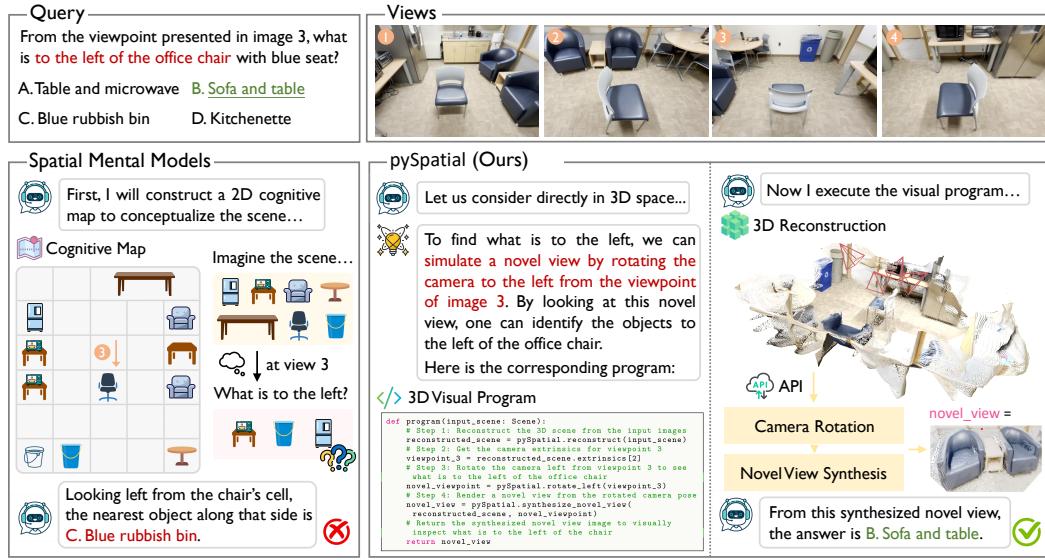


PYSPATIAL: GENERATING 3D VISUAL PROGRAMS FOR ZERO-SHOT SPATIAL REASONING

000
001
002
003
004
005 **Anonymous authors**
006 Paper under double-blind review
007



026 **Figure 1: Comparing our pySpatial with spatial mental models for multi-view spatial reasoning tasks.** Unlike spatial mental models (Yin et al., 2025), which rely on the implicit imagination of
027 MLLMs to construct a 2D cognitive map, we introduce pySpatial, a visual programming frame-
028 work that flexibly composes spatial tools (e.g., 3D reconstruction, camera movements, and novel
029 view synthesis) to enable MLLMs to explicitly reason in 3D space for diverse spatial reasoning tasks.
030

ABSTRACT

033 Multi-modal Large Language Models (MLLMs) have demonstrated strong capa-
034 bilities in general-purpose perception and reasoning, but they still struggle with
035 tasks that require spatial understanding of the 3D world. To address this, we in-
036 troduce pySpatial, a visual programming framework that equips MLLMs with
037 the ability to interface with spatial tools via Python code generation. Given an
038 image sequence and a natural-language query, the model composes function calls
039 to spatial tools including 3D reconstruction, camera-pose recovery, novel-view
040 rendering, etc. These operations convert raw 2D inputs into an explorable 3D
041 scene, enabling MLLMs to reason explicitly over structured spatial representa-
042 tions. Notably, pySpatial requires no gradient-based fine-tuning and operates in
043 a fully zero-shot setting. Experimental evaluations on the challenging MINDCUBE
044 and OMNI3D-BENCH benchmarks demonstrate that our framework pySpatial
045 consistently surpasses strong MLLM baselines; for instance, it outperforms GPT-
046 4.1-mini by 12.94% on MINDCUBE. Furthermore, we conduct real-world in-
047 door navigation experiments where the robot can successfully traverse complex
048 environments using route plans generated by pySpatial, highlighting the prac-
049 tical effectiveness of our approach. Our project website will be available at
050 <https://pySpatial.github.io>.

1 INTRODUCTION

051 Multi-modal Large Language Models (MLLMs) have achieved remarkable success across diverse
052 tasks such as image captioning (Bucciarelli et al., 2024; Wu et al., 2024), referring ground-

ing (Kazemzadeh et al., 2014; Mao et al., 2016), video understanding (Zeng et al., 2025; Fu et al., 2025), and document parsing (Mathew et al., 2021; 2022; Luo et al., 2024). However, this progress has not translated into robust 3D spatial reasoning: recent studies (Wu et al., 2025; Chen et al., 2024a; Chang et al., 2025) reveal that MLLMs still struggle with challenges spanning from basic tasks such as judging relative object positions or estimating depth in a single image (Liu et al., 2023; Cheng et al., 2024) to more complex reasoning over egocentric motion and multi-view relations (Yin et al., 2025; Yang et al., 2025). Such limitations pose a substantial barrier to their reliable deployment in safety-critical applications including robotics, augmented reality, and embodied intelligence, where tasks such as navigation, manipulation, and human–robot interaction depend on precise spatial understanding (Li et al., 2024; Duan et al., 2024; Song et al., 2025; Qiao et al., 2025).

While recent efforts (Chen et al., 2024a; Cheng et al., 2024) have primarily targeted improving spatial understanding from a single image (e.g., “*Is the stool in front of the oven?*”), in this work we focus on the more challenging problem of 3D spatial reasoning, where the environment is only partially observed with limited views and models must reason across perspectives to answer queries such as “*Where should I move from view 1 to view 2?*”—a setting in which state-of-the-art MLLMs perform only slightly above random guess (Yin et al., 2025). Recent studies (Chen et al., 2024a; Ma et al., 2025) suggest that this weakness largely stems from the training data: although MLLMs are pre-trained on internet-scale image-caption pairs, explicit 3D supervision is sparse and costly, making it difficult to learn reliable correspondences between language and 3D spatial structures and thereby constraining models’ ability to reason effectively in 3D space. More recently, Yin et al. (2025) explores the use of data structures such as 2D cognitive maps, where the model encodes object positions in a top-down view to mentally simulate spatial layouts, as shown in Figure 1. However, these approaches still rely on implicit “imagination” mechanisms and offer only limited effectiveness.

These limitations motivate our central research question: *how can we equip MLLMs with explicit reasoning capabilities in 3D space?* A natural first step toward this goal is to obtain an explicit geometric foundation on which such reasoning can take place. Recent advances in feed-forward 3D reconstruction (Wang et al., 2024b; 2025a) makes this feasible by recovering scene geometry directly from sparse 2D views, including camera parameters, depth maps, and scene-level point clouds. Such representations transform limited 2D views into an *explorable* 3D scene, within which models can perform spatial transformations (hereafter referred to as *spatial tools*) such as camera translation, rotation, and viewpoint shifts to enrich visual context and build interactive reasoning chains. For instance, given the query “*what is behind me if I am at view 3?*,” the model could rotate the virtual camera by 180° at the specified viewpoint within the reconstructed scene, thereby uncovering previously occluded regions and grounding its reasoning in geometric evidence.

However, how to enable MLLMs to flexibly compose spatial tools and seamlessly interact with 3D environments in a context-aware manner remains a critical challenge. To address this, inspired by pioneering works on visual programming (Gupta & Kembhavi, 2023; Surís et al., 2023), we introduce `pySpatial`, a framework that employs MLLMs like GPT-4o as Python code generation agents to invoke function calls for tools such as 3D reconstruction, natural language description of movements, and novel view synthesis. As illustrated in Figure 1, `pySpatial` leverages a well-defined API to automatically select and compose the appropriate tools to solve diverse spatial reasoning tasks. Notably, `pySpatial` operates fully in a zero-shot setting and serves as a plug-and-play framework applicable to both open-source and closed-source MLLMs, offering interpretable solutions and reliable responses that make it well-suited for diverse real-world tasks.

We evaluate the effectiveness of our approach on the MINDCUBE and OMNI3D-BENCH benchmarks, where results demonstrate that `pySpatial` consistently outperforms strong MLLM baselines by substantial margins (e.g., achieving a 12.94% improvement over GPT-4.1-mini on MINDCUBE). Qualitative analyses further verify that our approach can generate high-quality executable and interpretable visual programs that can effectively solve complex spatial reasoning tasks in a zero-shot manner. Furthermore, we apply `pySpatial` to real-world indoor navigation, where it successfully enables a quadrupedal robot to traverse complex environments using generated route plans.

Our contributions can be summarized as follows:

- We present `pySpatial`, a novel *zero-shot* framework that enables MLLMs to reason explicitly in 3D space by generating and executing visual programs that leverage various spatial tools in a structured, compositional manner to solve diverse spatial reasoning tasks.

- 108 • We evaluate `pySpatial` on MINDCUBE and OMNI3D-BENCH, where it demonstrates superior
109 performance over strong MLLM baselines. Qualitative analysis validates that `pySpatial` reliably
110 generates executable and interpretable visual programs for diverse spatial reasoning tasks.
- 111 • We further assess the practical effectiveness of `pySpatial` on indoor navigation tasks, showing
112 that it can generate route plans that enable a quadrupedal robot to traverse complex environments,
113 demonstrating strong potentials for practical use cases.

115 2 RELATED WORK

117 **MLLMs for Spatial Reasoning.** Recent MLLMs have demonstrated remarkable performance on
118 multi-modal tasks such as image captioning and visual grounding (Liu et al., 2024a; Alayrac et al.,
119 2022; Bai et al., 2025; Chen et al., 2024b). However, studies have shown that these models exhibit
120 significant limitations in interpreting spatial relations (Yu et al., 2024; Kamath et al., 2023; Wang
121 et al., 2024a; Tong et al., 2024), a critical precursor to a wide range of practical applications, including
122 robotic manipulation (Huang et al., 2022; Shridhar et al., 2023) and embodied navigation (Qiao
123 et al., 2025; Huang et al., 2023). To address this, recent works such as SpatialIVLM (Chen et al.,
124 2024a) and SpatialRGPT (Cheng et al., 2024) typically propose scalable data synthesis and curation
125 pipelines to strengthen *single-view* spatial reasoning capabilities through large-scale pre-training.
126 Despite these advances, more recently, Yin et al. (2025) demonstrates that current MLLMs and such
127 approaches still struggle with geometric understanding and perspective-taking in *multi-view* settings.
128 In this work, we are among the first to tackle this challenge, and we propose a novel zero-shot visual
129 programming framework called `pySpatial` that systematically combines and applies various spatial
130 tools, enabling models to explicitly reason in 3D and solve diverse spatial tasks.

131 **3D Reconstruction.** Classical 3D reconstruction methods, such as Structure-from-Motion (Schon-
132 berger & Frahm, 2016), typically involve multiple stages and often rely on time-consuming
133 optimization pipelines. More recently, feed-forward 3D reconstruction approaches such as
134 DUS3R (Wang et al., 2024b), MAST3R (Leroy et al., 2024), CUT3R (Wang et al., 2025b) and
135 VGGT (Wang et al., 2025a) leverage large-scale 3D pre-training and vision transformers to directly
136 predict pixel-aligned 3D point maps. **These data-driven methods demonstrate strong generalizabil-
137 ity, even in scenarios without overlapping views.** Building on this progress, subsequent works have
138 extended feed-forward 3D reconstruction to applications in neural rendering (Charatan et al., 2024),
SLAM (Maggio et al., 2025), and dynamic reconstruction (Lin et al., 2025).

139 **Modular Visual Reasoning.** To enhance compositional multi-modal understanding, recent ad-
140 vances treats vision specialists (such as GroundingDINO (Liu et al., 2024b) and SAM (Ravi et al.,
141 2025)) as symbolic operators and composes them to solve complex reasoning problems. Repre-
142 sentative works such as Visual ChatGPT (Wu et al., 2023), MM-REACT (Yang et al., 2023), and Hug-
143 gingGPT (Shen et al., 2023) follow this direction by integrating LLMs with predefined toolchains
144 to process multi-modal inputs. Building on this idea, VisProg (Gupta & Kembhavi, 2023) and
145 ViperGPT (Surís et al., 2023) introduce *visual programming* that extends this paradigm by prompt-
146 ing MLLMs to generate executable Python programs that call a set of visual parsers through pre-
147 defined APIs. More recently, VADAR (Marsili et al., 2025) introduces the visual programming
148 paradigm for single-view spatial reasoning tasks with an adaptive API design. In contrast, our
149 `pySpatial` introduces a framework explicitly designed for multi-view spatial reasoning, equipping
150 models with compositional 3D tools to handle diverse and complex spatial scenarios.

151 3 METHOD

152 In this section, we present `pySpatial`, a visual programming framework that enables MLLMs to
153 reason explicitly in 3D space by generating and executing visual programs that orchestrate multiple
154 spatial tools to address diverse spatial reasoning tasks. We also describe the framework design,
155 including the `pySpatial` API signatures and the spatial tools it employs.

156 3.1 PROBLEM FORMULATION

157 We consider a setting where an MLLM \mathcal{M} is provided with an image sequence $\mathcal{I} = \{I_n\}_{n=1}^N$,
158 where each view has resolution $H \times W$ and captures partial observations of a 3D scene, along with
159 a natural-language query q concerning spatial relations between objects or camera movements. The
160 objective is to produce the correct response r^* from the answer space \mathcal{A} that answers the query.

162 As introduced in Section 1, we convert the limited 2D views into an *explorable 3D scene* via feed-
 163 forward reconstruction. This yields consistent depth estimates D , camera intrinsics $K \in \mathbb{R}^{3 \times 3}$, and
 164 extrinsics $\mathbf{G} \in \text{SE}(3)$ for each frame. Together, these quantities define a point cloud \mathcal{P} in world
 165 coordinates, which serves as the geometric basis for downstream reasoning.

166 In addition, we adopt a program synthesis-perspective following Surís et al. (2023). Given an input
 167 of an image sequence and a query (\mathcal{I}, q) , a code agent \mathcal{F} generates a Python program z that invokes
 168 a set of spatial tools through a well-defined API. The program is executed by an interpreter \mathcal{E} to
 169 produce an intermediate output O , which may take the form of text, a single image, or a list of images
 170 depending on the program z . This output provides direct visual evidence to support answering the
 171 query. For instance, when the query asks, “*what is behind me if I am at view 3*,” the program renders
 172 a new view by rotating the camera 180° at the specified viewpoint. Finally, the MLLM \mathcal{M} integrates
 173 both the original visual inputs and the program outputs to generate the final response $r \in \mathcal{A}$.

174 3.2 SPATIAL TOOLS AND API

175 To guide the MLLMs to explicitly reason in 3D space, we introduce various spatial tools such as
 176 3D reconstruction, camera description, and novel view synthesis. We provide the pySpatial API
 177 signatures in Code 1 and the details of each tool are described in the following sections.

178 **3D Reconstruction.** We adopt two feed-
 179 forward reconstructions depending on the
 180 specific task requirement. For metric-scale
 181 scenes, we use CUT3R (Wang et al., 2025b),
 182 which returns depth in real-world units. When
 183 **relative distance in normalized unit space**
 184 suffices, we adopt VGGR (Wang et al., 2025a)
 185 for its generalizability.

186 Formally, each pixel $\mathbf{p}_i \in \mathbb{R}^2$ in a view I_n with
 187 predicted depth $D_n(\mathbf{p}_i)$ is back-projected into
 188 the camera coordinate system using the intrin-
 189 sics K , and then transformed into world coor-
 190 dinates via the estimated pose $\mathbf{G}_n \in \text{SE}(3)$:

$$191 \mathbf{X}_i = \mathbf{G}_n^{-1} \pi^{-1}(\mathbf{p}_i, D_n(\mathbf{p}_i), K^{-1}), \quad (1)$$

192 where π^{-1} denotes the back-projection from
 193 image coordinates to the 3D point in the
 194 camera frame. We get the point cloud \mathcal{P} in the
 195 world space by concatenating \mathbf{X}_i for all pixels
 196 in all frames.

197 **Camera Description.** We translate raw
 198 camera pose matrices into natural language
 199 labels to make egocentric motion interpretable
 200 to the language model. Each pose is represented by an extrinsic matrix $\mathbf{G} = [\mathbf{R} \mid \mathbf{t}] \in \mathbb{R}^{3 \times 4}$,
 201 which maps world points into the camera frame. The corresponding camera center in world
 202 coordinates is $\mathbf{C} = -\mathbf{R}^\top \mathbf{t}$. Given two poses $(\mathbf{R}_1, \mathbf{t}_1)$ and $(\mathbf{R}_2, \mathbf{t}_2)$, the displacement in world
 203 coordinates is $\Delta \mathbf{C}_w = \mathbf{C}_2 - \mathbf{C}_1$. We then express this displacement in the first camera’s frame
 204 as $\Delta \mathbf{C}_1 = \mathbf{R}_1 \Delta \mathbf{C}_w$. Restricting the displacement to the horizontal plane, we compute the yaw
 205 angle $\theta = \text{atan2}(d_x, d_z) \cdot 180/\pi$, where (d_x, d_z) are the x and z components of $\Delta \mathbf{C}_1$. The
 206 angle is discretized into eight canonical motion categories (forward, backward, left, right, and four
 207 diagonals), yielding a compact natural-language description of egocentric movement.

208 **Novel View Synthesis.** To facilitate active exploration of the reconstructed 3D scene, we enable
 209 the agent to render *novel views* from arbitrary camera poses. Given a point cloud \mathcal{P} and a
 210 world-to-camera transformation $\mathbf{G} = [\mathbf{R} \mid \mathbf{t}] \in \mathbb{R}^{3 \times 4}$, we rasterize \mathcal{P} into an RGB image with
 211 respect to \mathbf{G} and the corresponding camera intrinsics K . The agent can then issue high-level
 212 actions such as `rotate_left` and `turn_around`, which are implemented as yaw rotations about
 213 the world y -axis by angle ϕ . The updated camera pose \mathbf{G}' is obtained by applying the rotation to the
 214 camera-to-world transform and inverting back to world-to-camera form. This operation provides
 215 interactive visual feedback that supports explicit spatial reasoning.

```
class pySpatial:
    """pySpatial interface for 3D vision tools."""

    def reconstruct(scene: Scene):
        # 3D reconstruction from scene images.

    def describe_camera_motion(recon: Reconstruction):
        # Describe camera motion from
        # reconstruction results.

    def synthesize_novel_view(recon: Reconstruction,
        new_camera_pose):
        # Generate novel view synthesis from
        # reconstruction results.

    def rotate_right(extrinsic, angle=45):
        # Rotate camera pose to the right, rotate
        45 degree by default

    def rotate_left(extrinsic, angle=45):
        # Rotate camera pose to the left rotate 45
        degree by default

    def move_forward(extrinsic, distance=0.3):
        # Move camera pose forward, a default
        distance is provided

    def move_backward(extrinsic, distance=0.3):
        # Move camera pose backward, a default
        distance is provided

    def turn_around(extrinsic):
        # Turn camera pose around 180 degrees
```

Code 1: pySpatial API signatures.

216 3.3 3D VISUAL PROGRAMMING
217

218 **Program Generation.** Given a query q , the code agent \mathcal{F} synthesizes a Python program $z = \mathcal{F}(q)$
 219 that composes function calls specified in the pySpatial API. By default, we use GPT-4o, a strong
 220 MLLM baseline that has demonstrated effectiveness in code generation, as it has been trained on
 221 Internet-scale Python code data. Note that the agent interacts only with the public interface (e.g.,
 222 `reconstruct`, `rotate_right`, `synthesize_novel_view`) and has no access to internal imple-
 223 mentation details such as model weights, file I/O, or rendering backends. This abstraction separates
 224 high-level reasoning from low-level execution. **We also provide default parameters for public inter-
 225 face regarding rotation and movement, i.e. 45 for rotation, and 0.3 for movement, as specified in
 226 Code 1.** We guide program synthesis using in-context examples, where the prompts include inter-
 227 face documentation and query–code pairs without ground-truth answers. In addition, we leverage
 228 structured outputs to first enable free-form natural language reasoning, followed by the synthesis of
 229 Python code. The generated Python code, or visual program, acts as an explicit intermediate repre-
 230 sentation that encodes a sequence of tool invocations. It is inherently interpretable, as researchers
 231 can readily inspect, debug, or modify the generated program, and composable, enabling seamless
 232 integration with additional tools or downstream reasoning modules. Once constructed, the program
 233 is executed by the interpreter to produce concrete spatial operations.

234 **Program Execution.** At execution time, the synthesized program z is executed by a Python inter-
 235 preter \mathcal{E} over the input image sequence \mathcal{I} , yielding an intermediate output $O = \mathcal{E}(z, \mathcal{I})$. Depending
 236 on the query, the output O may take the form of text, a single image, or a sequence of rendered
 237 views. This intermediate output provides an explicit grounding of the program’s reasoning steps in
 238 observable evidence. In the final stage, a MLLM \mathcal{M} integrates the original image sequence \mathcal{I} , the
 239 program output O , and the natural language query q to generate the final response $r = \mathcal{M}(\mathcal{I}, O, q)$.

240 4 EXPERIMENTS

241 In this section, we assess the effectiveness of pySpatial on MINDCUBE (Yin et al., 2025) and
 242 OMNI3D-BENCH (Marsili et al., 2025), comparing it with existing state-of-the-art approaches.

243 4.1 EXPERIMENTAL SETTINGS

244 **Benchmarks.** We mainly evaluate our framework on the MINDCUBE (Yin et al., 2025), which is
 245 designed to probe the spatial reasoning capabilities of MLLMs under limited views. Specifically,
 246 MINDCUBE contains over 21,000 spatial question–answer pairs grounded in 3,268 multi-view in-
 247 door scenes, spanning three canonical camera motion types: rotation, around, and among. We also
 248 evaluate on MINDCUBE-1k, a subset of MINDCUBE with 1,050 questions, specifically designed for
 249 evaluation purposes. In addition, following prior work (Marsili et al., 2025), we also evaluate our
 250 framework on OMNI3D-BENCH, a single-view spatial reasoning benchmark, to examine whether
 251 our visual programming approach can generalize beyond multi-view settings.

252 **Baselines.** We compare the performance of pySpatial against four categories of existing baselines:
 253 (1) open-weight multi-image MLLMs, such as LLaVA-OneVision-7B (Li et al., 2025) and Qwen2.5-
 254 VL-3B-Instruct (Bai et al., 2025); (2) proprietary MLLMs, including GPT-4o, GPT-4.1-mini, and
 255 Claude-4-Sonnet; (3) specialized spatial models, such as Space-Qwen (Chen et al., 2024a) and
 256 VLM-3R (Fan et al., 2025), and (4) prior visual programming approaches such as ViperGPT (Surís
 257 et al., 2023), VisProg (Gupta & Kembhavi, 2023), and VADAR (Marsili et al., 2025).

258 **Implementation Details.** By default, we follow prior visual programming work (Marsili et al.,
 259 2025) to leverage GPT-4o as the code agent to generate Python programs and produce final responses
 260 to queries. We use VGGT (Wang et al., 2025a) as 3D reconstruction model on the MINDCUBE and
 261 OMNI3D-BENCH benchmarks. For real-world navigation, we use CUT3R (Wang et al., 2025b),
 262 which provides metric-scale reconstructions rather than normalized outputs. For point cloud rasteri-
 263 zation, we use Open3D (Zhou et al., 2018) to render novel views. All experiments are conducted on
 264 a single NVIDIA A6000 Ada GPU. We provide full implementation details of pySpatial, along
 265 with the prompts used, in Appendix B and C. Code will be made publicly available upon acceptance.

266 4.2 QUANTITATIVE RESULTS

267 **Results on MINDCUBE.** We first perform comprehensive evaluations of pySpatial on the chal-
 268 lenging MINDCUBE benchmark to rigorously assess its effectiveness in multi-view spatial reason-

270
271 **Table 1: Performance comparison on the full MINDCUBE (Yin et al., 2025) dataset.** The best
272 results are shown in **bold**, and the second-best are underlined. Note that we implement pySpatial
273 using GPT-4.1-mini as the code agent for this dataset due to budget constraints.

Method	Reference	Overall	Rotation	Among	Around
<i>Baseline</i>					
Random (chance)	-	32.35	36.36	32.29	30.66
Random (frequency)	-	33.02	38.30	32.66	35.79
<i>Open-Weight Multi-Image Models</i>					
LLaVA-OneVision-7B	Li et al. (2025)	47.43	36.45	48.42	44.09
LLaVA-Video-Qwen-7B	Zhang et al. (2025)	41.96	35.71	43.55	30.12
mpPLUG-Owl3-7B-241101	Ye et al. (2025)	44.85	37.84	47.11	26.91
InternVL2.5-8B	Chen et al. (2024b)	18.68	36.45	18.20	13.11
Qwen2.5-VL-7B-Instruct	Bai et al. (2025)	29.26	38.76	29.50	21.35
Qwen2.5-VL-3B-Instruct	Bai et al. (2025)	33.21	37.37	33.26	30.34
DeepSeek-VL2-Small	Lu et al. (2024)	<u>47.62</u>	37.00	<u>50.38</u>	26.91
<i>Proprietary Models</i>					
GPT-4o	OpenAI (2024)	38.81	32.65	40.17	29.16
GPT-4.1-mini	OpenAI (2025)	45.62	37.84	47.22	34.56
Claude-4-Sonnet	Anthropic (2025)	44.75	48.42	44.21	<u>47.62</u>
<i>Specialized Spatial Models</i>					
RoboBrain	Ji et al. (2025)	37.38	35.80	38.28	29.53
SpaceMantis	Chen et al. (2024a)	22.81	37.65	21.26	29.32
Spatial-MLLM	Wu et al. (2025)	32.06	38.39	20.92	32.82
Space-Qwen	Chen et al. (2024a)	33.28	38.02	33.71	26.32
VLM-3R	Fan et al. (2025)	42.09	36.73	44.22	24.45
pySpatial (Ours)	-	58.56	<u>43.20</u>	60.54	48.10

296 **Table 2: Performance comparison on the MINDCUBE-1k (Yin et al., 2025) dataset.** The evaluated
297 mental models (Yin et al., 2025) are based on Qwen2.5-VL-3B-Instruct (Bai et al., 2025).
298 VADAR w/ Recon. denotes that we implement VADAR with our 3D reconstruction module. The
299 best results are highlighted in **bold**, and the second-best are underlined.

Method	Reference	Overall	Rotation	Among	Around
<i>Baseline Models</i>					
Qwen2.5-VL-3B-Instruct	Bai et al. (2025)	37.81	34.00	36.00	45.20
GPT-4o	OpenAI (2024)	<u>42.29</u>	35.00	<u>43.00</u>	46.40
<i>Spatial Mental Models</i>					
Chain-of-Thought		40.48	32.00	36.00	58.00
View Interpolation	Yin et al. (2025)	37.81	35.50	36.50	42.80
Cognitive Map		41.43	<u>37.00</u>	41.67	44.40
<i>Visual Programming Approaches</i>					
ViperGPT	Surís et al. (2023)	36.95	20.50	41.00	40.40
VADAR	Marsili et al. (2025)	40.76	33.50	40.67	46.80
VADAR w/ Recon.	-	35.62	31.00	36.83	36.40
pySpatial (Ours)	-	62.35 ± 1.18	41.83 ± 2.34	64.89 ± 2.60	72.67 ± 3.30

311 ing. Table 1 summarizes the results in comparison with baseline approaches. Overall, pySpatial
312 achieves a clear performance margin over all categories of baselines. Specifically, it reaches an
313 overall accuracy of 58.56%, outperforming the best open-weight model DeepSeek-VL2-Small by
314 10.94%, and surpassing the strongest proprietary model GPT-4.1-mini by 12.94%. On the *Among*
315 category, which requires reasoning over how the central object relates to all surrounding objects,
316 pySpatial achieves 60.54%, substantially outperforming all baselines, none of which exceed
317 50%. Remarkably, pySpatial also outperforms VLM-3R (Fan et al., 2025), which leverages
318 CUT3R (Wang et al., 2025b) as the 3D encoder and is fine-tuned on synthetic spatial reasoning
319 data, by 16.5%, despite operating entirely in a zero-shot setting. These results demonstrate that
320 pySpatial generalizes well across diverse task categories on MINDCUBE. By explicitly decom-
321 posing spatial reasoning into modular tool calls, our approach provides a stronger inductive bias
322 than both open-weight and proprietary MLLMs, including those specialized for spatial reasoning.

323 **Results on MINDCUBE-1k.** Table 2 compares pySpatial against approaches based on implicit
mental modeling (Yin et al., 2025) (e.g., chain-of-thought reasoning, cognitive maps) and prior

324
 325 **Table 3: Performance comparison on OMNI3D-BENCH.** Following VADAR (Marsili et al.,
 326 2025), We report mean relative accuracy (Yang et al., 2025) for the *numeric (other)* and accuracy
 327 for the other category. The best results are shown in **bold**, and the second-best are underlined.

328 Method	329 Reference	330 numeric (ct)	331 numeric (other)	332 y/n	333 multi-choice	334 Total
<i>Baseline Models</i>						
GPT-4o	OpenAI (2024)	28.1	<u>35.5</u>	66.7	<u>57.2</u>	42.9
Claude3.5-Sonnet	Anthropic (2024)	22.4	20.6	<u>62.2</u>	50.6	32.2
Llama-3.2	Meta (2024)	24.3	19.3	47.5	27.4	25.6
Gemini1.5-Pro	Google (2024)	<u>25.2</u>	28.1	46.2	37.6	32.0
SpaceMantis	Chen et al. (2024a)	20.0	21.7	50.6	48.2	30.3
<i>Visual Programming Approaches</i>						
ViperGPT	Surís et al. (2023)	20.0	15.4	56.0	42.4	26.7
VisProg	Gupta & Kembhavi (2023)	2.9	0.9	<u>54.7</u>	25.9	13.5
VADAR	Marsili et al. (2025)	21.7	<u>35.5</u>	56.0	57.6	40.4
pySpatial (Ours)	-	22.9	38.6	72.0	54.7	44.2

335 visual programming agents (e.g., ViperGPT, VADAR) on MINDCUBE-1k. We have the following
 336 key observations: (1) Spatial mental models (Yin et al., 2025), which rely on the implicit imagination
 337 mechanisms of MLLMs for spatial reasoning, yield only limited performance gains, whereas
 338 pySpatial outperforms each of them by roughly 20%; (2) Our pySpatial substantially outperforms
 339 existing visual programming approaches, achieving, for example, a 21.9% improvement over
 340 VADAR. Notably, pySpatial also surpasses VADAR w/ Recon., where we re-implement VADAR
 341 using our 3D reconstruction module. This result demonstrates that even when equipped with 3D
 342 information, VADAR’s adaptive API design remains unreliable and lacks robustness for reasoning
 343 in 3D space. These results validate the superior effectiveness of pySpatial over existing baselines,
 344 demonstrating the advantages of enabling explicit 3D reasoning for multi-view spatial reasoning.

345 **Results on OMNI3D-BENCH.** We further evaluate pySpatial on the recent single-view spatial
 346 reasoning benchmark OMNI3D-BENCH, demonstrating that our framework generalizes effectively
 347 to single-view settings and provides consistent improvements across task categories. Table 3 shows
 348 results on OMNI3D-BENCH, where we follow the evaluation protocol of VADAR (Marsili et al.,
 349 2025): mean relative accuracy (MRA) is reported for the *numeric (other)* subtask, and standard ac-
 350 curacy is used for the remaining categories. Our pySpatial outperforms prior visual programming
 351 approaches, achieving gains of 3.8% over VADAR and 17.5% over ViperGPT, and sets a new over-
 352 all state-of-the-art on OMNI3D-BENCH. Notably, pySpatial also surpasses GPT-4o on the total
 353 score, underscoring that our visual programming framework provides benefits even over advanced
 354 proprietary MLLMs. This result highlights the broad generalizability of pySpatial: even in single-
 355 view settings where geometric cues are less apparent, explicitly invoking 3D functions through the
 356 code agent continues to enhance spatial reasoning.

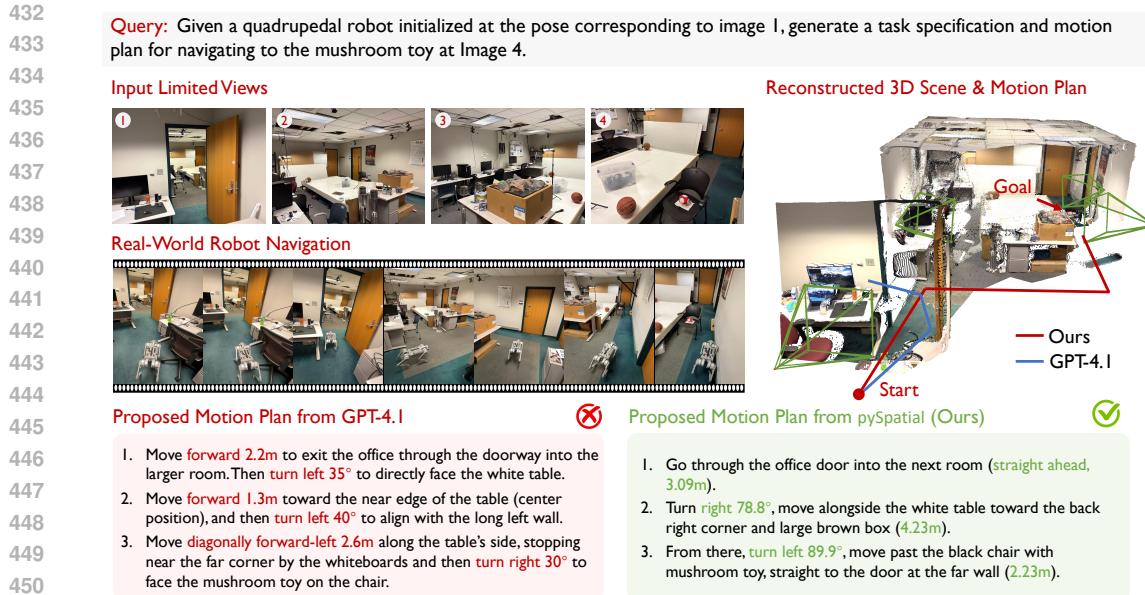
357 4.3 QUALITATIVE RESULTS

358 To further illustrate the capabilities of our pySpatial framework, we conduct qualitative experiments
 359 on representative examples from the MINDCUBE benchmark. As shown in Figure 2, each
 360 query is paired with the generated 3D visual program, the reconstructed 3D scene, the program out-
 361 puts, and the final response produced by pySpatial. These examples highlight how pySpatial
 362 enables MLLMs to reason explicitly within an explorable 3D scene reconstructed from sparse 2D
 363 inputs. By synthesizing executable and interpretable visual programs that perform operations such
 364 as camera translation, rotation, and novel view synthesis, the framework provides interpretable out-
 365 puts that ground the reasoning process in geometric evidence. Across diverse spatial reasoning
 366 tasks, pySpatial produces responses that closely align with ground-truth annotations, highlighting
 367 the effectiveness of our approach. It is worth noting that the generated 3D visual programs
 368 include well-structured comments that capture the reasoning process of pySpatial, thereby pro-
 369 viding transparency and interpretability that researchers can readily verify, debug, or modify.

370 4.4 REAL-WORLD ROBOT NAVIGATION

371 To test the potential of real-world deployment using purely MLLMs, we employ a quadrupedal robot
 372 with a velocity-tracking controller in a 50 m² two-room laboratory. In this setup, the MLLM gen-
 373 erates high-level position commands, which are manually converted into temporal velocity targets that
 374 the controller tracks, enabling the robot to navigate from an initial pose to a target object (a mush-

378	Query: In which direction did I move from the first view to the second view?	Ground Truth: Diagonally forward and left.																																				
379	<pre data-bbox="290 249 930 369"> 1 2 def program(input_scene: Scene): # Step 1: reconstruct the 3D scene from the input images reconstruction3D = pySpatial.reconstruct(input_scene) # Step 2: describe the camera motion between the two views using extrinsics camera_motion_description = pySpatial.describe_camera_motion(reconstruction3D) # Return the description which should match one of the answer choices return camera_motion_description </pre> 	3D Reconstruction: 	Program Output: From image 1 to image 2, the camera move diagonally forward and left.																																			
380	Final Response: The visual clue explicitly states the camera moved from pose 1 to pose 2 diagonally forward and left. Therefore, the direction of the camera movement is diagonally forward and left .																																					
381	Query: From the viewpoint 2, what is on to the right of the black chair?	Ground Truth: Grey Sofa.																																				
382	<pre data-bbox="290 473 930 623"> 1 2 def program(input_scene: Scene): # Step 1: reconstruct the 3D scene from the input images reconstruction3D = pySpatial.reconstruct(input_scene) # Step 2: get the camera extrinsic for the second viewpoint (index 1) viewpoint_2 = reconstruction3D.extrinsics[1] # Step 3: rotate the camera to the right from viewpoint 2 to see what is on the # right side of the black chair novel_viewpoint = pySpatial.rotate_right(viewpoint_2) # Step 4: synthesize the novel view from the rotated camera pose novel_view = pySpatial.synthesize_novel_view(reconstruction3D, novel_viewpoint) # Step 5: return the novel synthesized view for visual inspection return novel_view </pre> 	3D Reconstruction: 	Program Output: 																																			
383	Final Response: From the rendered point cloud image, we can see a large grey sofa. Its shape, cushion structure, and placement along the wall make it clearly identifiable as a sofa rather than a table, cabinet, or office setup.																																					
384	Query: From viewpoint 4, if I turn left and move forward, will I get closer to the two single sofas?	Ground Truth: Yes.																																				
385	<pre data-bbox="290 747 1101 897"> 1 2 def program(input_scene: Scene): # Reconstruct the 3D scene from input images reconstructed_scene = pySpatial.reconstruct(input_scene) # Get the camera extrinsics of image 4 (index 3) base_viewpoint = reconstructed_scene.extrinsics[3] # Turn left from the base viewpoint viewpoint_turn_left = pySpatial.rotate_left(base_viewpoint) # Move forward from the turned left viewpoint viewpoint_move_forward = pySpatial.move_forward(viewpoint_turn_left) # Synthesize novel views for the turned left and moved forward viewpoints image_turn_left = pySpatial.synthesize_novel_view(reconstructed_scene, viewpoint_turn_left) image_move_forward = pySpatial.synthesize_novel_view(reconstructed_scene, viewpoint_move_forward) # Return the two images for visual comparison to check if closer to the sofas return [image_turn_left, image_move_forward] </pre> 	3D Reconstruction: 	Program Output: 																																			
386	Final Response: The second view shows the two single sofas prominently and larger, suggesting a closer distance.	Ground Truth: Yes.																																				
387	Query: From viewpoint 3, what is to the right of the blue bag?	Ground Truth: Black sofa.																																				
388	<pre data-bbox="290 1001 1019 1151"> 1 2 def program(input_scene: Scene): # Reconstruct the 3D scene from the four images reconstructed_scene = pySpatial.reconstruct(input_scene) # Get the camera extrinsics for image 3 (index 2) viewpoint_3 = reconstructed_scene.extrinsics[2] # Rotate the camera right from viewpoint 3 to see what is # to the right of the blue bag viewpoint_right_of_3 = pySpatial.rotate_right(viewpoint_3) # Synthesize the novel view from the rotated camera pose novel_view = pySpatial.synthesize_novel_view(reconstructed_scene, viewpoint_right_of_3) # Return the novel view image for visual inspection to identify # what is to the right of the blue bag return novel_view </pre> 	3D Reconstruction: 	Program Output: 																																			
389	Final Response: The novel view synthesized by rotating the camera to the right of the viewpoint 3 shows a black sofa point cloud. The shape, color, and texture in that part align well with the appearance of a black sofa rather than a window, plush toy, or display shelves.																																					
390	Figure 2: Qualitative results on four examples from MINDCUBE. We show that pySpatial enables MLLMs to explicitly reason within the reconstructed explorable 3D scene, effectively addressing diverse spatial reasoning tasks through interpretable and executable 3D visual programs.																																					
391	room toy). From limited 2D views, pySpatial reconstructs an explorable 3D scene, infers camera poses via visual programming, and generates a structured motion plan for the robot to execute.																																					
392	As shown in Figure 3, our pySpatial successfully guides the robot through doorways, make correct turns, and finally toward the correct goal location. Notably, the MLLM baseline GPT-4.1 struggles to resolve relative direction such as left-right and fails to provide absolute metric distance estimates, leading to navigation errors. In contrast, our agent outputs precise rotations and translations that align with real-world execution, resulting in reliable task completion. This experiment demonstrates that our approach not only produces coherent spatial reasoning in question answering benchmarks, but also transfers effectively to physical robotic platforms for complex indoor navigation tasks.																																					
393	4.5 DISCUSSIONS																																					
394	Ablation Study on the Code Agent. To ablate the effect of our code agent, we conduct experiments on the MINDCUBE-1k benchmark by comparing the performance of various MLLM baselines with and without integration of pySpatial. As summarized in Table 4, augmenting models with pySpatial consistently leads to substantial improvements across all tested MLLMs, including GPT-4o, GPT-4.1-mini, and GPT-4.1. For instance, GPT-4o im-	Table 4: Ablation study on the code agent. We report the accuracy on the MINDCUBE-1k dataset.	<table border="1"> <thead> <tr> <th>Method</th><th>Overall</th><th>Rotation</th><th>Among</th><th>Around</th></tr> </thead> <tbody> <tr> <td>GPT-4o</td><td>42.29</td><td>35.00</td><td>43.00</td><td>46.40</td></tr> <tr> <td>+ pySpatial</td><td>62.67</td><td>41.00</td><td>65.00</td><td>66.33</td></tr> <tr> <td>GPT-4.1-mini</td><td>43.34</td><td>36.00</td><td>45.00</td><td>44.80</td></tr> <tr> <td>+ pySpatial</td><td>58.19</td><td>37.50</td><td>62.00</td><td>65.60</td></tr> <tr> <td>GPT-4.1</td><td>44.67</td><td>35.50</td><td>45.33</td><td>50.40</td></tr> <tr> <td>+ pySpatial</td><td>63.42</td><td>42.00</td><td>66.33</td><td>73.60</td></tr> </tbody> </table>	Method	Overall	Rotation	Among	Around	GPT-4o	42.29	35.00	43.00	46.40	+ pySpatial	62.67	41.00	65.00	66.33	GPT-4.1-mini	43.34	36.00	45.00	44.80	+ pySpatial	58.19	37.50	62.00	65.60	GPT-4.1	44.67	35.50	45.33	50.40	+ pySpatial	63.42	42.00	66.33	73.60
Method	Overall	Rotation	Among	Around																																		
GPT-4o	42.29	35.00	43.00	46.40																																		
+ pySpatial	62.67	41.00	65.00	66.33																																		
GPT-4.1-mini	43.34	36.00	45.00	44.80																																		
+ pySpatial	58.19	37.50	62.00	65.60																																		
GPT-4.1	44.67	35.50	45.33	50.40																																		
+ pySpatial	63.42	42.00	66.33	73.60																																		



451 **Figure 3: Qualitative results on real-world robot navigation.** We deploy pySpatial on a Unitree-
 452 Go1 robot to navigate toward a target object (mushroom toy) using limited views as input. The figure
 453 shows the reconstructed 3D scene, motion plans, and physical execution. Compared to the GPT-
 454 4.1 baseline, which fails due to an incorrect initial turn and produces a collision-prone trajectory,
 455 pySpatial generates a geometrically consistent plan that successfully reaches the goal.

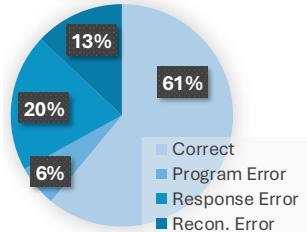
456 proves from 42.3% to 62.7% overall accuracy, indicating that pySpatial generalizes across different
 457 MLLMs and effectively enhance spatial reasoning.

458 **Failure Case Analysis.** From the MINDCUBE benchmark, we select a representative subset of about 100 samples and conduct a manual analysis to identify the underlying sources of error in cases where the final response is incorrect. As shown in Figure 4, among the 39% of failure cases, only 6% are attributable to incorrectly generated visual programs that fail to address the query, validating the effectiveness of our overall programming pipeline. Beyond this, 20% of errors arise from the MLLMs at the final reasoning step, while 13% stem from limitations in the 3D reconstruction models, where the generated visual programs are correct but the program outputs do not provide useful information. These results also suggest that advances in 3D reconstruction and code generation models hold the potential to further enhance our performance.

471 **Remarks on Efficiency.** Our pySpatial completes the MINDCUBE-1k benchmark in 2.17 hours
 472 on a single GPU using a single thread for 1,050 queries, averaging 7.45 seconds per query. As the
 473 breakdown, code generation requires 2.41 seconds, program execution 2.14 seconds, and answer
 474 generation 2.90 seconds. For comparison, VADAR (Marsili et al., 2025) requires 17.25 seconds per
 475 query on average. These results demonstrate that our visual programming framework enhances the
 476 spatial reasoning capabilities of MLLMs while remaining efficient to deploy without excessive cost.

477 5 CONCLUSION

479 In this work, we present pySpatial, a visual programming framework that enhance spatial reasoning
 480 capabilities of MLLMs through zero-shot Python code generation. By composing functions
 481 such as 3D reconstruction and novel-view synthesis, pySpatial converts 2D image sequences into
 482 explorable 3D scenes, enabling explicit reasoning in 3D space. Experiments on the MINDCUBE
 483 and OMNI3D-BENCH benchmarks demonstrate that pySpatial consistently outperforms strong
 484 MLLM baselines, with gains of up to 12.94% on MINDCUBE compared to GPT-4.1-mini. Beyond
 485 benchmarks, real-world indoor navigation experiments further validate its practicality, showing that
 robots can successfully traverse complex environments using route plans generated by pySpatial.



459 **Figure 4: Failure case study.**
 460 We manually examine the error sources in about 100 samples from MINDCUBE.

486 REPRODUCIBILITY STATEMENT
487

488 We are committed to ensuring the reproducibility of our results. All the spatial tools used in this
489 work are open-sourced, and the benchmark datasets we evaluate on are publicly available. We have
490 provided detailed descriptions of our experimental setup and implementation details in Section 4 and
491 Appendix to facilitate reproducibility. Code will be publicly available upon acceptance to enable the
492 community to reproduce our experiments and build upon our work.

493 REFERENCES
494

495 Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel
496 Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language
497 model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–
498 23736, 2022. 3

499 Anthropic. Claude 3.5 sonnet, October 2024. URL <https://www.anthropic.com/news/claude-3-5-sonnet>. 7

500 Anthropic. Claude 4 sonnet system card, May 2025. URL <https://www-cdn.anthropic.com/6be99a52cb68eb70eb9572b4cafad13df32ed995.pdf>. 6

501 Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang,
502 Shijie Wang, Jun Tang, et al. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*,
503 2025. 3, 5, 6

504 Davide Bucciarelli, Nicholas Moratelli, Marcella Cornia, Lorenzo Baraldi, and Rita Cucchiara. Personalizing
505 multimodal large language models for image captioning: an experimental analysis. In *European Conference on Computer Vision*, pp. 351–368. Springer, 2024. 1

506 Chun-Peng Chang, Alain Pagani, and Didier Stricker. 3d spatial understanding in mllms: Disambiguation
507 and evaluation. In *IEEE International Conference on Robotics and Automation*, pp. 13537–13544. IEEE, 2025. 2

508 David Charatan, Sizhe Lester Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelSplat: 3d gaussian
509 splats from image pairs for scalable generalizable 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19457–19467, 2024. 3

510 Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia.
511 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,
512 2024a. 2, 3, 5, 6, 7

513 Zhe Chen, Jiannan Wu, Wenhui Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong
514 Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning
515 for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24185–24198, 2024b. 3, 6

516 An-Chieh Cheng, Hongxu Yin, Yang Fu, Qiushan Guo, Ruihan Yang, Jan Kautz, Xiaolong Wang,
517 and Sifei Liu. Spatialrgpt: Grounded spatial reasoning in vision-language models. *Advances in
518 Neural Information Processing Systems*, 37:135062–135093, 2024. 2, 3

519 Jianguo Duan, Liwen Zhuang, Qinglei Zhang, Ying Zhou, and Jiyun Qin. Multimodal perception-
520 fusion-control and human–robot collaboration in manufacturing: A review. *The International
521 Journal of Advanced Manufacturing Technology*, 132(3):1071–1093, 2024. 2

522 Zhiwen Fan, Jian Zhang, Renjie Li, Junge Zhang, Runjin Chen, Hezhen Hu, Kevin Wang, Huaizhi
523 Qu, Dilin Wang, Zhicheng Yan, et al. Vlm-3r: Vision-language models augmented with
524 instruction-aligned 3d reconstruction. *arXiv preprint arXiv:2505.20279*, 2025. 5, 6

525 Chaoyou Fu, Yuhang Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu
526 Zhou, Yunhang Shen, Mengdan Zhang, et al. Video-mme: The first-ever comprehensive eval-
527 uation benchmark of multi-modal llms in video analysis. In *Proceedings of the Computer Vision
528 and Pattern Recognition Conference*, pp. 24108–24118, 2025. 2

540 Google. Our next-generation model: Gemini 1.5, February 2024. URL <https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/>. 7

541

542

543 Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning
544 without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern
545 Recognition*, pp. 14953–14962, 2023. 2, 3, 5, 7

546 Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot
547 navigation. In *IEEE International Conference on Robotics and Automation*, pp. 10608–10615.
548 IEEE, 2023. 3

549 Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot
550 planners: Extracting actionable knowledge for embodied agents. In *International Conference on
551 Machine Learning*, pp. 9118–9147. PMLR, 2022. 3

552

553 Yuheng Ji, Huajie Tan, Jiayu Shi, Xiaoshuai Hao, Yuan Zhang, Hengyuan Zhang, Pengwei Wang,
554 Mengdi Zhao, Yao Mu, Pengju An, et al. Robobrain: A unified brain model for robotic manipu-
555 lation from abstract to concrete. In *Proceedings of the Computer Vision and Pattern Recognition
556 Conference*, pp. 1724–1734, 2025. 6

557 Amita Kamath, Jack Hessel, and Kai-Wei Chang. What’s “up” with vision-language models? inves-
558 tigating their struggle with spatial reasoning. In *Proceedings of the 2023 Conference on Empirical
559 Methods in Natural Language Processing*, pp. 9161–9175, 2023. 3

560 Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. Referitgame: Referring to
561 objects in photographs of natural scenes. In *Proceedings of the Conference on Empirical Methods
562 in Natural Language Processing*, pp. 787–798, 2014. 2

563

564 Vincent Leroy, Yohann Cabon, and Jérôme Revaud. Grounding image matching in 3d with mast3r.
565 In *European Conference on Computer Vision*, pp. 71–91. Springer, 2024. 3

566

567 Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan
568 Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. LLaVA-onevision: Easy visual task transfer.
569 *Transactions on Machine Learning Research*, 2025. URL <https://openreview.net/forum?id=zKv8qULV6n>. 5, 6

570

571 Xiaoqi Li, Mingxu Zhang, Yiran Geng, Haoran Geng, Yuxing Long, Yan Shen, Renrui Zhang,
572 Jiaming Liu, and Hao Dong. Manipllm: Embodied multimodal large language model for object-
573 centric robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision
574 and Pattern Recognition*, pp. 18061–18070, 2024. 2

575

576 Chenguo Lin, Yuchen Lin, Panwang Pan, Yifan Yu, Honglei Yan, Katerina Fragkiadaki, and
577 Yadong Mu. Movies: Motion-aware 4d dynamic view synthesis in one second. *arXiv preprint
arXiv:2507.10065*, 2025. 3

578

579 Fangyu Liu, Guy Emerson, and Nigel Collier. Visual spatial reasoning. *Transactions of the Association
580 for Computational Linguistics*, 11:635–651, 2023. 2

581

582 Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction
583 tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 26296–26306, 2024a. 3

584

585 Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan
586 Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training
587 for open-set object detection. In *European Conference on Computer Vision*, pp. 38–55. Springer,
2024b. 3

588

589 Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren,
590 Zhuoshu Li, Hao Yang, et al. Deepseek-vl: towards real-world vision-language understanding.
591 *arXiv preprint arXiv:2403.05525*, 2024. 6

592

593 Chuwei Luo, Yufan Shen, Zhaoqing Zhu, Qi Zheng, Zhi Yu, and Cong Yao. Layoutllm: Layout
instruction tuning with large language models for document understanding. In *Proceedings of the
IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15630–15640, 2024. 2

594 Wufei Ma, Yu-Cheng Chou, Qihao Liu, Xingrui Wang, Celso de Melo, Jianwen Xie, and Alan
 595 Yuille. Spatialreasoner: Towards explicit and generalizable 3d spatial reasoning. *arXiv preprint*
 596 *arXiv:2504.20024*, 2025. 2

597 Dominic Maggio, Hyungtae Lim, and Luca Carlone. Vggt-slam: Dense rgb slam optimized on the
 598 sl (4) manifold. *arXiv preprint arXiv:2505.12549*, 2025. 3

599 Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L Yuille, and Kevin Murphy.
 600 Generation and comprehension of unambiguous object descriptions. In *Proceedings of the IEEE*
 601 *Conference on Computer Vision and Pattern Recognition*, pp. 11–20, 2016. 2

602 Damiano Marsili, Rohun Agrawal, Yisong Yue, and Georgia Gkioxari. Visual agentic ai for spatial
 603 reasoning with a dynamic api. In *Proceedings of the Computer Vision and Pattern Recognition*
 604 *Conference*, pp. 19446–19455, 2025. 3, 5, 6, 7, 9

605 Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document
 606 images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*,
 607 pp. 2200–2209, 2021. 2

608 Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and CV Jawahar.
 609 Infographicvqa. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer*
 610 *Vision*, pp. 1697–1706, 2022. 2

611 Meta. Llama 3.2: Revolutionizing edge ai and vision with open, cus-
 612 tomizable models, September 2024. URL <https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/>. 7

613 OpenAI. Hello gpt-4o, August 2024. URL <https://openai.com/index/hello-gpt-4o/>. 6, 7

614 OpenAI. Introducing gpt-4.1 in the api, April 2025. URL <https://openai.com/index/gpt-4-1/>. 6

615 Yanyuan Qiao, Haodong Hong, Wenqi Lyu, Dong An, Siqi Zhang, Yutong Xie, Xinyu Wang, and
 616 Qi Wu. Navbench: Probing multimodal large language models for embodied navigation. *arXiv*
 617 *preprint arXiv:2506.01031*, 2025. 2, 3

618 Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham
 619 Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Va-
 620 sudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollar, and Christoph Fe-
 621 ichtenhofer. SAM 2: Segment anything in images and videos. In *International Conference on*
 622 *Learning Representations*, 2025. URL <https://openreview.net/forum?id=Ha6RTeWMd0>. 3

623 Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings*
 624 *of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4104–4113, 2016. 3

625 Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugging-
 626 gpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information*
 627 *Processing Systems*, 36:38154–38180, 2023. 3

628 Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for
 629 robotic manipulation. In *Conference on Robot Learning*, pp. 785–799. PMLR, 2023. 3

630 Chan Hee Song, Valts Blukis, Jonathan Tremblay, Stephen Tyree, Yu Su, and Stan Birchfield. Ro-
 631 bospatial: Teaching spatial understanding to 2d and 3d vision-language models for robotics.
 632 In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 15768–15780,
 633 2025. 2

634 Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for
 635 reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp.
 636 11888–11898, 2023. 2, 3, 4, 5, 6, 7

637 Peter Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Adithya Jairam Vedagiri IYER, Sai Charitha
 638 Akula, Shusheng Yang, Jihan Yang, Manoj Middepogu, Ziteng Wang, et al. Cambrian-1: A fully
 639 open, vision-centric exploration of multimodal llms. *Advances in Neural Information Processing*
 640 *Systems*, 37:87310–87356, 2024. 3

648 Jianyu Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David
 649 Novotny. Vggt: Visual geometry grounded transformer. In *Proceedings of the Computer Vision*
 650 and *Pattern Recognition Conference*, pp. 5294–5306, 2025a. 2, 3, 4, 5

651 Jiayu Wang, Yifei Ming, Zhenmei Shi, Vibhav Vineet, Xin Wang, Sharon Li, and Neel Joshi. Is
 652 a picture worth a thousand words? delving into spatial reasoning for vision language models.
 653 *Advances in Neural Information Processing Systems*, 37:75392–75421, 2024a. 3

654 Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A Efros, and Angjoo Kanazawa. Con-
 655 tinuous 3d perception model with persistent state. In *Proceedings of the Computer Vision and*
 656 *Pattern Recognition Conference*, pp. 10510–10522, 2025b. 3, 4, 5, 6

657 Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. Dust3r: Ge-
 658 ometric 3d vision made easy. In *Proceedings of the IEEE/CVF Conference on Computer Vision*
 659 and *Pattern Recognition*, pp. 20697–20709, 2024b. 2, 3

660 Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Vi-
 661 sual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint*
 662 *arXiv:2303.04671*, 2023. 3

663 Diankun Wu, Fangfu Liu, Yi-Hsin Hung, and Yueqi Duan. Spatial-mllm: Boosting mllm capabilities
 664 in visual-based spatial intelligence. *arXiv preprint arXiv:2505.23747*, 2025. 2, 6

665 Jiannan Wu, Muyan Zhong, Sen Xing, Ziqiang Lai, Zhaoyang Liu, Zhe Chen, Wenhui Wang,
 666 Xizhou Zhu, Lewei Lu, Tong Lu, et al. Visionllm v2: An end-to-end generalist multimodal
 667 large language model for hundreds of vision-language tasks. *Advances in Neural Information*
 668 *Processing Systems*, 37:69925–69975, 2024. 1

669 Jihan Yang, Shusheng Yang, Anjali W Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. Thinking in
 670 space: How multimodal large language models see, remember, and recall spaces. In *Proceedings*
 671 of the *Computer Vision and Pattern Recognition Conference*, pp. 10632–10643, 2025. 2, 7

672 Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng
 673 Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal
 674 reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023. 3

675 Jiabo Ye, Haiyang Xu, Haowei Liu, Anwen Hu, Ming Yan, Qi Qian, Ji Zhang, Fei Huang, and
 676 Jingren Zhou. mPLUG-owl3: Towards long image-sequence understanding in multi-modal large
 677 language models. In *International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=pr37sbuhVa>. 6

678 Baiqiao Yin, Qineng Wang, Pingyu Zhang, Jianshu Zhang, Kangrui Wang, Zihan Wang, Jieyu
 679 Zhang, Keshigeyan Chandrasegaran, Han Liu, Ranjay Krishna, et al. Spatial mental modeling
 680 from limited views. *arXiv preprint arXiv:2506.21458*, 2025. 1, 2, 3, 5, 6, 7

681 Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang,
 682 and Lijuan Wang. Mm-vet: Evaluating large multimodal models for integrated capabilities. In
 683 *International Conference on Machine Learning*, pp. 57730–57754. PMLR, 2024. 3

684 Xiangyu Zeng, Kunchang Li, Chenting Wang, Xinhao Li, Tianxiang Jiang, Ziang Yan, Songze
 685 Li, Yansong Shi, Zhengrong Yue, Yi Wang, Yali Wang, Yu Qiao, and Limin Wang. Time-
 686 suite: Improving MLLMs for long video understanding via grounded tuning. In *International*
 687 *Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=nAVejJURqZ>. 2

688 Yuanhan Zhang, Jinming Wu, Wei Li, Bo Li, Zejun MA, Ziwei Liu, and Chunyuan Li. LLaVA-
 689 video: Video instruction tuning with synthetic data. *Transactions on Machine Learning Research*,
 690 2025. URL <https://openreview.net/forum?id=EE1FGvt39K>. 6

691 Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing.
 692 *arXiv preprint arXiv:1801.09847*, 2018. 5

693

702 Generating 3D Visual Programs for Zero-Shot Spatial Reasoning

703

704 Appendix

705

706 In the appendix, we provide additional experimental results and implementation details of our
 707 pySpatial framework. The appendix is organized as follows:

- 709 • Section A provides more experimental results.
- 710 • Section B shows the API specification for pySpatial.
- 711 • Section C presents the prompt implementation for our agent.
- 712 • Section D discloses the use of large language models.

715 A MORE EXPERIMENTAL RESULTS

717 A.1 RESULTS ON MMSI-BENCH

719 We further evaluate our approach on MMSI-Bench and report the results in Table A1. We observe
 720 that pySpatial improves the overall MMSI-Bench performance by 6.4% on average, further demon-
 721 strating the effectiveness of our approach.

722 Table A1: Evaluation results on MMSI-Bench. Our pySpatial is based on GPT-4o.

724 Models	Positional Relationship				Attribute			Motion		MSR	Avg.
	Cam.–Cam.	Obj.–Obj.	Reg.–Reg.	Cam.–Obj.	Obj.–Reg.	Cam.–Reg.	Meas.	Appr.	Cam.	Obj.	
725 Proprietary											
GPT-5	43.0	35.1	32.1	48.8	42.4	51.8	60.9	36.4	32.4	36.8	42.0
o3	45.2	39.4	37.0	44.2	47.1	62.6	54.7	28.8	31.1	32.9	34.9
GPT-4.5	34.4	29.8	39.5	51.2	47.1	55.4	39.1	33.3	41.9	40.8	36.4
GPT-4.1	36.6	26.6	27.2	29.1	36.5	27.7	37.5	24.2	36.5	32.9	28.8
GPT-4o	34.4	24.5	23.5	19.8	37.6	27.7	32.8	31.8	35.1	36.8	30.8
Gemini-2.5-Pro	39.7	31.9	39.5	45.3	35.2	43.3	51.5	21.2	36.4	30.2	34.3
730 Open-source											
InternVL3-78B	34.4	23.4	32.1	12.8	37.6	26.5	37.5	19.7	28.4	31.6	29.3
InternVL2.5-78B	23.7	22.3	39.5	29.1	31.8	42.2	35.9	19.7	17.6	26.3	27.3
Qwen2.5-VL-72B	25.8	34.0	34.6	23.3	34.1	36.1	45.3	27.3	27.0	30.3	27.3
LLaVA-OneVision-72B	43.0	31.9	33.3	30.2	37.6	38.6	28.1	19.7	13.5	32.9	15.7
734 Baseline											
GPT-4o	34.4	24.5	23.5	19.8	37.6	27.7	32.8	31.8	35.1	36.8	30.8
+ pySpatial (Ours)	51.6	28.7	27.2	20.9	41.2	38.6	46.9	39.4	46.0	38.2	36.4

737 A.2 MORE ABLATION STUDIES

739 **Code Agents.** In Table A2, we compare the performance of our pySpatial framework when
 740 paired with different code agents. Across all categories, pySpatial consistently improves upon
 741 the base GPT-4o model, regardless of the underlying LLM used for code generation. Among the
 742 evaluated agents, GPT-4o achieves the strongest overall performance, reaching 62.67% accuracy,
 743 while Qwen3-Coder and DeepSeek-v3 deliver comparable results at 62.10% and 61.05%, respec-
 744 tively. Notably, Qwen3-Coder performs best on the *Around* category, whereas GPT-4o provides the
 745 most balanced improvements across all task types. These findings indicate that our framework is
 746 robust to the choice of code agent and that most of the performance gains stem from the 3D visual
 747 programming paradigm rather than the specific code LLM used.

748 **3D Reconstruction Backbones.** In Table A3, we compare the impact of different 3D recon-
 749 struction backbones on the performance of our pySpatial framework. All three backbones, including
 750 VGGT, Pi3, and CUT3R, lead to substantial improvements over the GPT-4o baseline, indicating
 751 that our 3D visual programming paradigm is robust to the choice of reconstruction method. Among
 752 them, Pi3 achieves the best overall performance (63.33%), with notable gains in the *Rotation* and
 753 *Among* categories. VGGT provides similarly strong results, while CUT3R performs slightly lower
 754 but still significantly surpasses the base model. These findings suggest that pySpatial can effec-
 755 tively leverage a range of modern reconstruction backbones, and its spatial reasoning improvements
 are not tied to a specific reconstruction architecture.

756

757 Table A2: Performance comparison of our pySpatial framework with different code agents.

758

Method	Overall	Rotation	Among	Around
GPT-4o	42.29	35.00	43.00	46.20
pySpatial w/ GPT-4o	62.67	41.00	66.33	71.20
pySpatial w/ Qwen3-Coder	62.10	40.00	64.50	74.00
pySpatial w/ DeepSeek-v3	61.05	40.50	65.33	64.80

764

765 Table A3: Performance comparison of pySpatial using different 3D reconstruction backbones.

766

Method	Overall	Rotation	Among	Around
GPT-4o	42.29	35.00	43.00	46.20
pySpatial w/ VGGT	62.67	41.00	66.33	71.20
pySpatial w/ Pi3	63.33	43.50	65.66	72.33
pySpatial w/ CUT3R	61.05	40.50	64.33	69.60

772

773 Table A4: Comparison of pySpatial with different numbers of in-context learning examples.

774

Method	Overall	Rotation	Among	Around
GPT-4o	42.29	35.00	43.00	46.20
pySpatial w/ 0 examples	53.62	32.50	55.16	66.80
pySpatial w/ 2 examples	62.67	41.00	66.33	71.20
pySpatial w/ 4 examples	63.14	46.00	64.50	73.60

780

In-Context Learning Examples for Code Agents. In Table A4, we examine how the number of in-context learning examples influences the performance of our pySpatial framework. We observe a clear upward trend: providing even a small number of examples substantially improves performance across all categories. Using 0 examples already offers a strong boost over the base GPT-4o model (53.62% vs. 42.29%), demonstrating that pySpatial can operate effectively even without demonstration guidance. Adding 2 examples leads to a significant further gain, reaching 62.67% overall accuracy. Increasing to 4 examples yields the best performance (63.14%), with notable improvements particularly in the *Rotation* and *Around* categories. These results suggest that pySpatial benefits from additional examples, but even minimal in-context supervision is sufficient to unlock strong spatial reasoning capabilities.

A.3 ADDITIONAL QUALITATIVE RESULTS

We present more qualitative results from MINDCUBE in Figure A1. Beyond the linear program flow demonstrated in Figure 2, pySpatial is also capable of executing more expressive control-flow operations, including for-loops, conditionals, and lambda-style functional compositions, allowing it to construct complex multi-step 3D operations. These richer program structures enable pySpatial to iteratively explore candidate viewpoints, selectively apply geometric predicates, and adapt its reasoning based on scene-dependent constraints. Across the presented examples, the 3D visual programs generated by our code agent consistently guide pySpatial toward responses that closely align with the ground-truth spatial relationships, further validating the correctness, interpretability, and robustness of our 3D reasoning framework.

A.4 ADDITIONAL REAL-WORLD EXPERIMENTS

We also include an additional qualitative example in a challenging dynamic scene, where the views are captured at different times and a person is moving through the environment, as shown in Figure A2. Despite the temporal inconsistency and the presence of dynamic elements, pySpatial remains effective: the reconstruction module robustly integrates the multi-view observations and preserves the stable structural cues needed for accurate spatial reasoning. In contrast, GPT-5 continues to struggle in this scenario, failing to generate a safe or correct navigation trajectory and often producing instructions that are incompatible with the underlying scene geometry. These extended real-world experiments further validate the robustness and practical effectiveness of our approach.

810

811

Query: From image 4, then I turn right and move forward, will I get closer to the door?



```

813     def program(input_scene: Scene):
814         # Reconstruct the 3D scene from the input images
815         reconstructed = pySpatial.reconstruct(input_scene)
816         base = reconstructed.extrinsics[3] # start from the same pose as image 4
817         actions = [
818             (pySpatial.rotate_right, {"angle": np.pi/4}), # turning right
819             (pySpatial.move_forward, {"distance": 0.3}), # walking forward
820         ]
821         clues = []
822         viewpoint = base
823
824         for fn, args in actions:
825             # Apply rotation/movement to simulate the physical turning/walking
826             # described in the question
827             viewpoint = fn(viewpoint, **args)
828             # Generate the view so I can compare how the door changes in appearance
829             # (e.g., Larger = closer)
830             clues.append(pySpatial.synthesize_novel_view(reconstructed, viewpoint))
831
832     return clues
  
```

Final Response: Based on the program's right-turn and forward-motion transformation and the resulting synthesized images, the doorway becomes larger and more prominent while the nearby dark chair-like object also appears bigger, indicating reduced distance.

Ground Truth: Yes.

3D Reconstruction:



Program Output:



Query: From the viewpoint presented in image 3, what is to the left of the blue ball?



```

824     def program(input_scene: Scene):
825         # Reconstruct 3D geometry from the four provided images.
826         reconstructed_scene = pySpatial.reconstruct(input_scene)
827         base_viewpoint = reconstructed_scene.extrinsics[2]
828         # Define transformations we want to apply:
829         # 1. A Left rotation from the base viewpoint.
830         # 2. Move forward toward the ball, then rotate Left.
831         transformations = [
832             ("turn_left", lambda vp: pySpatial.rotate_left(vp, np.pi/4)),
833             ("forward_then_left", lambda vp: pySpatial.rotate_left(pySpatial.move_forward(vp, 0.3)))
834         ]
835
836         # Store generated views here.
837         synthesized_views = []
838         # Apply transformations in a loop.
839         for name, transform_fn in transformations:
840             new_viewpoint = transform_fn(base_viewpoint)
841             # Synthesize novel view from that transformed camera pose.
842             # Each view should reveal environmental geometry Located LEFT of the ball relative to
843             # image 3's pose.
844             view = pySpatial.synthesize_novel_view(reconstructed_scene, new_viewpoint)
845             synthesized_views.append(view)
846
847         # Output the pair of visual clues that help identify what lies to the left of the ball
848         return synthesized_views
  
```

Final Response: Both synthesized viewpoints display the window, and the object to the left of the blue ball (from the viewpoint of image 3) is the yellow-framed window on the wall.

Ground Truth: Window.

3D Reconstruction:



Program Output:



Figure A1: **More qualitative examples from MINDCUBE.** We show that pySpatial enables MLLMs to explicitly reason within a reconstructed, explorable 3D scene, allowing the model not only to interpret spatial structure but also to compose executable 3D visual programs with control flow, such as for-loops to robustly solve diverse spatial reasoning tasks.

Query: Suppose you are looking at the viewpoint corresponding to Image 1, generate a task specification and motion plan for navigating to the goal destination at Image 4.

Input Limited Views

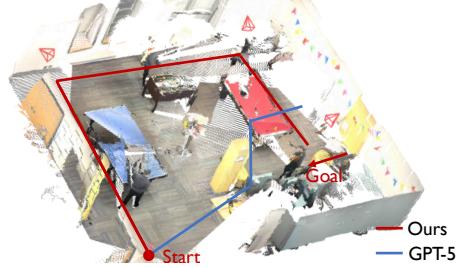


Proposed Motion Plan from pySpatial (Ours)



1. Turn left 37.6°, orienting yourself slightly toward the foosball table and white barstools, then walk forward 7.543 m, passing the ping-pong table on your right.
2. Turn right 84.2° so that you face toward the red-felt pool table beyond the foosball area, then walk forward 7.011 m, moving between the foosball table and the couch area.
3. Turn right 72.8°, aligning your view with the yellow ottoman, then walk forward 4.573 m to arrive at the goal.

Reconstructed 3D Scene & Motion Plan



Proposed Motion Plan from GPT-5



1. Turn 45° to your right and walk 4 meters forward toward the space beside the foosball table.
2. Turn 35° to your left and walk 2 meters forward toward the corner near the red pool table.
3. Turn 80° to your right and walk 2 meters forward toward the area with the guitar and chairs.

Figure A2: An additional qualitative real-world example in a challenging dynamic scene.

864
865

B IMPLEMENTATION FOR PYSPATIAL

866
867

A complete description of the API interface is provided in Code 2.

868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917

```

import os
import glob
from typing import List, Union
from pathlib import Path

from tool.reconstruct import reconstruct_3d
from tool.segment import segment_image, segment_automatic
from tool.estimate_depth import estimate_depth
from tool.camera_understanding import analyze_camera_trajectory
from tool.novel_view_synthesis import novel_view_synthesis, rotate_right, rotate_left,
    move_forward, move_backward, turn_around
import re

class Reconstruction:
    def __init__(self, point_cloud, extrinsics, intrinsics, colors=None):
        self.point_cloud = point_cloud
        self.extrinsics = extrinsics # list of 4 *4 numpy array
        self.intrinsics = intrinsics
        self.colors = colors # Add colors attribute

class Scene:
    """Simple scene class that holds image data."""

    def __init__(self, path_to_images: Union[str, List[str]], question: str = "", scene_id: str = None):
        self.question = question
        self.scene_id = scene_id
        self.original_path = path_to_images # Store original path for reconstruction
        self.images = self._load_images(path_to_images)
        self.reconstruction: Reconstruction = None
        self.code: str = None
        self.visual_clue = None

    def _load_images(self, path_to_images: Union[str, List[str]]) -> List[str]:
        """Load image paths from directory or list."""
        if isinstance(path_to_images, str):
            if os.path.isdir(path_to_images):
                # Check if this is a demo directory (contains .glb files)
                demo_path = Path(path_to_images)
                glb_files = list(demo_path.glob("*.glb"))

                if glb_files:
                    # This is a demo directory, load images from color/ subdirectory
                    color_dir = demo_path / "color"
                    if color_dir.exists():
                        image_extensions = ['*.png', '*.jpg', '*.jpeg']
                        images = []
                        for ext in image_extensions:
                            images.extend(glob.glob(os.path.join(str(color_dir), ext)))
                        return sorted(images)
                    else:
                        print(f"Warning: Demo directory detected but no color/ subdirectory found in {path_to_images}")
                        return []
                else:
                    # Regular directory, load all images from directory
                    image_extensions = ['*.png', '*.jpg', '*.jpeg']
                    images = []
                    for ext in image_extensions:
                        images.extend(glob.glob(os.path.join(path_to_images, ext)))
                    return sorted(images)
            else:
                # Single image file
                return [path_to_images]
        else:
            # List of image paths
            return list(path_to_images)

class pySpatial:
    """Simple interface for 3D vision tools."""
    # we disable other function for now

    @staticmethod

```

```

918
919     def reconstruct(scene: Scene):
920         """3D reconstruction from scene images."""
921
922         # Check if this is a demo directory (contains .glb files)
923         if isinstance(scene.original_path, str) and os.path.isdir(scene.original_path):
924             demo_path = Path(scene.original_path)
925             glb_files = list(demo_path.glob("*.glb"))
926
927             if glb_files:
928                 # This is a demo directory, pass the directory path for demo data loading
929                 result = reconstruct_3d(scene.original_path, scene_id=scene.scene_id)
930             else:
931                 # Regular reconstruction with image paths
932                 result = reconstruct_3d(scene.images, scene_id=scene.scene_id)
933
934             else:
935                 # Regular reconstruction with image paths
936                 result = reconstruct_3d(scene.images, scene_id=scene.scene_id)
937
938         # Convert the raw result dictionary to a Reconstruction object
939         point_cloud = result.get('points', None)
940         cameras = result.get('cameras', None)
941         colors = result.get('colors', None) # Get colors from result
942
943         # Convert point cloud to numpy if it's a tensor
944         if point_cloud is not None:
945             if hasattr(point_cloud, 'cpu'): # PyTorch tensor
946                 point_cloud = point_cloud.cpu().numpy()
947             elif hasattr(point_cloud, 'numpy'): # Other tensor types
948                 point_cloud = point_cloud.numpy()
949
950         # Convert colors to numpy if it's a tensor
951         if colors is not None:
952             if hasattr(colors, 'cpu'): # PyTorch tensor
953                 colors = colors.cpu().numpy()
954             elif hasattr(colors, 'numpy'): # Other tensor types
955                 colors = colors.numpy()
956
957         # Extract extrinsics and intrinsics from cameras if available
958         extrinsics = None
959         intrinsics = None
960
961         if cameras is not None:
962             # Assume cameras contains extrinsic matrices
963             extrinsics = cameras.cpu().numpy() if hasattr(cameras, 'cpu') else cameras
964
965         # Create and return Reconstruction object with colors
966         reconstruction = Reconstruction(point_cloud, extrinsics, intrinsics, colors)
967
968         # Store the raw result for debugging
969         reconstruction._raw_result = result
970
971         return reconstruction
972
973     @staticmethod
974     def describe_camera_motion(recon: Reconstruction):
975         """Describe camera motion from reconstruction results.
976         Args:
977         """
978
979         extrinsics = recon.extrinsics
980         return analyze_camera_trajectory(extrinsics)
981
982     @staticmethod
983     def synthesize_novel_view(recon: Reconstruction, new_camera_pose, width=512, height=512, out_path=None):
984         """Generate novel view synthesis from reconstruction results.
985         Args:
986             recon: Reconstruction object with point_cloud, extrinsics, intrinsics
987             new_camera_pose: 3x4 or 4x4 extrinsic matrix for the new viewpoint
988             width: output image width (default: 512)
989             height: output image height (default: 512)
990             out_path: output image path (default: None, returns image object if not provided)
991             Returns:
992                 str or image: path to the rendered image if out_path provided, otherwise
993                 image object
994             """
995
996         return novel_view_synthesis(recon, new_camera_pose, width, height, out_path)
997
998     @staticmethod
999     def rotate_right(extrinsic, angle=None):

```

```

972     """Rotate camera pose to the right"""
973     return rotate_right(extrinsic, angle)
974
975     @staticmethod
976     def rotate_left(extrinsic, angle=None):
977         """Rotate camera pose to the left"""
978         return rotate_left(extrinsic, angle)
979
980     @staticmethod
981     def move_forward(extrinsic, distance=None):
982         """Move camera pose forward"""
983         return move_forward(extrinsic, distance)
984
985     @staticmethod
986     def move_backward(extrinsic, distance=None):
987         """Move camera pose backward"""
988         return move_backward(extrinsic, distance)
989
990     @staticmethod
991     def turn_around(extrinsic):
992         """Turn camera pose around 180 degrees"""
993         return turn_around(extrinsic)
994
995     class Agent:
996         def __init__(self, api_key: str = None):
997             self.api_key = api_key or os.getenv('OPENAI_API_KEY')
998
999         def generate_code(self, scene: Scene):
1000             from agent.codeAgent.query import generate_code_from_query
1001             return generate_code_from_query(scene, self.api_key)
1002
1003         def parse_LLM_response(self, scene: Scene, response: str):
1004             """
1005                 Extracts the first python code block (```python ... ```) from text.
1006                 Returns the code as a string, or "" if not found.
1007             """
1008             from agent.codeAgent.execute import parse_LLM_response
1009             code = parse_LLM_response(response)
1010             scene.code = code
1011             return code
1012
1013         def execute(self, scene: Scene):
1014             """
1015                 Execute a code string with a scene and return the visual clue result.
1016             """
1017             # try:
1018             #     from agent.codeAgent.execute import execute_code
1019             #     program = execute_code(scene.code)
1020
1021             #     visual_clue = program(scene)
1022             #     return visual_clue
1023             # except Exception as e:
1024             #     import traceback
1025             #     error_details = f"Execution failed: {str(e)}\nTraceback: {traceback.
1026             format_exc()}"
1027             #     # Store the error for detailed reporting
1028             #     self.last_execution_error = error_details
1029             #     return f"there is an error during code generation, no visual clue provided.
1030             Error: {str(e)}"
1031
1032             from agent.codeAgent.execute import execute_code
1033             program = execute_code(scene.code)
1034
1035             visual_clue = program(scene)
1036             return visual_clue
1037
1038         def answer(self, scene: Scene, visual_clue):
1039             # answer the question with visual clue
1040             from agent.anwer import answer
1041
1042             # Set the visual clue in the scene
1043             scene.visual_clue = visual_clue
1044
1045             # Call the answer function with API key
1046             return answer(scene, self.api_key)
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
20310
20311
20312
20313
20314
20315
20316
20317
20318
20319
20320
20321
20322
20323
20324
20325
20326
20327
20328
20329
20330
20331
20332
20333
20334
20335
20336
20337
20338
20339
203310
203311
203312
203313
203314
203315
203316
203317
203318
203319
203320
203321
203322
203323
203324
203325
203326
203327
203328
203329
203330
203331
203332
203333
203334
203335
203336
203337
203338
203339
2033310
2033311
2033312
2033313
2033314
2033315
2033316
2033317
2033318
2033319
2033320
2033321
2033322
2033323
2033324
2033325
2033326
2033327
2033328
2033329
2033330
2033331
2033332
2033333
2033334
2033335
2033336
2033337
2033338
2033339
20333310
20333311
20333312
20333313
20333314
20333315
20333316
20333317
20333318
20333319
20333320
20333321
20333322
20333323
20333324
20333325
20333326
20333327
20333328
20333329
20333330
20333331
20333332
20333333
20333334
20333335
20333336
20333337
20333338
20333339
203333310
203333311
203333312
203333313
203333314
203333315
203333316
203333317
203333318
203333319
203333320
203333321
203333322
203333323
203333324
203333325
203333326
203333327
203333328
203333329
203333330
203333331
203333332
203333333
203333334
203333335
203333336
203333337
203333338
203333339
2033333310
2033333311
2033333312
2033333313
2033333314
2033333315
2033333316
2033333317
2033333318
2033333319
2033333320
2033333321
2033333322
2033333323
2033333324
2033333325
2033333326
2033333327
2033333328
2033333329
2033333330
2033333331
2033333332
2033333333
2033333334
2033333335
2033333336
2033333337
2033333338
2033333339
20333333310
20333333311
20333333312
20333333313
20333333314
20333333315
20333333316
20333333317
20333333318
20333333319
20333333320
20333333321
20333333322
20333333323
20333333324
20333333325
20333333326
20333333327
20333333328
20333333329
20333333330
20333333331
20333333332
20333333333
20333333334
20333333335
20333333336
20333333337
20333333338
20333333339
203333333310
203333333311
203333333312
203333333313
203333333314
203333333315
203333333316
203333333317
203333333318
203333333319
203333333320
203333333321
203333333322
203333333323
203333333324
203333333325
203333333326
203333333327
203333333328
203333333329
203333333330
203333333331
203333333332
203333333333
203333333334
203333333335
203333333336
203333333337
203333333338
203333333339
2033333333310
2033333333311
2033333333312
2033333333313
2033333333314
2033333333315
2033333333316
2033333333317
2033333333318
2033333333319
2033333333320
2033333333321
2033333333322
2033333333323
2033333333324
2033333333325
2033333333326
2033333333327
2033333333328
2033333333329
2033333333330
2033333333331
2033333333332
2033333333333
2033333333334
2033333333335
2033333333336
2033333333337
2033333333338
2033333333339
20333333333310
20333333333311
20333333333312
20333333333313
20333333333314
20333333333315
20333333333316
20333333333317
20333333333318
20333333333319
20333333333320
20333333333321
20333333333322
20333333333323
20333333333324
20333333333325
20333333333326
20333333333327
20333333333328
20333333333329
20333333333330
20333333333331
20333333333332
20333333333333
20333333333334
20333333333335
20333333333336
20333333333337
20333333333338
20333333333339
203333333333310
203333333333311
203333333333312
203333333333313
203333333333314
203333333333315
203333333333316
203333333333317
203333333333318
203333333333319
203333333333320
203333333333321
203333333333322
203333333333323
203333333333324
203333333333325
203333333333326
203333333333327
203333333333328
203333333333329
203333333333330
203333333333331
203333333333332
203333333333333
203333333333334
203333333333335
203333333333336
203333333333337
203333333333338
203333333333339
2033333333333310
2033333333333311
2033333333333312
2033333333333313
2033333333333314
2033333333333315
2033333333333316
2033333333333317
2033333333333318
2033333333333319
2033333333333320
2033333333333321
2033333333333322
2033333333333323
2033333333333324
2033333333333325
2033333333333326
2033333333333327
2033333333333328
2033333333333329
2033333333333330
2033333333333331
2033333333333332
2033333333333333
2033333333333334
2033333333333335
2033333333333336
2033333333333337
2033333333333338
2033333333333339
20333333333333310
20333333333333311
20333333333333312
20333333333333313
20333333333333314
20333333333333315
20333333333333316
20333333333333317
20333333333333318
20333333333333319
20333333333333320
20333333333333321
20333333333333322
20333333333333323
20333333333333324
20333333333333325
20333333333333326
20333333333333327
20333333333333328
20333333333333329
20333333333333330
20333333333333331
20333333333333332
20333333333333333
20333333333333334
20333333333333335
20333333333333336
20333333333333337
20333333333333338
20333333333333339
203333333333333310
203333333333333311
203333333333333312
203333333333333313
203333333333333314
203333333333333315
203333333333333316
203333333333333317
203333333333333318
203333333333333319
203333333333333320
203333333333333321
203333333333333322
203333333333333323
203333333333333324
203333333333333325
203333333333333326
203333333333333327
203333333333333328
203333333333333329
2033
```

1026 C IMPLEMENTATION DETAILS OF THE AGENT PROMPT IN PYSPATIAL
10271028 We present the prompts used in our experiments in the box below.
1029

```

1030
1031     AGENT PROMPT IN PYSPATIAL
1032
1033     task_description = """
1034         You are now asked to solve a spatial reasoning related problem.
1035         The input are image(s) and a natural language question that
1036         specifically designed to test your spatial reasoning ability.
1037         It is not trivial to solve these tasks directly as a vision
1038         language model. However, You have access to the following Python API:
1039         """
1040
1041     api_specification = """
1042         In the PySpatial API, we explicitly introduce the 3D inductive bias.
1043         We provide a Scene class that contains the image(s) and a question.
1044         Further, we also provide a 3D reconstruction process that can be
1045         used to generate a 3D point cloud and camera parameters.
1046
1047         class Reconstruction:
1048             def __init__(self, point_cloud, extrinsics, intrinsics):
1049                 self.point_cloud = point_cloud
1050                 self.extrinsics = extrinsics
1051                 self.intrinsics = intrinsics
1052
1053         class Scene:
1054             """Simple scene class that holds image data."""
1055             def __init__(self, path_to_images: Union[str, List[str]],
1056                         question: str = ""):
1057                 self.question = question
1058                 self.images = self._load_images(path_to_images)
1059                 self.reconstruction = None
1060
1061             def _load_images(self, path_to_images: Union[str, List[str]]):
1062                 """Load image paths from directory or list."""
1063                 if isinstance(path_to_images, str):
1064                     if os.path.isdir(path_to_images):
1065                         # Load all images from directory
1066                         image_extensions = ['*.png', '*.jpg', '*.jpeg']
1067                         images = []
1068                         for ext in image_extensions:
1069                             images.extend(glob.glob(os.path.join(
1070                                 path_to_images, ext)))
1071                         return sorted(images)
1072                 else:
1073                     # Single image file
1074                     return [path_to_images]
1075             else:
1076                 # List of image paths
1077                 return list(path_to_images)
1078
1079         class pySpatial:
1080             """Simple interface for 3D vision tools."""
1081             # we disable other function for now

```

```

1080
1081
1082     @staticmethod
1083     def reconstruct(scene: Scene):
1084         "3D reconstruction from scene images."
1085
1086         return reconstruct_3d(scene.images)
1087
1088     @staticmethod
1089     def describe_camera_motion(recon: Reconstruction):
1090         "Describe camera motion from reconstruction results.
1091         Args:
1092             "
1093         extrinsics = recon.extrinsics
1094         return describe_camera_motion(extrinsics)
1095
1096     @staticmethod
1097     def synthesize_novel_view(recon: Reconstruction,
1098                               new_camera_pose):
1099         "Generate novel view synthesis from reconstruction results.
1100         Args:
1101             "
1102         return novel_view_synthesis(recon)
1103
1104     # methods to manipulate camera pose
1105     def rotate_right(extrinsic, angle=np.pi/2):
1106
1107         def rotate_left(extrinsic, angle=np.pi/2):
1108
1109             def move_forward(extrinsic, distance=0.1):
1110
1111                 def move_backward(extrinsic, distance=0.1):
1112
1113                     def turn_around(extrinsic):
1114
1115
1116                     @staticmethod
1117                     def estimate_depth(image):
1118                         return estimate_depth(image)
1119
1120                     """
1121
1122                     # in-context learning examples
1123                     example_problems = """
1124                         Problem 1:
1125                         Question: "Based on these two views showing the same scene:
1126                         in which direction did I move from the first view to the
1127                         second view?
1128                         A. Diagonally forward and left
1129                         B. Directly right
1130                         C. Directly left
1131                         D. Diagonally forward and right"
1132
1133                         How to solve this problem?
1134                         Step 1: we can easily find the answer with camera extrinsics.
1135                         Step 2: therefore, we should first reconstruct the scene,
1136                         and then use the camera extrinsics to find the answer.
1137                         Step 3: it is still not trivial to directly get the answer
1138                         from extrinsic matrix.
1139                         Step 4: we can use the pySpatial.describe_camera_motion
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
```

```

1134
1135     to get the answer.
1136     Next, write python code within the pySpatial API provided,
1137     then an agent will automatically collect the code
1138     I wrote and execute it.
1139
1140     ```python
1141     def program(input_scene: Scene):
1142         reconstruction3D = pySpatial.reconstruct(input_scene)
1143         camera_motion = pySpatial.describe_camera_motion(
1144             reconstruction3D)
1145         return camera_motion
1146     ```
1147
1148     Step 5: After I get the visual clue from execution,
1149     I can easily match the answer:
1150
1151     Problem 2:
1152     Based on these four images (image 1, 2, 3, and 4)
1153     showing the pink bottle from different viewpoints (front, left, back,
1154     and right), with each camera aligned with room walls and partially
1155     capturing the surroundings: If I am standing at the same spot and
1156     facing the same direction as shown in image 1, then I turn right
1157     and move forward, will I get closer to the pink plush toy
1158     and headboard?
1159
1160     since we do not have the way to compare distance in 3D space,
1161     we can render two images, and use these two images as visual clue.
1162     ```python
1163
1164     def program(input_scene: Scene):
1165
1166         reconstructed_scene = pySpatial.reconstruct(input_scene)
1167         base_viewpoint = reconstructed_scene.extrinsics[0]
1168         # the image 1 indicates the 0th index in the array
1169
1170         viewpoint_turn_right = pySpatial.rotate_right
1171             (base_viewpoint)
1172         viewpoint_move_forward = pySpatial.move_forward
1173             (viewpoint_turn_right)
1174
1175         image_right = pySpatial.synthesize_novel_view
1176             (reconstructed_scene, viewpoint_turn_right)
1177         image_forward = pySpatial.synthesize_novel_view
1178             (reconstructed_scene, viewpoint_move_forward)
1179
1180         # we should compare these two images, check if the object
1181         # exists and if the distance is closer.
1182         visual_clue = [image_right, image_forward]
1183         return visual_clue
1184
1185     """
1186
1187     code_generation_prompt = f"""
1188         Now please utilize the PySpatial API and write a python function
1189         to solve the problem.
1190         Noted that you can first do reasoning and then write the code.
1191         But the code should be wrapped in the ```python ``` block.
1192         Write a compact code block
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
331
```

```

1188
1189     Also, the function written should be named as program
1190     and the input parameter should be a Scene object.
1191     for example,
1192     ```python
1193     def program(input_scene: Scene):
1194         ...
1195         return ...
1196     ...
1197
1198     try to add simple comments to the code to explain your logic.
1199
1200     Make sure to first reasoning, why we write program like this,
1201     because we have a pySpatial API that allows us to explore the 3D
1202     space, please first do a reasoning like (I want to know what is
1203     to the right of something, therefore I just render a novel view
1204     from that).
1205     """
1206
1207     # Prompt template for ReAct: ReAct: Synergizing Reasoning and Acting
1208     # in Language Models https://arxiv.org/abs/2210.03629
1209
1210     answer_background = f"""
1211         We are now solving a spatial reasoning problem.
1212         It is not trivial to solve these tasks directly as a vision language
1213         model.
1214         However, We have access to the following PySpatial API:
1215         {api_specification}
1216
1217         We generate a python code based on the PySpatial API to solve
1218         this problem.
1219         """
1220
1221
1222     answer_prompt = """
1223         Based on the code and the visual clue from the execution, answer
1224         the question.
1225         """
1226
1227     # Prompt for the answer without visual clue
1228     without_visual_clue_background = """
1229         Solve this spatial reasoning problem based on the question
1230         and the image input.
1231
1232         First, analyze the question, extract useful information from
1233         the question description, then try to answer it based on the
1234         useful visual information.
1235
1236         Give your best guess if you cannot find the best answer.
1237         """
1238
1239
1240
1241

```

D THE USE OF LARGE LANGUAGE MODELS (LLMs)

We employed LLMs solely as an auxiliary tool to polish the writing of this manuscript. They were used to improve grammar, clarity, and readability, but no LLMs were involved in ideation, data analysis, experiment design, or result interpretation.