024); Hong et al. (2023); Cai et al. (2025); Wang et al. (2026). To address these limitations, recent methods have expanded the geometric information and computational tools available for spatial reasoning. SandboxVLM represents scenes with abstract 3D bounding boxes, while pySpatial generates programs for 3D reconstruction and viewpoint exploration Liu et al. (2025); Luo et al. (2026). VADAR introduces dynamically generated Python APIs, and SpatialClaw supports iterative spatial computation through a persistent Python interpreter Marsili et al. (2025); Cho et al. (2026). These approaches establish useful foundations for geometric reasoning and adaptive program execution. Our focus is the scene representation on which programs operate: how should object geometry, reference frames, and editable states be organized so that successive queries and modifications remain consistent? For example, rotating an object should update the geometry used by later measurements and renderings, while preserving its identity.

Our central insight is that scene geometry and the operations that act on it should form a single programmable model. Many spatial questions can be resolved by measuring distances, evaluating directions in a specified frame, or recomputing relations after a geometric transformation. These computations require explicit object geometry and state, but do not inherently require photorealistic scene

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

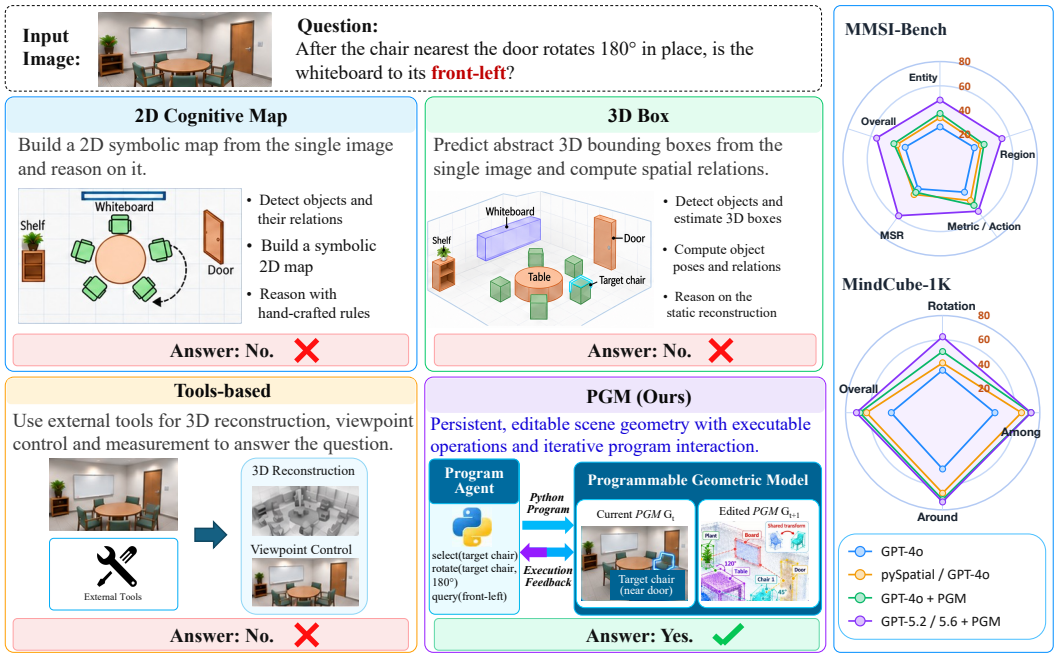


Figure 1: **PGM versus existing spatial reasoning approaches.** PGM enables programmatic interaction with editable scene geometry and achieves the best overall results among compared methods on MMSI-Bench and MindCube-1K.

synthesis. An appropriate abstraction should retain observed surfaces, object extents, orientations, and camera poses, and define how they change under an operation. Queries can then return geometric quantities directly, while abstract 3D renderings provide an additional means of inspecting the scene. This design connects visual reconstruction to a sequence of explicit, inspectable computations.

We introduce the *Programmable Geometric Model (PGM)*, which represents a visual scene as a structured Python object with editable entities and callable operations. Pretrained perception tools recover question-relevant object point clouds, oriented bounding boxes, semantic orientations, and camera poses in a common world coordinate system. PGM retains both surface samples and bounding boxes, supporting measurements of observed geometry alongside comparisons of object extent. Each object placement controls the transformation of its points and box, so an edit is reflected consistently in subsequent queries and views. Camera poses and object orientations provide explicit reference frames for observer-relative and object-relative questions. Together, these components form an environment in which programs can measure the scene, alter its configuration, and evaluate the resulting spatial relations.

A *Program Agent* interacts with PGM by generating Python programs, as illustrated in Figure 2. A persistent Python interpreter executes each program and returns computed values, requested 3D renderings, or execution diagnostics. The agent uses this feedback to decide which relation to evaluate next, whether to inspect an edited configuration, or how to correct a failed operation. Programs can also compose existing operations into reusable functions for the current question. Scene updates and registered functions remain available across executions, allowing later programs to continue from the state established by earlier ones. For the chair example, the agent can identify the chair nearest the door, rotate it 180° in place, and evaluate whether the whiteboard lies to its front-left in the updated chair frame. The resulting programs and outputs expose the geometric computations underlying the answer, while all perception and language models remain frozen.

Our contributions are summarized as follows:

- We introduce PGM, a programmable scene representation that combines observed object geometry, camera poses, and editable states with a shared interface for measurement, modification, and visual inspection.

- We develop a spatial reasoning framework in which a Program Agent uses Python execution and feedback to perform dependent computations on PGM, carrying scene updates across successive reasoning steps.
- Our framework achieves state-of-the-art performance on multiple spatial reasoning benchmarks, demonstrating the effectiveness of reasoning through programmable geometric models.

2 Related Work

Spatial Reasoning in VLMs. Multimodal large language models (MLLMs) have shown strong general visual-language capabilities, yet they still struggle with geometric estimation, metric understanding, and viewpoint-dependent spatial relations Yu et al. (2023); Kamath et al. (2023); Wang et al. (2024a); Tong et al. (2024). Existing work improves spatial competence in several ways. SpatialVLM Chen et al. (2024a) and SpatialRGPT Cheng et al. (2024) introduce large-scale spatial supervision derived from 3D scenes, while 3D-LLM Hong et al. (2023), MiniGPT-3D Tang et al. (2024), ShapeLLM Qi et al. (2024), and SpatialBot Cai et al. (2025) expose models to point clouds, depth, or other 3D-aware representations. At inference time, Spatial Mental Modeling Wang et al. (2026) constructs cognitive maps from limited observations, whereas MindJourney Yang et al. (2026b) leverages generative world-model priors to enrich spatial inference. These methods provide stronger geometric evidence, but usually encode it in model features, auxiliary structures, or query-specific context. PGM instead maintains a persistent scene state with object entities, geometric representations, camera poses, reference frames, and editable spatial states, allowing measurements and scene edits to operate on the same evolving representation throughout reasoning.

Tool-based Agentic Methods. Another direction delegates perceptual and geometric subproblems to external modules. Visual ChatGPT Wu et al. (2023), MM-ReAct Yang et al. (2023), and HuggingGPT Shen et al. (2023) established the general paradigm of coordinating specialist models for multimodal tasks. Spatial agents adapt this idea to geometric reasoning. RieMind Ropero et al. (2026) represents scene structure explicitly and applies geometric operators to support spatial inference. SpaceTools Chen et al. (2026a) coordinates a predefined set of perception tools. GCA Chen et al. (2026b) derives geometric constraints from the query to guide tool execution, while Think3D Zhang et al. (2026) selects viewpoints and reasons over renderings of reconstructed geometry. These approaches mainly organize how geometric evidence is acquired or processed. PGM instead uses a shared programmable scene representation as the organizing layer: perception outputs are incorporated into object and camera entities in a common world coordinate system, and subsequent measurements, frame-dependent queries, and hypothetical edits are evaluated with respect to the current scene state.

Code Agents for Spatial Reasoning. Executable programs provide greater flexibility than fixed tool sequences by allowing task-specific computations to be composed at runtime. VisProg Gupta & Kembhavi (2023) and ViperGPT Surís et al. (2023) generate programs that combine visual modules through predefined APIs, while CodeAct Wang et al. (2024b) studies executable code more generally as an action interface for language-model agents. For spatial reasoning, VADAR Marsili et al. (2025) constructs query-specific functionality for single-view tasks, and pySpatial Luo et al. (2026) generates 3D visual programs over reconstructed geometry, camera information, and synthesized viewpoints. SpatialClaw Cho et al. (2026) further introduces iterative execution with a persistent Python workspace, enabling later computations to reuse and revise intermediate results. PGM makes persistence explicit in the scene representation itself: programs operate on object entities and their editable spatial states, while hypothetical edits update the scene configuration and subsequent geometric measurements and renderings are evaluated from the corresponding updated state.

3 Motivation

Spatial reasoning often requires a sequence of operations on the same scene. Consider asking whether a lamp remains to the left of a chair after the chair turns toward a table. The answer depends on identifying the chair, changing its orientation, and evaluating the position of the lamp in the updated chair frame. The chair identity must remain unchanged, while its geometry and reference frame must

reflect the rotation. This example motivates a scene representation that defines both the geometric entities available to a program and how their states change under an operation.

Existing coding agents provide complementary foundations for such reasoning. pySpatial composes reconstruction, camera transformations, and novel-view rendering through Python programs Luo et al. (2026). VADAR generates reusable Python functions to address subproblems beyond a fixed API Marsili et al. (2025). SpatialClaw maintains a persistent Python workspace and adapts successive computations to numerical results, visual observations, and execution feedback Cho et al. (2026). These methods establish the value of executable tools, extensible computation, and iterative analysis. Our focus is the geometric scene model exposed within this execution process. A Python workspace can retain points, boxes, and poses, but their storage alone does not specify which quantities belong to the same object or how an edit should propagate across them. Encoding these relationships in the scene model gives successive programs a common basis for interpreting and modifying geometry.

Our insight is to make object geometry and the operations that change it part of a single programmable representation. PGM associates each object identity and local geometry with an editable spatial placement, while camera poses define the frames used for observer-relative queries. An object transformation updates the placement shared by its point cloud and bounding box, so subsequent measurements and renderings use the same modified configuration. Relations are evaluated from the current geometry when queried. This design lets the Program Agent compose measurements and interventions without separately implementing how each edit updates the object geometric representations. The agent can still define new functions over PGM and use execution feedback to decide which computation to perform next. PGM therefore combines a consistent geometric basis for scene operations with the flexibility of question-specific programming.

4 METHOD

4.1 OVERVIEW

Given a limited set of images $X = \{x_n\}_{n=1}^{N}$ that partially observe a scene and a spatial question q, our goal is to infer an answer y. We introduce a **Programmable Geometric Model (PGM)**, denoted by $\mathbf{G}$, that represents the scene as a structured Python object. PGM combines object geometry, camera poses, and callable operations within an editable environment. This design allows spatial reasoning to proceed through measurements and interventions on a shared scene representation. Figure 2 illustrates the framework. We first recover question-relevant geometry with pretrained perception tools and organize it into the initial PGM $\mathbf{G}_0$ (Section 4.2). A **Program Agent A** then writes Python programs to query, inspect, and modify PGM. A persistent Python interpreter executes these programs and returns feedback, which guides subsequent programs until the agent produces an answer (Section 4.3). Scene updates remain available across executions, allowing later computations to build on earlier operations. All perception and language models remain frozen.

4.2 FROM IMAGES TO PGM

We construct $\mathbf{G}_0$ by converting the geometry recovered from X into object and camera entities. These entities share a world coordinate system and expose operations for spatial computation and editing.

Geometric information extraction. Recent spatial reasoning systems increasingly leverage explicit 3D geometry, object-level representations, and executable geometric computation to support metric and viewpoint-dependent reasoning Liu et al. (2025); Luo et al. (2026); Cho et al. (2026). VGGT Wang et al. (2025a) estimates depth, camera intrinsics, and camera poses, which we use to reconstruct a shared point cloud. SAM3 Carion et al. (2026) provides instance masks for question-relevant objects. We associate reconstructed points with these masks by image projection, then filter the resulting object point sets using reconstruction confidence and robust geometric statistics, following the robust point-filtering strategy used in SpatialClaw Cho et al. (2026). For each object, we fit an upright oriented bounding box. Let $\mathbf{H}_i$ be the convex hull of its point cloud projected onto the horizontal plane of the scene frame. The box footprint is the minimum-area enclosing rectangle:

$$\mathbf{R}_i^{\star} = \arg\min_{\mathbf{R} \supseteq \mathbf{H}_i} \mathrm{Area}(\mathbf{R}),$$

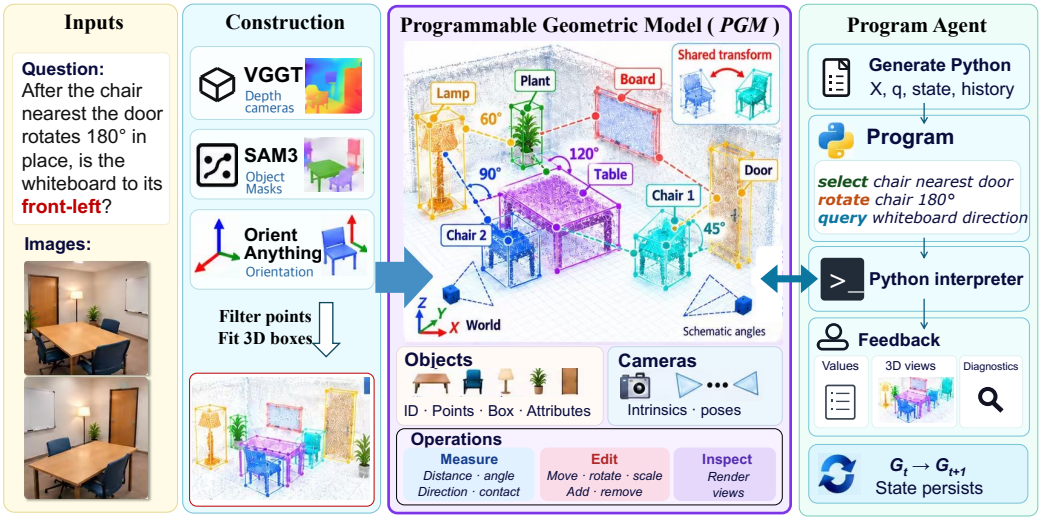


Figure 2: **Spatial reasoning with PGM.** Perception tools recover object geometry and camera poses to initialize a programmable scene. The Program Agent interacts with PGM through Python programs that measure, inspect, or modify the scene. The interpreter returns execution feedback, while PGM retains scene updates for subsequent reasoning steps.

where $\mathbf{R}$ ranges over oriented rectangles. Combining this footprint with robust vertical bounds yields the 3D box. We separately estimate semantic object orientation from individual object crops using Orient Anything Wang et al. (2024c). After converting its predictions to our camera convention, we map the orientations into the world frame using the corresponding camera poses. The fitted box describes geometric extent, while semantic orientation supports object-relative queries. Appendix B details the coordinate conventions, point association and filtering, box estimation, and semantic-orientation mapping.

Programmable entity construction. Each object is stored as a Python entity:

$$o_i = (\mathrm{id}_i, P_i, B_i, T_i, a_i), \qquad T_i(\mathbf{u}) = s_i R_i \mathbf{u} + \mathbf{t}_i. \tag{1}$$

Here id_i identifies the object, P_i contains its observed surface points, and B_i is its oriented bounding box. Both P_i and B_i are stored in an object-local frame, initialized from the estimated semantic orientation. The transformation T_i maps local coordinates to the shared world frame through rotation R_i, translation $\mathbf{t}_i$, and uniform scale s_i, initialized to one. The box retains its independently fitted orientation within this local frame. The attributes a_i store the object semantic label, observed appearance, and observation metadata. Each input camera is represented by $c_n = (K_n, R_n^{\mathrm{cam}}, \mathbf{t}_n^{\mathrm{cam}})$, where K_n contains its intrinsics and $(R_n^{\mathrm{cam}}, \mathbf{t}_n^{\mathrm{cam}})$ maps camera coordinates to the same world frame. The initial model is

$$\mathbf{G}_0 = (\mathbf{V}_0, \mathbf{C}, \mathbf{O}_0), \qquad \mathbf{V}_0 = \{o_i\}_{i=1}^{M}, \quad \mathbf{C} = \{c_n\}_{n=1}^{N}, \tag{2}$$

where M is the number of reconstructed objects and $\mathbf{O}_0$ contains the initial callable operations. Spatial computations use the world-space geometry $P_i^w = T_i(P_i)$ and $B_i^w = T_i(B_i)$, placing objects and camera viewpoints in a common reference.

Executable geometric interface. PGM provides operations for geometric measurement, scene editing, and visual inspection. Programs can select and count objects, measure distances and angles, compare dimensions, and evaluate directions relative to a camera or another object. The oriented boxes also support geometric tests of overlap and potential support contact. For example, a support-contact test combines footprint overlap with the vertical gap between two boxes. Scene edits translate, rotate, scale, add, or remove objects. A transformation U_i updates an object's placement as $T_i' = U_i \circ T_i$, applying the same edit to its point cloud and bounding box. Rendering operations produce abstract 3D views of the resulting configuration from specified viewpoints.

4.3 Reasoning through PGM Programs

The Program Agent $\mathbf{A}$ interacts with PGM through a loop of program generation, execution, and feedback. At step t, the interpreter exposes the current model $\mathbf{G}_t$ as a Python object. The agent

Table 1: **Results on MMSI-Bench.** All scores are percentages (↑). Bold and underlined values indicate the best and second-best distinct scores among the listed methods, with ties marked equally.

Method / backbone	Entity	Region	Metric/Action	MSR	Overall ↑
Baseline Models					
GPT-4o	26.2	29.6	34.1	30.8	30.3
GPT-4.1	30.8	30.5	32.8	28.8	30.9
GPT-4.5	38.5	<u>47.3</u>	38.8	36.4	40.3
GPT-5	42.3	42.1	41.6	42.0	41.9
GPT-5.6 Sol	42.8	40.5	<u>49.3</u>	48.5	<u>45.2</u>
Gemini-2.5-Pro	39.0	39.3	34.8	34.3	36.9
Gemini-3-Pro	<u>46.4</u>	46.6	48.2	37.9	<u>45.2</u>
LLaVA-OneVision-72B	35.0	36.5	23.6	15.7	28.4
Qwen2.5-VL-72B	27.7	34.9	32.5	27.3	30.7
Qwen3-VL-235B	29.1	34.2	30.7	31.8	31.4
Qwen3.5-9B	36.7	42.5	37.0	28.8	36.5
InternVL2.5-78B	25.0	37.8	24.9	27.3	28.5
InternVL3-78B	23.5	32.1	29.3	29.3	28.5
InternVL3.5-14B	24.5	28.4	27.1	31.3	27.6
Spatially specialized models					
SpatialRGPT	26.8	25.3	27.9	25.8	26.5
Spatial-MLLM v1.1	27.9	26.5	22.9	25.3	25.6
AVIC / GPT-4.1	27.8	34.8	40.0	32.3	33.8
MindJourney / GPT-4.1	24.4	35.6	36.7	<u>50.0</u>	34.0
pySpatial / GPT-4o	33.7	35.7	42.6	36.4	37.3
Ours					
Qwen3-VL-235B + PGM	37.1	38.7	45.7	42.4	42.1
GPT-4o + PGM	37.3	38.1	47.4	34.2	40.0
GPT-5.6 Sol + PGM	**48.3**	**53.5**	**53.6**	**58.1**	**54.9**

receives scene observations z_t, comprising available entity attributes, geometric summaries, and requested 3D renderings, together with the execution history H_t. It generates a Python program p_t and receives feedback r_t from its execution:

$$p_t = \mathbf{A}(X, q, z_t, H_t), \quad (\mathbf{G}_{t+1}, r_t) = \text{Python}(p_t, \mathbf{G}_t), \quad H_{t+1} = H_t \,\|\, [(p_t, r_t)], \tag{3}$$

where H_0 is empty and $\|$ denotes sequence concatenation. The index t denotes program executions.

Execution updates the model according to the operations performed by p_t. A read-only query leaves $\mathbf{G}_t$ unchanged, while an edit changes the affected entities. Later programs access this updated state and can continue measuring or modifying it. The agent may also define reusable Python functions that compose existing operations Marsili et al. (2025). Functions registered with PGM become part of its callable interface and remain available in subsequent executions.

The feedback r_t contains computed values, requested renderings, or execution diagnostics. The agent uses it to identify which spatial conditions in q have been evaluated and which require further computation. It then generates the next program to resolve remaining relations, inspect an edited configuration, or correct an execution error. The loop ends when the agent produces an answer y or reaches the interaction budget. The retained programs and outputs make the geometric computations underlying the answer available for inspection.

5 Experiments

5.1 Benchmarks and Implementation Details

We evaluate on MMSI-Bench Yang et al. (2026a), MindCube-1K Wang et al. (2026), and Omni3D-Bench Marsili et al. (2025). MMSI-Bench contains 1,000 multiple-choice questions that require spatial reasoning across images. MindCube-1K contains 1,050 questions covering Rotation, Among, and Around settings. Omni3D-Bench comprises 500 questions about object locations, dimensions, and spatial relations in real-world images. We compare against general-purpose models from the GPT, Gemini, Claude, and Llama families, together with LLaVA-OneVision and the Qwen and InternVL

Table 2: **Results on MindCube-1K.** All scores are percentages (↑). Bold and underlined values indicate the best and second-best distinct scores among the listed methods, including ties.

Method / backbone	Rotation	Among	Around	Overall ↑
Baseline Models				
GPT-4o	35.0	43.0	46.4	42.3
GPT-4.1	35.5	45.3	50.4	44.7
GPT-5.2	48.5	45.2	62.4	49.9
Claude Opus 4.6	29.0	50.7	58.8	48.5
InternVL3.5-4B	34.0	35.3	41.6	36.6
Qwen3-VL-235B	**87.0**	35.0	47.3	47.3
Spatially specialized models				
ViperGPT	20.5	41.0	40.4	37.0
VADAR	33.5	40.7	46.8	40.8
pySpatial / GPT-4o	41.0	65.0	66.3	62.7
pySpatial / GPT-4.1	41.8	64.9	<u>72.7</u>	62.4
GCA / Qwen3-VL-235B	82.0	59.8	61.8	64.2
Think3D / GPT-4.1	63.0	51.2	57.5	55.3
Think3D / Gemini-2.5-Pro	**87.0**	49.7	62.8	60.0
Think3D / Qwen3-VL-4B	32.0	30.3	20.5	27.3
Ours				
Qwen3-VL-235B + PGM	<u>83.8</u>	62.5	58.3	65.6
GPT-4o + PGM	50.4	<u>72.6</u>	71.6	<u>68.1</u>
GPT-5.2 + PGM	62.6	**72.8**	**73.5**	**71.0**

variants listed in the tables OpenAI (2026); Google DeepMind (2026); Anthropic (2026); Meta AI (2024); Li et al. (2024); Bai et al. (2025b;a); Qwen Team (2026a;b); Chen et al. (2024b); Zhu et al. (2025); Wang et al. (2025b). Spatially specialized baselines include SpatialRGPT, Spatial-MLLM, and SpaceMantis Cheng et al. (2024); Wu et al. (2026); remyx.ai (2024). We also compare with visual programming methods ViperGPT, VisProg, and VADAR Surís et al. (2023); Gupta & Kembhavi (2023); Marsili et al. (2025), and spatial reasoning frameworks AVIC, MindJourney, pySpatial, GCA, Think3D, and SpatialClaw Yu et al. (2026); Yang et al. (2026b); Luo et al. (2026); Chen et al. (2026b); Zhang et al. (2026); Cho et al. (2026). For PGM, we report results with multiple Program Agent backbones and compare them with their corresponding base models.

We use VGGT Wang et al. (2025a) for 3D reconstruction, SAM3 Carion et al. (2026) for instance segmentation, and Orient Anything Wang et al. (2024c) for object orientation estimation. For each object, we filter reconstructed points using a confidence threshold of 0.1, retaining the unfiltered set if fewer than 10 points remain. We remove horizontal outliers using a threshold of three median absolute deviations along each horizontal axis. Generated Python programs access PGM through its callable interface and execute with a time limit of 300 seconds.

5.2 Comparison with State-of-the-Art Methods

MMSI-Bench. Table 1 shows that PGM improves overall performance with all three evaluated backbones. The gains over the corresponding base models are 10.7 percentage points for Qwen3-VL-235B and 9.7 points for both GPT-4o and GPT-5.6 Sol. GPT-5.6 Sol with PGM reaches 54.9, the highest overall score among the listed methods, and leads all four diagnostic groups. Its multi-step reasoning score reaches 58.1, exceeding the strongest listed alternative by 8.1 points. With GPT-4o, PGM also improves over pySpatial from 37.3 to 40.0 overall. These results show that the benefit of PGM extends across the evaluated backbones on multi-image spatial questions.

MindCube-1K. PGM achieves overall accuracies of 65.6, 68.1, and 71.0 with Qwen3-VL-235B, GPT-4o, and GPT-5.2, respectively (Table 2). Relative to their base models, these results correspond to gains of 18.3, 25.8, and 21.1 percentage points. GPT-5.2 with PGM obtains the highest overall score among the listed methods and leads the Among and Around categories. With GPT-4o, PGM exceeds pySpatial by 5.4 points overall. The gains vary across categories: Qwen3-VL-235B improves from 35.0 to 62.5 on Among, while its Rotation accuracy decreases from 87.0 to 83.8.

Table 3: **Results on Omni3D-Bench.** All scores are percentages (↑). Bold and underlined values indicate the best and second-best distinct scores among the listed methods.

Method / backbone	Count	Other	Yes/no	MC	Overall ↑
Baseline Models					
GPT-4o	<u>28.1</u>	35.5	66.7	57.2	42.9
Claude 3.5 Sonnet	22.4	20.6	62.2	50.6	32.2
Gemini-1.5-Pro	25.2	28.1	46.2	37.6	32.0
Llama-3.2	24.3	19.3	47.5	27.4	25.6
Qwen3.6-27B	–	–	–	–	55.9
Spatially specialized models					
SpaceMantis	20.0	21.7	50.6	48.2	30.3
ViperGPT	20.0	15.4	56.0	42.4	26.7
VisProg	2.9	0.9	54.7	25.9	13.5
VADAR / GPT-4o	21.7	35.5	56.0	57.6	40.4
VADAR / Gemma4-31B	–	–	–	–	44.6
pySpatial / GPT-4o	22.9	<u>38.6</u>	72.0	54.7	44.2
pySpatial / Gemma4-31B	–	–	–	–	50.6
SpatialClaw / Qwen3.6-27B	–	–	–	–	<u>57.7</u>
Ours					
GPT-4o + PGM	24.3	37.6	<u>76.0</u>	<u>66.3</u>	52.3
Qwen3.6-27B + PGM	**34.2**	**51.2**	**80.3**	**78.0**	**61.7**

Table 4: **PGM operation ablations.** Geometric inputs are fixed across configurations.

Function	G	R	T	A	Full
Geometric queries	✗	✓	✓	✓	✓
Scene rendering	✗	✓	✓	✓	✓
Object transforms	✗	✗	✗	✓	✓
Object addition/removal	✗	✗	✓	✗	✓
Omni3D Overall ↑	-	46.7	49.2	50.8	52.3

G: geometry records; R: read-only PGM; T: without transforms; A: without addition/removal.

Table 5: **Program interaction ablations.** All configurations use the full PGM.

Function	S	V	D	H	Full
Iterative program	✗	✓	✓	✓	✓
Rendered feedback	✓	✗	✓	✓	✓
Diagnostic details	✓	✓	✗	✓	✓
Earlier history	✗	✓	✓	✗	✓
Omni3D Overall ↑	47.1	50.6	51.3	50.9	52.3

S: single-pass program; V: without rendered feedback; D: without diagnostic details; H: without history.

Omni3D-Bench. As shown in Table 3, Qwen3.6-27B with PGM reaches an overall score of 61.7, improving on the base model by 5.8 points and SpatialClaw with the same backbone by 4.0 points. This configuration also achieves the highest available score in each answer category. With GPT-4o, PGM reaches 52.3 overall, exceeding the base model by 9.4 points and pySpatial by 8.1 points. The comparison with pySpatial shows gains in counting, yes/no, and multiple-choice questions, while the continuous numerical score is slightly lower.

5.3 Ablation Studies

We evaluate the contributions of PGM operations and program interaction on Omni3D using GPT-4o as the Program Agent. All configurations share the same perception outputs and evaluation protocol. **PGM operations:** Table 4 shows that editable scene states improve reasoning beyond geometric queries and rendering. Full PGM achieves an overall score of 52.3, compared with 46.7 for the read-only configuration. Disabling object transformations reduces the score to 49.2, while disabling object addition and removal reduces it to 50.8. The larger drop from removing transformations indicates that modifying object placement, orientation, and scale contributes more strongly under this evaluation. Together, these results support treating the reconstructed scene as an editable environment in which programs can evaluate spatial changes. **Program interaction:** Table 5 shows that execution-guided iteration improves reasoning with the same PGM representation. The full interaction process outperforms a single-pass program by 5.2 points, increasing the score from 47.1 to 52.3. Removing rendered feedback lowers the score to 50.6, indicating that abstract scene views complement numerical execution results. Removing earlier interaction history or detailed

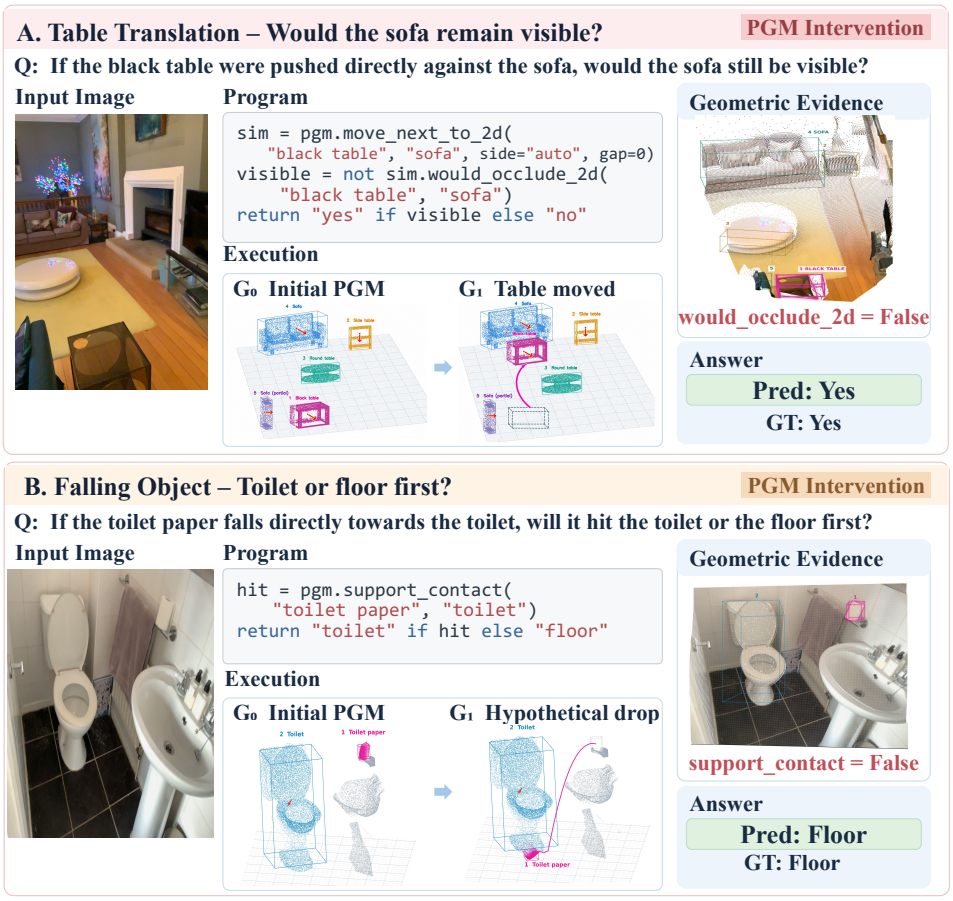


Figure 3: **Qualitative results of PGM.** Generated Python program sketches perform hypothetical scene edits and evaluate the resulting geometric relations, linking each answer to explicit scene operations and execution evidence.

diagnostics reduces the score to 50.9 and 51.3, respectively. These results support a reasoning process in which the agent retains prior findings and uses execution feedback to guide subsequent programs.

5.4 Qualitative Results

Figure 3 shows how the Program Agent uses PGM for hypothetical spatial reasoning. Each example presents the input image, an illustrative Python program, the resulting PGM state, and execution evidence. In the first example, the black table is moved against the sofa, after which the program evaluates whether the sofa remains visible. In the second, a hypothetical drop of the toilet paper is evaluated using geometric contact evidence to determine whether it encounters the toilet or the floor. These examples illustrate how editable scene states and executable geometric operations allow answers to be traced to explicit scene modifications and their resulting spatial evidence.

6 Conclusion

We presented PGM, a programmable geometric model that makes reconstructed scenes directly accessible to spatial reasoning programs. By combining object geometry, editable states, and callable operations, PGM provides a shared scene representation for measurement, intervention, and visual inspection. A Program Agent uses Python execution feedback to carry out successive computations while retaining the scene changes established by earlier programs. Experiments across three spatial reasoning benchmarks demonstrate the effectiveness of this approach, while ablations support the roles of scene editing and iterative interaction. These findings support organizing scene geometry and its executable operations together as a foundation for spatial reasoning.

AI USE STATEMENT

In this work, we used generative AI tools solely for language polishing, including improving grammar, clarity, and readability of the manuscript. We did not use generative AI tools for any other purpose. All AI-assisted edits were carefully reviewed by the authors to ensure that they preserved the intended meaning and did not introduce substantive changes or unsupported claims. We take full responsibility for the final content of this work.

REFERENCES

Anthropic. Claude model system cards. https://www.anthropic.com/system-cards, 2026. Anthropic model system cards.

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

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zeren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025b. URL https://arxiv.org/abs/2502.13923.

Wenxiao Cai, Iaroslav Ponomarenko, Jianhao Yuan, Xiaoqi Li, Wankou Yang, Hao Dong, and Bo Zhao. Spatialbot: Precise spatial understanding with vision language models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9490–9498. IEEE, 2025.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, Joseph Greer, Meng Wang, Peize Sun, Roman Rädle, Triantafyllos Afouras, Effrosyni Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, Sagar Vaze, Francois Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, Ho Kei Cheng, Piotr Dollár, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. SAM 3: Segment anything with concepts, 2026. URL https://arxiv.org/abs/2511.16719.

Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14455–14465, 2024a.

Siyi Chen, Mikaela Angelina Uy, Chan Hee Song, Faisal Ladhak, Adithyavairavan Murali, Qing Qu, Stan Birchfield, Valts Blukis, and Jonathan Tremblay. Spacetools: Tool-augmented spatial reasoning via double interactive rl. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 37109–37120, 2026a.

Zeren Chen, Xiaoya Lu, Zhijie Zheng, Pengrui Li, Lehan He, Yijin Zhou, Jing Shao, Bohan Zhuang, and Lu Sheng. Geometrically-constrained agent for spatial reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 38689–38699, 2026b.

Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*, 2024b.

An-Chieh Cheng, Hongxu Yin, Yang Fu, Qiushan Guo, Ruihan Yang, Jan Kautz, Xiaolong Wang, and Sifei Liu. Spatialrgpt: Grounded spatial reasoning in vision-language models. *Advances in Neural Information Processing Systems*, 37:135062–135093, 2024.

Seokju Cho, Ryo Hachiuma, Abhishek Badki, Hang Su, Byung-Kwan Lee, Chan Hee Song, Sifei Liu, Subhashree Radhakrishnan, Seungryong Kim, Yu-Chiang Frank Wang, and Min-Hung Chen. SpatialClaw: Rethinking action interface for agentic spatial reasoning, 2026. URL https://arxiv.org/abs/2606.13673.

Google DeepMind. Gemini model cards. https://deepmind.google/models/model-cards/, 2026. Google DeepMind model cards.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14953–14962. IEEE, 2023.

Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3d-llm: Injecting the 3d world into large language models. *Advances in Neural Information Processing Systems*, 36:20482–20494, 2023.

Amita Kamath, Jack Hessel, and Kai-Wei Chang. What's "up" with vision-language models? investigating their struggle with spatial reasoning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9161–9175, 2023.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, et al. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*, 2024.

Yifan Liu, Fangneng Zhan, Kaichen Zhou, Yilun Du, Paul Pu Liang, and Hanspeter Pfister. Abstract 3D perception for spatial intelligence in vision-language models, 2025. URL https://arxiv.org/abs/2511.10946.

Zhanpeng Luo, Ce Zhang, Silong Yong, Cunxi Dai, Qianwei Wang, Haoxi Ran, Guanya Shi, Katia Sycara, and Yaqi Xie. pySpatial: Generating 3D visual programs for zero-shot spatial reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=yv15C8ql24.

Damiano Marsili, Rohun Agrawal, Yisong Yue, and Georgia Gkioxari. Visual agentic AI for spatial reasoning with a dynamic API. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19446–19455, 2025. URL https://arxiv.org/abs/2502.06787.

Meta AI. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, September 2024. URL https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

OpenAI. Openai models. https://developers.openai.com/api/docs/models/all, 2026. OpenAI model documentation.

Zekun Qi, Runpei Dong, Shaochen Zhang, Haoran Geng, Chunrui Han, Zheng Ge, Li Yi, and Kaisheng Ma. Shapellm: Universal 3d object understanding for embodied interaction. In *European Conference on Computer Vision*, pp. 214–238. Springer, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026a. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27b dense model, April 2026b. URL https://qwen.ai/blog?id=qwen3.6-27b.

remyx.ai. SpaceMantis. https://huggingface.co/remyxai/SpaceMantis, 2024. Hugging Face model.

Fernando Ropero, Erkin Turkoz, Daniel Matos, Junqing Du, Antonio Ruiz, Yanfeng Zhang, Lu Liu, Mingwei Sun, and Yongliang Wang. Riemind: Geometry-grounded spatial agent for scene understanding. *arXiv preprint arXiv:2603.15386*, 2026.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11854–11864. IEEE, 2023.

Yuan Tang, Xu Han, Xianzhi Li, Qiao Yu, Yixue Hao, Long Hu, and Min Chen. Minigpt-3d: Efficiently aligning 3d point clouds with large language models using 2d priors. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 6617–6626, 2024.

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai C Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, et al. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. *Advances in Neural Information Processing Systems*, 37: 87310–87356, 2024.

Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual geometry grounded transformer, 2025a. URL https://arxiv.org/abs/2503.11651.

Jiayu Wang, Yifei Ming, Zhenmei Shi, Vibhav Vineet, Xin Wang, Yixuan Li, and Neel Joshi. Is a picture worth a thousand words? delving into spatial reasoning for vision language models. *Advances in Neural Information Processing Systems*, 37:75392–75421, 2024a.

Qineng Wang, Baiqiao Yin, Pingyue Zhang, Jianshu Zhang, Kangrui Wang, Zihan Wang, Jieyu Zhang, Keshigeyan Chandrasegaran, Han Liu, Ranjay Krishna, Saining Xie, Jiajun Wu, Li Fei-Fei, and Manling Li. Mindcube: Spatial mental modeling from limited views. In *International Conference on Learning Representations*, 2026.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*, 2025b.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024b.

Zehan Wang, Ziang Zhang, Tianyu Pang, Chao Du, Hengshuang Zhao, and Zhou Zhao. Orient anything: Learning robust object orientation estimation from rendering 3d models. *arXiv preprint arXiv:2412.18605*, 2024c.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

Diankun Wu, Fangfu Liu, Yi-Hsin Hung, and Yueqi Duan. Spatial-mllm: Boosting mllm capabilities in visual-based spatial intelligence. *Advances in neural information processing systems*, 38: 13569–13597, 2026.

Sihan Yang, Runsen Xu, Yiman Xie, Sizhe Yang, Mo Li, Jingli Lin, Chenming Zhu, Xiaochen Chen, Haodong Duan, Xiangyu Yue, Dahua Lin, Tai Wang, and Jiangmiao Pang. Mmsi-bench: A benchmark for multi-image spatial intelligence. In *International Conference on Learning Representations*, 2026a. URL https://github.com/InternRobotics/MMSI-Bench.

Yuncong Yang, Jiageng Liu, Zheyuan Zhang, Siyuan Zhou, Reuben Tan, Jianwei Yang, Yilun Du, and Chuang Gan. Mindjourney: Test-time scaling with world models for spatial reasoning. *Advances in Neural Information Processing Systems*, 38:109855–109885, 2026b.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

Shoubin Yu, Yue Zhang, Zun Wang, Jaehong Yoon, Huaxiu Yao, Mingyu Ding, and Mohit Bansal. When and how much to imagine: Adaptive test-time scaling with world models for visual spatial reasoning. *arXiv preprint arXiv:2602.08236*, 2026.

Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. Mm-vet: Evaluating large multimodal models for integrated capabilities. *arXiv preprint arXiv:2308.02490*, 2023.

Zaibin Zhang, Yuhan Wu, Lianjie Jia, Yifan Wang, Zhongbo Zhang, Yijiang Li, Binghao Ran, Fuxi Zhang, Zhuohan Sun, Yizhuang Peng, et al. Think3d: Thinking with space for spatial reasoning. *arXiv preprint arXiv:2601.13029*, 2026.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.