Supplementary Material: Geometrically-Constrained Agent for Spatial Reasoning

A. Spatial Task Constraint

As introduced in the main paper, the core of Geometrically-Constrained Agent (GCA) paradigm is the **formal task constraint** $\mathcal{C}_{\text{task}}$. It serves as a deterministic bridge to resolve the fundamental semantic-to-geometric gap, effectively decoupling the VLM's role into a semantic analyst and a constrained task solver. Recall that $\mathcal{C}_{\text{task}}$ is formally defined as a tuple containing two key sub-constraints: the **Reference Frame Constraint** ($\mathcal{C}_{\mathcal{R}}$), which defines the coordinate system, and the **Objective Constraint** ($\mathcal{C}_{\mathcal{O}}$), which specifies what to measure within that frame. We guide the Visual Language Model to automatically generate $\mathcal{C}_{\text{task}}$ (for specific prompts used in GCA, refer to Section E). In this section, we first elaborate on the universality of this constraint across different spatial task domains.

A.1. Universality of $\mathcal{C}_{\text{task}}$ in Spatial Tasks

The $\mathcal{C}_{\text{task}}$ is not a rigid definition fixed to a single problem type, but rather a general principle for a wide range of spatial tasks with semantic-to-geometric gap. The core idea is to leverage the VLM's semantic advantage to formalize the most significant geometric ambiguity of the task. The nature of this ambiguity shifts depending on the task domain.

Spatial Understanding and Reasoning. For spatial reasoning tasks, such as those evaluated in the main paper [5, 7, 18, 20, 21], the objective is typically simple and explicitly stated in the query (*e.g.*, "what is the relative position?" or "which object is wider?"). Therefore, the objective constraint $\mathcal{C}_{\mathcal{O}}$ is often trivial to formalize. The primary geometric ambiguity lies in the reference frame constraint $\mathcal{C}_{\mathcal{R}}$. A query like "where is the table relative to you?" is unsolvable until the reference frame ("you") is geometrically grounded. GCA's formalization of $\mathcal{C}_{\mathcal{R}}$ (*e.g.*, object-based, camera-based, direction-based frames) is specifically designed to resolve this ambiguity.

Robotic Manipulation and Interaction. Conversely, for robotic manipulation and interaction tasks [1, 8, 11, 19, 22, 23], the reference frame constraint is often simple. The frame is typically the robot's egocentric perspective or a fixed world frame aligned with the primary camera. The geometric ambiguity shifts entirely to the objective constraint $\mathcal{C}_{\mathcal{O}}$. A command like "pour tea into the cup" has a trivial $\mathcal{C}_{\mathcal{R}}$ but a highly complex $\mathcal{C}_{\mathcal{O}}$. The objective is not a simple measurement but a complex, multi-stage procedure involving affordances, contact points, and trajectories. For example, ReKep [4] tackles the manipulation by instructing the VLM to formalize the objective $\mathcal{C}_{\mathcal{O}}$ as a set of *Relational Keypoint Constraints*. These constraints $\mathcal{C}_{\mathcal{O}}$ are literally generated as cost functions written by VLM in Python that map 3D keypoints to a numerical cost. The cost functions are then passed to an inverse kinematics solver ($\mathcal{F}_{\text{compute}}$) to find the optimal robot action. Similarly, EmbodiedCoder [9] formalizes the $\mathcal{C}_{\mathcal{O}}$ as an executable program. The VLM is prompted to first generate code for *geometric parameterization*, fitting point clouds to functional primitives like a rectangle and a hinge axis for a door. Through generating a precise, parametric motion that conforms to the geometric shape just defined, the Python interpreter ($\mathcal{F}_{\text{compute}}$) simply executes the code to produce the final waypoints.

These works are not in conflict with our proposed task constraint $\mathcal{C}_{\text{task}}$. Instead, they can serve as complementary examples of its core principle. They demonstrate how the $\mathcal{C}_{\mathcal{O}}$ for complex manipulation and interaction can be formalized as code, cost functions, or geometric constraints. These constraints are then passed to a solver, just as GCA proposes. This confirms the universality of the $\mathcal{C}_{\text{task}}$: whether the ambiguity lies in the reference frame or the objective, the first and most critical step is to use the VLM's semantic strength to formalize a deterministic, geometrically-sound constraint to bridge the semantic-to-geometric gap.

A.2. Generalizability of $\mathcal{R}$ Definintion

We define three types of reference frame $\mathcal{R}$ in GCA: object-based, camera-based and direction-based reference frame, providing a robust and flexible framework. We find that these categories are sufficient to cover the vast majority of static spatial reasoning queries encountered in existing benchmarks [5, 7, 20, 21]. As spatial reasoning advances toward more complex, dynamic, and abstract scenarios, we identify key limitations and challenges for the current implementation of $\mathcal{C}_{\mathcal{R}}$. These represent important avenues for future research.

Dynamic and Time-Varying Reference Frame. A signif-

Table A. **Ablation Study on Task Constraint.**

Tool Integration	Ref. $\mathcal{C}_{\mathcal{R}}$	Obj. $\mathcal{C}_{\mathcal{O}}$	MMSI-Bench
✓	✓	✓	**47.6**
✓	✓	✗	46.4
✓	✗	✓	41.0
✓	✗	✗	40.1
✗	✓	✓	33.5

icant challenge arises in video-based spatial reasoning [18], particularly in tasks involving continuous navigation or long-horizon agent actions. For example, an instruction like, "Move forward to the right first, then move backward to the right, and finally turn left," involves a $\mathcal{C}_{\mathcal{R}}$ that is continuously changing at each step, contingent on the agent's previous state. Solving this requires extending the $\mathcal{C}_{\mathcal{R}}$ formalization to become a time-dependent function $\mathcal{C}_{\mathcal{R}}(t)$, capable of tracking and updating the agent's pose and orientation throughout a sequence.

Frames from Abstract Concepts. This challenge emerges when the reference entity is not a rigid, easily-definable object. A query like "the living room is south of the kitchen" poses a significant problem. It is often impossible to compute a meaningful direction vector between the geometric centroids of two abstract areas or regions. Currently, GCA relies on proxies, for example, if such direction (from kitchen to living room) aligns with the camera's view from the background to the foreground, we might substitute $-Z_{\text{cam}}$ as the direction vector. This workaround, however, can introduce cumulative errors and lacks generalizability. Future work could explore novel methods to address this. One promising direction is to empower the VLM to directly annotate the reference frame, *i.e.*, outputting two points in the image whose corresponding 3D vector defines the abstract direction.

B. More Ablation Studies

To further explore the proposed formal task constraint $\mathcal{C}_{\text{task}}$ and the stability of GCA, we present four additional ablation studies. These experiments are designed to (1) precisely quantify the relative importance of the reference frame constraint ($\mathcal{C}_{\mathcal{R}}$) versus the objective constraint ($\mathcal{C}_{\mathcal{O}}$) for spatial reasoning tasks, (2) validate the constraint's effectiveness in improving the VLM's internal, tool-free reasoning, (3) evaluate the stability and robustness of the GCA paradigm, and (4) assess whether the additional computational overhead from multi-turn tool invocation yields corresponding performance improvements.

Importance of $\mathcal{C}_{\mathcal{R}}$ and $\mathcal{C}_{\mathcal{O}}$ in Spatial Reasoning. To validate our claim in Section A.1 that $\mathcal{C}_{\mathcal{R}}$ represents the primary geometric ambiguity in spatial reasoning tasks, we conduct a detailed ablation on the sub-components of $\mathcal{C}_{\text{task}}$. As shown in Table A, removing the objective constraint ($\mathcal{C}_{\mathcal{O}}$) results in a minor 1.2 point performance drop. This demonstrates that for spatial reasoning queries, the objective is often simple and clearly stated, allowing the task solver to infer it from the query during the $\mathcal{F}_{\text{compute}}$ stage. In contrast, removing the reference frame constraint ($\mathcal{C}_{\mathcal{R}}$) causes a 6.6 point performance drop. This suggests that $\mathcal{C}_{\mathcal{R}}$ is the most critical component in spatial reasoning, as it resolves the core geometric ambiguity of "from where" that VLMs cannot solve in their lossy semantic space.

$\mathcal{C}_{\text{task}}$ without Tool Integration. The $\mathcal{C}_{\text{task}}$ is not a prompt that can fix the VLM's internal spatial reasoning. Instead, it unlocks the VLM's agentic reasoning capability and coding skills by constraining the subsequent computation stage ($\mathcal{F}_{\text{compute}}$). This is confirmed by the results in Table A, where we compare GCA with a VLM that receives $\mathcal{C}_{\text{task}}$ as the prompt but relies solely on chain-of-thought (CoT) reasoning. With the constraint as a textual hint, it only yields a negligible 0.9 point improvement. Even when told the reference frame and objective, the VLM still cannot bypass the internal flawed spatial imagination and high-precision computation in its lossy semantic space.

Stability and Robustness Analysis. A key consideration for any agentic framework, especially one involving multiple VLM calls and tool interactions, is the stability of its results. The probabilistic nature of VLMs could potentially lead to high variance in final performance. To assess the robustness of GCA, we conduct $N = 10$ independent evaluation runs on the complete MMSI-Bench dataset, using the same Qwen3-VL-Thinking [13] for each run. All settings are kept identical as in the Table 1 (main paper). The mean accuracy and the standard deviation across all 10 runs is 47.6 ± 0.3. The results demonstrate a very low standard deviation, indicating that the GCA framework is highly stable. This stability is a direct benefit of our core design: by forcing the VLM to first generate a deterministic formal task constraint $\mathcal{C}_{\text{task}}$, we significantly reduce the stochasticity and ambiguity in the subsequent $\mathcal{F}_{\text{compute}}$ stage. The task solver operates within the non-negotiable geometric bounds defined by $\mathcal{C}_{\text{task}}$, leading to a consistent and verifiable reasoning pathway.

Inference Latency and Computation Costs. To evaluate the cost-efficiency of our tool-integrated GCA paradigm, we provide a detailed latency breakdown in Table B. While GCA requires longer inference time than end-to-end baselines (103.56s *vs.* 34.58s per query), it delivers substantial performance gains (47.6% *vs.* 32.6% for end-to-end baseline and 27.8% for TIGeR). To further quantify the cost-efficency of GCA, we measure the "Latency Cost Per 1% Gain", *i.e.*, the additional time required per percentage point of accuracy improvement over random guess (25%). Remarkably, GCA achieves 4.58s per 1% gain, nearly identical

Table B. **Measurement of Average Latency per Query on MMSI-Bench using Qwen3-VL-Think.**

Method	$\mathcal{F}_{\text{formalize}}$	$\mathcal{F}_{\text{compute}}$			Latency	Acc.	Latency Cost Per 1% Gain ↓
		#Round	Plan-Per-Round	Tool-Per-Round			
Random-Guess	–	–	–	–	–	25.0	–
TIGeR	–	2.58	18.71s	0.32s	49.10s	27.8	17.54s
Qwen3-VL-Think (end-to-end)	–	–	–	–	34.58s	32.6	4.55s
GCA	40.89s	3.78	16.22s	0.36s	103.56s	**47.6**	4.58s

to the end-to-end baseline (4.55s) and 3.8× more efficient than TIGeR (17.54s). This demonstrates that the additional latency from iterative tool calls and constraint formalization follows a linear efficiency scaling: the extra computational investment directly yields proportional reliability improvements without diminishing returns.

C. More Implementation Details

C.1. Visual Foundation Models

We deploy several Visual Foundation Models (VFMs) for agent to parameterize the visual world, facilitating the deterministic $\mathcal{F}_{\text{compute}}$ stage constrained by $\mathcal{C}_{\text{task}}$.

- **VGGT** [15]. Visual Geometry Grounded Transformer (VGGT) is a large feed-forward model that infers key 3D attributes from one or multiple images. It predicts camera parameters, point maps, and depth maps for all input views. Within GCA, VGGT serves as the primary geometry parameterization engine for 3D reconstruction.
- **MoGe-2** [16]. The Monocular Geometry (MoGe) estimation model is designed to recover 3D point maps with metric scale from a single image. It achieves this by decoupling the problem, predicting both an affine-invariant point map and a separate global scale factor. GCA leverages this unique capability to derive the correct real-world scale for the scene.
- **GroundingDINO** [10]. GroundingDINO is an open-set object detector that combines a transformer-based detector with grounded language pre-training. This architecture enables it to detect arbitrary objects specified by natural language, such as category names or referring expressions. It serves as a specialized detection tool in GCA.
- **SAM-2** [14]. SAM-2 is a foundation model for promptable visual segmentation in both images and videos. It generalizes the original SAM by incorporating a streaming memory architecture to handle temporal data. GCA uses SAM-2 as a bridge connecting pixels and boxes, allowing us to extract object point clouds from the VG-GT output based on the boxes.
- **Orient Anything** [17]. Orient Anything is trained on rendered 3D models to estimate the 3D orientation of an object from a single, free-view image. It predicts the object's azimuth, polar, and rotation angles relative to the camera. This capability is crucial for GCA to construct a object-based reference frame.

Note that these foundation models are not provided directly to the agent. Instead, we wrap them and offer some abstract tool interfaces as APIs for invocation (see Section C.2).

C.2. Tool Interfaces

The GCA agent's $\mathcal{F}_{\text{compute}}$ stage is driven by a discrete set of 8 exposed tool APIs. These APIs form the agent's action space, encapsulating the underlying VFMs.

- `reconstruct`. It ingests one or more images and produces a comprehensive 3D reconstruction. Internally, it leverages VGGT and automatically selects the optimal reconstruction strategy. If multiple, non-static images are provided, it first consults the VLM to identify common static objects for alignment. The output includes the 3D world points, camera extrinsics, and intrinsics.
- `detect`. It detects target objects in a single image based on a text prompt. For capable VLMs like Qwen3-VL-Thinking [13], we directly instruct the VLM itself to locate the target object through prompts. Otherwise, we use GroundingDINO as the detector. It returns the bounding boxes and corresponding labels.
- `project_box_to_3d_points`. It takes a 2D bounding box and projects it into the 3D world coordinate system defined by the VGGT model output. Internally, it first uses SAM-2 to convert the bounding box into a precise pixel mask, then filters the points using this mask.
- `predict_obj_pose`. It computes the 6-DoF semantic pose of an object, which is essential for establishing an object-based reference frame. This tool first calls `project_box_to_3d_points` to find the object's 3D centroid, and then calls Orient Anything to determine its 3D orientation. It then combines these to return the final object-to-world transformation matrix.
- `estimate_scale`. This tool is called when metric measurements (*e.g.*, "meters", "feet") are required. It aligns MoGe-2's metric depth with the VGGT model's relative depth prediction to compute a single scale factor that converts the entire reconstruction into meters.
- `ocr`. It performs optical character recognition (OCR) on an image using the EasyOCR library. It returns a list of recognized texts and their bounding boxes.

- `analyze_motion`. It analyzes pixel-level motion between two sequential images using a Farneback optical flow algorithm [3]. It is used to infer subtle camera movements that may be too small for full 3D reconstruction.
- `code`. This is the agent's primary computation engine. It generates and executes Python code within a sandbox environment. It tasks a set of context variables (*e.g.*, poses, points) and an natural language description (*e.g.*, request and description of the variables) as input. The code is generated using a knowledge-augmented strategy, where relevant geometric formulas are injected into the prompt, ensuring the computation is both deterministic and sound.

C.3. Agentic Framework

The GCA paradigm is implemented as a high-throughput, modular system. The core VLM deployment and the perception tool suite are physically decoupled to ensure scalability and robustness.

System Backend and State Management. The entire system is built using Ray [12] and LangGraph. LangGraph is used to define and manage the agent's state and orchestrate the two-stage, graph-based reasoning flow (*i.e.*, $\mathcal{F}_{\text{formalize}}$ followed by the $\mathcal{F}_{\text{compute}}$ loop).

Tool Suite and VFMs Deployment. The perceptual and computational tools (listed in Section C.2) are encapsulated as independent Ray Serve actors. This microservice architecture allows GCA to make concurrent perception requests (*e.g.*, running reconstruct and detect in parallel), enabling high parallelism and automatic scaling. This entire tool suite is deployed on 2 NVIDIA A100 GPUs.

VLM Roles and Deployment. In GCA, a single VLM fulfills the three distinct roles within the GCA paradigm:

- **Semantic Analyst.** In the $\mathcal{F}_{\text{formalize}}$ stage, it interprets the query and visual context to generate the formal $\mathcal{C}_{\text{task}}$.
- **Tool Orchestrator.** In the $\mathcal{F}_{\text{compute}}$ stage, it manages the ReAct-style tool call loop, resolves ambiguities, and generates natural language descriptions for the coder.
- **Coder.** It generates Python code for the `code` tool.

The VLM deployment is separate from the tool suite. For open-source models (*e.g.*, Qwen3-VL-Thinking [13]), we use vLLM [6] for efficient, high-throughput inference on 8 NVIDIA A100 GPUs. For closed-source models (*e.g.*, Gemini-2.5-Pro [2]), we access them via their standard commercial APIs. We employ different sampling parameters based on the VLM's role. For the semantic analyst and tool orchestrator roles, which require reasoning and flexibility, we use `TEMPERATURE=0.6` and `TOP_P=0.95`. For the coder role, which demands deterministic and reliable output, we set `TEMPERATURE=0.0`. All roles use `MAX_TOKENS=32768`.

D. Evaluation Benchmark Details

We evaluate GCA on multiple challenging spatial reasoning benchmarks. This section provides a detailed description of each benchmark and its subcategories, corresponding to the results presented in Table 1 of the main paper.

D.1. MMSI-Bench

MMSI-Bench [20] is a comprehensive benchmark designed to evaluate a VLM's ability to perform spatial reasoning by integrating information from multiple, distinct images. It is organized into four distinct subcategories:

- **PR. (Positional Relationship).** This subcategory evaluates the model's ability to understand the relative positions between different objects, cameras and semantic regions (*e.g.*, a kitchen) across multiple views.
- **Attr. (Attribute).** This subcategory evaluates the model's ability to identify object attributes related to spatial properties, such as geometric properties (*e.g.*, size, length) or visual characteristics (*e.g.*, shape).
- **Mot. (Motion).** This subcategory evaluates the model's ability to understand object or camera's movement.
- **MSR (Multi-Step Reasoning).** This subcategory evaluates the model's ability to perform complex reasoning by chaining multiple spatial understandings described above together to arrive at a final answer.

D.2. MindCube-tiny

MindCube-tiny is a subset of the MindCube benchmark [21], which is designed to test Spatial Mental Modeling (SMM). The core task evaluates a VLM's ability to construct and manipulate a 3D mental model of a scene using only a limited set of 2D images as input. The "tiny" version is a smaller-scale version of the full benchmark. We evaluate on its three primary sub-tasks:

- **Rot. (Rotation).** This task requires the model to infer the complete environment based on partial visual information, testing its understanding of sequential views and consistent spatial cues between images (*e.g.*, lighting).
- **Ard. (Around).** This task requires the model to infer the scene from a novel viewpoint, testing its ability to interpolate and extrapolate its mental 3D model.
- **Amg. (Among).** This task requires the model to infer the 3D spatial arrangement based on four orthogonal views characterized by significant occlusion, testing its ability to establish consistency relationships across perspectives and reason about the relative positions of unseen objects.

D.3. OmniSpatial

OmniSpatial [5] is a comprehensive benchmark designed to evaluate a broad spectrum of visual-based spatial intelligence capabilities in VLMs. The full benchmark consists

of four categories. We primarily focus on the two subcategories most closely related to geometric perception:

- **Pers. (Perspective Taking).** This subcategory assesses the model's understanding of 3D spatial relationships by adopting varied viewpoints, *e.g.*, egocentric, allocentric, and hypothetical perspective.
- **Dyn. (Dynamic Reasoning).** This subcategory assesses the model's understanding of object motion and judgments in uncertain or rapidly changing environments.

We exclude the other two subcategories: "Spatial Interaction" (which focuses on diagrams and user-interface, *e.g.*, terrain map) and "Complex Logic" (which involves abstract spatial reasoning, *e.g.*, puzzles), as they are more centered on abstract or symbolic reasoning rather than the high-fidelity geometric perception that GCA is designed to solve.

D.4. SPBench

SPBench [7] is a benchmark designed to evaluate both VLM's spatial reasoning in single view and multiple views:

- **SI (Single Image).** This subset contains questions that test the model's understanding and reasoning capabilities within a single image, including absolute distance, object size, relative distance, and relative direction.
- **MV (Multiple Views).** This subset requires the model to integrate information from multiple viewpoints to answer questions about relative position and object counts within a overlapping scene.

D.5. CV-Bench

CV-Bench is a visual-centric benchmark that evaluates the spatial understanding capabilities of VLMs. It is broadly composed of 2D and 3D reasoning tasks:

- **2D (2D Relationship).** This subcategory tests fundamental 2D spatial understanding, including 2D positional relationships and object counting.
- **3D (3D Relationship).** This subcategory assesses the model's grasp of 3D concepts, such as depth analysis and 3D distance comparisons between objects in the scene.

E. Prompts Used in GCA

We provide detailed prompts used in GCA, including task formalization (Table C and D), tool orchestration (Table E) and knowledge-augmented code generation (Table F and G). Besides, we also provide the in context examples used in the reference frame formalization (see Figure A and Table C).

F. Qualitative Case Study

We provide several qualitative case studies on how GCA effectively tackles spatial reasoning queries. These challenging cases includes unique object counting across multiple views (Figure B), direction-based reference frame (Figure C), object-based reference frame (Figure D), camera rotation analysis (Figure E), object movement analysis (Figure F), and metric-scale estimation (Figure G).

Figure A. **In Context Examples Used in Formalizing Reference Frame.** The output format follows the prompt in Table C.

G. Broader Impacts

Advancing Embodied AI Systems. Applications in robotics and AR/VR depend on an agent's ability to translate ambiguous human instructions (e.g., "sit on the sofa") into precise geometric actions. GCA provides a robust framework for this translation, potentially facilitating the development of robots and AR/VR interfaces that can interact with the physical world with high fidelity.

Trust, Interpretability, and Verification. Unlike opaque end-to-end spatial VLMs, GCA's reasoning process is highly traceable. $\mathcal{C}_{\text{task}}$ serves as an explicit, human-readable artifact that can be verified before a high-stakes action is executed. This verify-then-execute capability is critical for safety in real-world applications. Furthermore, the structured outputs from both the $\mathcal{F}_{\text{formalize}}$ and $\mathcal{F}_{\text{compute}}$ stages can serve as a valuable source of process-level supervision, acting as a reliable validator to guide the training of more efficient end-to-end spatial VLM.

Inheritance and Amplification of Bias. The reliance on VFMs for perception and geometry (*e.g.*, 3D reconstruction, object orientation) creates a new dependency chain for bias and failure. If these perceptual tools perform poorly on objects, scenes, or lighting conditions, GCA will not only

inherit these biases but may also amplify them, leading to incorrect outcomes in real-world interactions. In addition to future GCA upgrades to more powerful VFMs (such as from SAM to SAM-2), further exploration is required to leverage VLM's analytical capabilities to establish a more robust error correction mechanism, just as humans analyze failure cases.

Table C. **Prompts Used for Formalizing Reference Frame Constraint.** Here, {example} is the in-context examples (see Figure A), and {question} is the placeholder that will be replaced.

[CORE MISSION]
You are an expert spatial reasoning analyst. Your sole mission is to analyze a user's question and define the final **Reference Frame**. Your goal is to find the single element (an object, a camera, or a vector) that provides the ultimate, non-negotiable definition for absolute directions (e.g., North, South) or relative perspectives (e.g., "front", "left") in the final answer. Ask yourself: *"What element holds the final authority on what 'north', 'front', or 'left' means in the question?"*

[OUTPUT FORMAT]
Your response **MUST** be a single, valid JSON object:

```
```json
{
  "reasoning": "A brief, step-by-step logical deduction, explaining WHY this anchor
               is the arbiter of direction.",
  "formalization": "The precise mathematical mapping of a semantic direction to one
                   of the Solvable Geometric Primitive listed below, e.g.,
                   $-Z_cam0, +Z_toaster$."
}
```
```

[FORMALIZATION]
1. Identify the Final Arbiter of Direction
- **Priority 1: Absolute Direction.** If an absolute direction (North, South, etc.) is explicitly tied to an element, that element is the arbiter, overriding everything else.
- **Priority 2: Relative Query.** If no absolute direction is given, the arbiter is the object of the relative question.

2. Formalize Reference Frame Using Solvable Geometric Primitives: o construct a mathematical formalization of reference frame, you MUST use following three types of **Solvable Geometric Primitives**:
- **A. Camera Axes**: A vector from a specific camera's coordinate system. Format: $\pm X_{\text{cam[i]}}, \pm Y_{\text{cam[i]}}, \pm Z_{\text{cam[i]}}$. The camera coordinate system follows OpenCV convention: +Z points forward, +Y points down, and +X follow right-hand rules.
- **B. Object Axes**: A vector from a specific object's semantic coordinate system. Format: $\pm X_{\text{[obj]}}, \pm Y_{\text{[obj]}}, \pm Z_{\text{[obj]}}$. The object's local coordinate system is defined by: +Z points its semantic "front", +Y points its semantic "down", +X follows right hand rules, and origin at centroid.
- **C. Inter-Object Vector (Direction)**: A vector connecting the centroids of two concrete, detectable objects. Format: Centroid(B) − Centroid(A).

3. Semantic Formalization
- **A. Object-based Reference Frame**: Usually can be defined by corresponding object's axes. Examples:
  - "when using the toaster" suggests user's "forward" is opposite the toaster's semantic "front", i.e., $+Z_{\text{ref}} = -Z_{\text{toaster}}$.
  - You must choose a **Physical and Detectable** object as the object anchor. Don't use abstract concepts like room/region/area.
- **B. Camera-based Reference Frame**: Usually can be defined by corresponding camera's axes. Examples:
  - "from the perspective of Figure 1" suggests reference frame is identical to camera 0's, i.e., $+Z_{\text{ref}} = +Z_{\text{cam0}}$.
- **C. Direction-based Reference Frame**
  - For spatial relationship between two **Physical and Detectable Objects**, it can be defined by inter-object vector. Examples: "object A is north of object B" suggests the direction from object B to A is north, i.e., $+Z_{\text{ref}} = \vec{BA}$ = Centroid(A) − Centroid(B) = North
  - For spatial relationship between two **Abstract Concepts**, you must use a physic object's axes or a camera's axes as the proxy to tie this direction. Examples: "moves from room A to room B, facing north".Assume this motion aligns with moving from background towards the foreground, formalized as $+Z_{\text{ref}} = -Z_{\text{cam[i]}}$ = North.

[EXAMPLES]
{examples}

[QUESTION]
{question}

Now, please analyze the above question and provide your response in the specified JSON format.

Table D. **Prompts Used for Formalizing Objective Constraint.** Here, {question} is the placeholder that will be replaced.

[CORE MISSION]
You are an expert spatial reasoning analyst. Your sole mission is to analyze a user's question and define the final **Objective**. Your goal is to rephrase the user's natural-language question into a single and precise sentence. This sentence describes the specific value or piece of information that definitively answer the question. Ask yourself: *"What is the single, final piece of information (e.g., a scalar value, a 3D vector, a sequence of rotations) that the user is finding?"*

[OUTPUT FORMAT]
Your response **MUST** be a single, valid JSON object:

```
```json
{
  "reasoning": "A brief, step-by-step logical deduction that breaks down the user's
               question into its final objective.",
  "formalization": "A single, concise sentence that defines the ultimate goal of the
                   question, stated in technical terms."
}
```
```

[OBJECTIVE]
- **Identify the target variable**: What type of answer is being sought? Is it a distance, a speed, an orientation, a direction, a count, a relationship, or a sequence of actions?
- **Identify the Key Entities**: What are the specific, concrete objects, cameras, or locations involved in the question?
- **Synthesize the Objective**: Combine the target variable and entities into a single, unambiguous sentence. This sentence must be a statement or a noun phrase, not a question. Example:
  - Bad: Which way the object are going?
  - Good: The 3D direction vector of the object' movement.

[QUESTION]
{question}

Now, please analyze the above question and provide your response in the specified JSON format.

Table E. **Prompts Used for Tool Orchestration.** Here, {`api_documents`} is the detailed documentation of provided APIs. {`history`} includes the initial user question, task formalization, previous planning and corresponding execution results.

[CORE MISSION]
You are an expert spatial intelligence agent. Your mission is to generate a sequence of tool calls that rigorously computes the answer. It follows an iterative Plan → Update cycle, using a workspace as your computational memory.

- **Plan**: Decide the next tool calls based on the goal and current workspace.
- **Update**: Tool results are saved as new variables in the workspace.
- **Repeat**: Continue until the workspace contains enough information to conclude the final answer.

[AVAILABLE APIS]
{api_documents}

[A TYPICAL WORKFLOW]
1. Strictly Follow the Task Formalization

- **Task Formalization**: The input question is pre-formalized and consists of two parts: **Reference Frame Constraint** and **Objective Constraint**. You MUST strictly follow this formalization to solve the question.
- **Reference Frame**: Reference frame is the only one coordinate system that matters for interpreting the final answer (left/right, north/south, etc.). The "formalization" is the equation you must solve with the specified geometric tools. E.g., $+Z_{\text{ref}} = -Z_{\text{toaster}}$ indicates reference frame is defined by object toaster's local frame, so we MUST perform all calculations in toaster's frame.
- **Objective**: Objective is the ultimate goal of the question. You must calculate this objective within the reference frame.

2. Acquire Geometric Data: Based on user's question and pre-defined task formalization, plan the necessary tool calls to gather all data required for the final calculation. This involves two parallel goals:

- **A. Solve for the Reference Frame**: The formalization mathematically defines the **World-to-Reference Transformation**. To solve this formalization, your plan MUST gather all the geometric ingredients in the world frame.
  - "reconstruct" tool provides the 3D reconstruction context in a unified world frame, bridging the gap between input 2D images and geometric perception.
  - If formalization involves an object's axes (e.g., $+Z_{\text{ref}} = -Z_{\text{toaster}}$), call "predict_obj_pose" to solve that object's local frame. The resulting "T_obj2world" is required to establish the reference frame.
  - If formalization involves a camera's axes (e.g., $+Z_{\text{ref}} = -Z_{\text{cam0}}$), you must acquire reconstruction context (include extrinsic matrix) to implement the formalization.
  - If formalization involves a vector between two objects (e.g., Centroid(B) − Centroid(A) = North), your plan MUST include calls to "project_box_to_3d_points" for both object A and B.
- **B. Solve for the Objective**: Follow the objective to identify the target data that need to be analyzed within the reference frame.

3. Perform Final Calculation in Reference Frame: Once all required variables are available in the workspace, call "code".

4. Conclusion: Conclude the final answer using "generate_final_answer".

[OUTPUT FORMAT]
Your response **MUST** be a single, valid JSON object:

```
```json
{
  "analysis": "Briefly analyze how you will implement the formalization and what
               target data is need. State the immediate next tool(s) you will call",
  "tool_calls": [
    {
      "api": "API name", "args": {...},
      "output_variable": "A unique name for output, stored in the workspace"
    }, ...
  ]
}
```
```

[HISTORY]
Here is the history so far:
{history}

Please analyze current situation and history messages, and then generate your response. Your plan MUST only includes the immediate next one step.

Table F. **Prompts Used for Coder.** Here, {question}, {formalization}, {objective} and {var_docs} are the placeholder that will be replaced. {knowledge} includes a set of releveant, fixed formulas based on the type of input variables.

[CORE MISSION]
You are an expert Python programmer. Your goal is to write a single Python function that correctly implements the computational objective based on the provided context and documentation.

[User's Question]
This provides the high-level context for your task.
{question}

[Reference Frame]
All geometric data are defined in the world frame (defined by camera 0) unless specified. All final interpretations MUST be expressed in the reference frame. The reference frame is defined:
{formalization}

[Objective]
The ultimate goal from the high-level question. You MUST write code to calculate this objective to answer the question.
{objective}

[Documentation of Available Variables]
{var_docs}

[Additional Knowledge]
This information is always true for the environment your code runs in.
{knowledge}

[Available Libraries]
You can use "numpy", "torch", "scipy", "math", and other standard Python libraries.
[Critical Rules and Output Format]
1. Synthesize and Self-Correct: Your primary duty is to write correct code. Use the objective as your goal, but critically verify and implement the logic using the provided documentation.
2. Handle Multiple-Choice Questions: If the user's question is multiple-choice, your code MUST systematically evaluate the conditions for every option (e.g., A, B, C, D). Besides, the logic of your code should focus ONLY on the given options.
3. DO NOT add any explanation in your final output. Your output MUST follow this format:

```python
def execute(func_signature):
    # Your code here
    ...

    return serializable_value  # return value MUST be a serializable type
```

Table G. **Knowledge and Formulas Used in Knowledge-Augmented Code Generation.** We will inject the relevant knowledge based on the type of input variable.

[Output of "reconstruct"]
Extrinsic Transformation (World ↔ Camera):
- **World → Camera**: To transform a world point "P_world" into camera s's frame, use its extrinsic matrix `E_s = extrinsic[s]`. The formula is: `P_cam_homo = P_world_homo @ E_s.T`.
- **Camera → World**: The pose of camera "s" in the world, "Pose_s", is the inverse of its extrinsic matrix: `Pose_s = np.linalg.inv(extrinsic[s])`. To transform a point from camera s's local frame to the world frame, use the formula: `P_world_homo = P_cam_homo @ Pose_s.T`.
- **Relative Rotation Analysis (Camera → Camera)**: To describe the rotation of camera j's pose relative to camera i's pose, use the camera poses in the world frame ("Pose_i", "Pose_j"). The relative rotation from i to j is: `R_rel = R_j_pose @ R_i_pose.T`, which simplifies to `R_rel = (R_j.T) @ (R_i.T).T = R_j.T @ R_i`.

[Output of "predict_obj_pose"]
Object Pose Transformation (World ↔ Object):
- **Object → World**: The "T_obj2world" matrix (aliased as "Pose_obj") transforms points from the object's local frame to the world frame using the formula: `P_world_homo = P_local_homo @ Pose_obj.T`.
- **World → Object**: To transform "P_world" into the object's local frame, use the inverse matrix: `T_world_to_obj = np.linalg.inv(Pose_obj)`. The formula is: `P_local_homo = P_world_homo @ T_world_to_obj.T`.

[Output of "reconstruct", "predict_obj_pose"]
Interpreting Rotation:
- **General Rotation Direction**: For most questions about rotation direction, convert the relative rotation matrix to a rotation vector `[rx, ry, rz]` via `scipy.spatial.transform.Rotation.from_matrix(R).as_rotvec()`.
  - The component with the largest absolute value indicates the primary axis of rotation.
  - Based on the OpenCV coordinate system (+X right, +Y down, +Z forward) and the right-hand rule: `ry > 0` corresponds to a pan to the right, `rx > 0` corresponds to a tilt upward, `rz > 0` corresponds to a clockwise roll.
- **Sequential Rotations**: Use `scipy.spatial.transform.Rotation.from_matrix(R).as_euler(order)` to compute sequential rotation. Verify the signs of the resulting angles. It must match the option's description.

[If formalization includes cardinal direction]
1. Identify the Cardinal Anchor Axis from the formalization.
  - The formalization string (e.g., $+Z_{\text{ref}} = -Z_{\text{obj}} = \text{South}$) links one of your reference axes to a cardinal direction.
  - Parse this string to find the anchor. In the example, the anchor is South, and it corresponds to the $Z_{\text{ref\_axis}}$. This defines your first cardinal vector: $\text{South\_axis} = Z_{\text{ref\_axis}}$.

2. Derive the Complete Set of Cardinal Axes: You must find the remaining cardinal axes via applying cross product:
  - If South_axis is known, starting with `West_axis = np.cross(Y_ref_axis, South_axis)`.
  - If West_axis is known, starting with `North_axis = np.cross(Y_ref_axis, West_axis)`.
  - If North_axis is known, starting with `East_axis = np.cross(Y_ref_axis, North_axis)`.
  - If East_axis is known, starting with `South_axis = np.cross(Y_ref_axis, East_axis)`.

3. Project and Determine the Final Quadrant: Project the target vector onto the primary horizontal cardinal axes (North and East).
  - `projection_north = np.dot(disp_vec_world, cardinal_map["N"])`
  - `projection_east = np.dot(disp_vec_world, cardinal_map["E"])`
  - Use the signs of these projections to determine the final answer.

[Output of "detect"]
Bounding Box Format: All bounding box in input variable is provided in the `[x1, y1, x2, y2]` format.

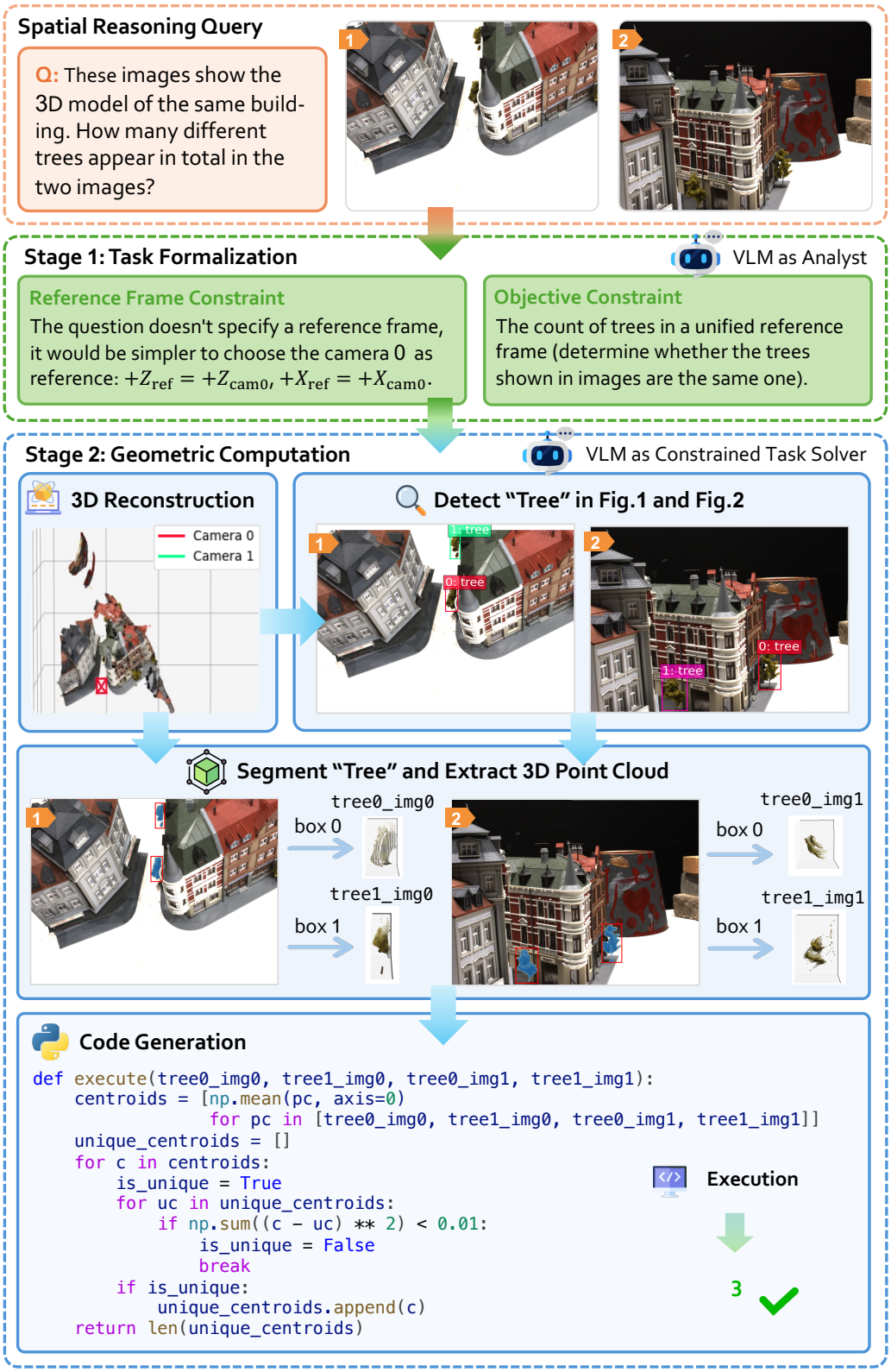


Figure B. **Case Study #1.** Unique object counting across multiple views.

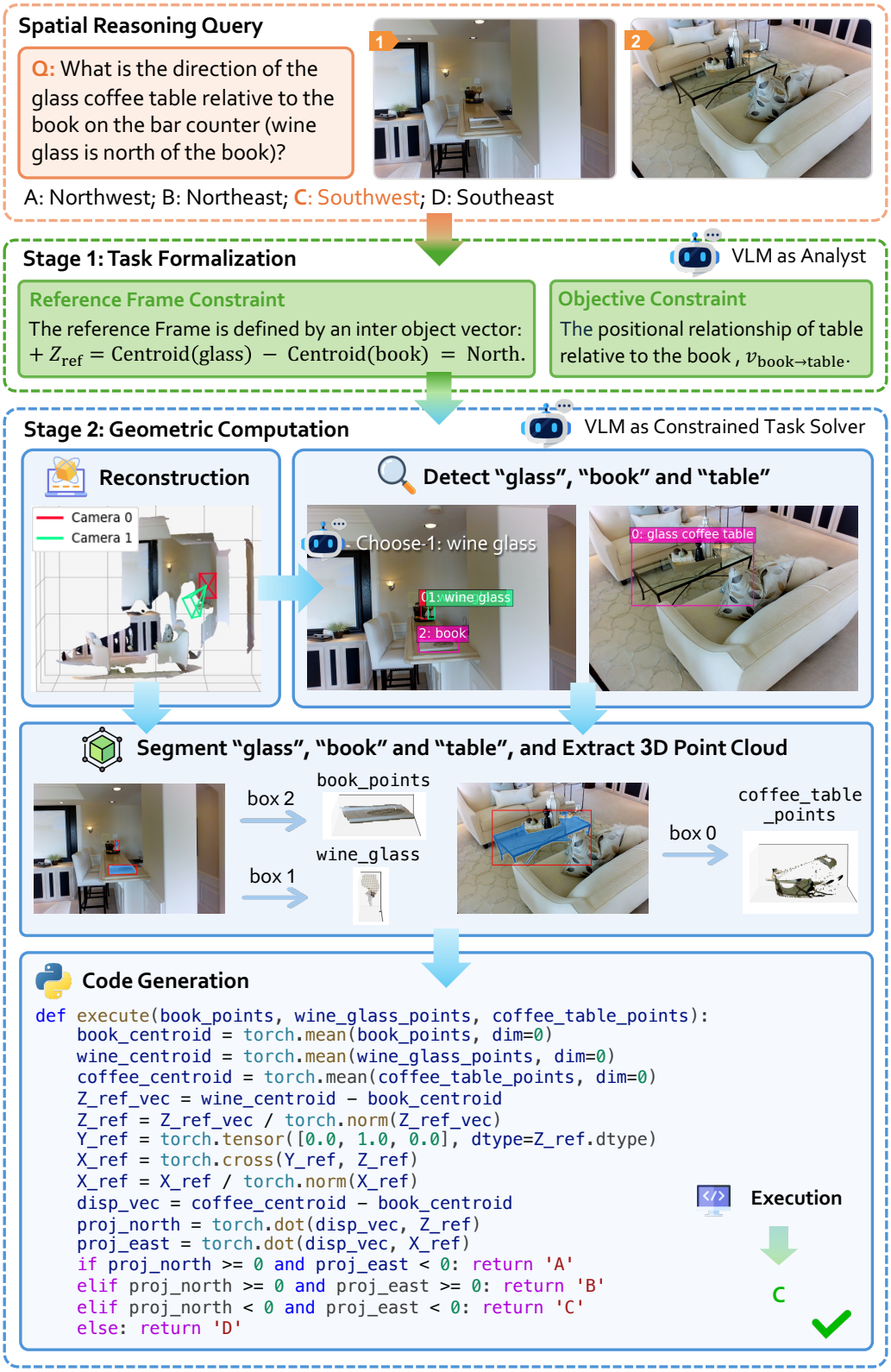


Figure C. **Case Study #2.** Direction-based reference frame.

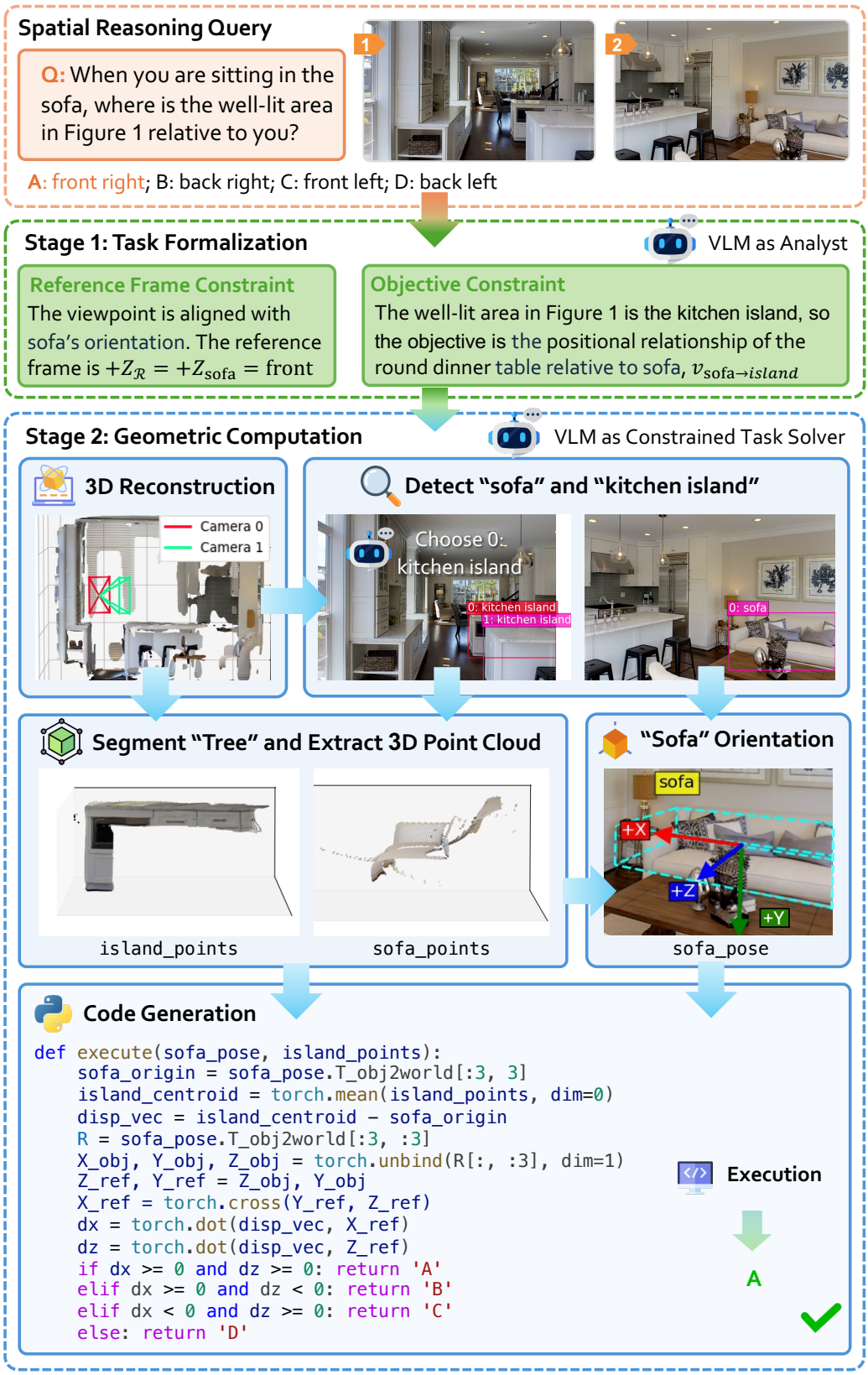
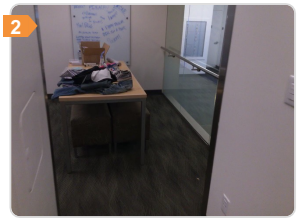


Figure D. **Case Study #3.** Object-based reference frame

Spatial Reasoning Query

Q: The camera coordinate system is defined as +Y up, -Z forward, and is a right-handed system. How can the viewpoint in the second image be obtained from the viewpoint in the first image in its corresponding camera coordinate system?



A: Rotate a positive angle around the Z axis, then a positive angle around the Y axis
B: Rotate a positive angle around the Y axis, then a negative angle around the X axis
C: Rotate a positive angle around the Y axis, then a positive angle around the X axis
D: Rotate a positive angle around the Z axis, then a negative angle around the Y axis

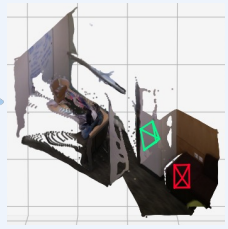
Stage 1: Task Formalization — VLM as Analyst

Reference Frame Constraint
The question asks for the viewpoint transformation, establishing cam_0 as the anchor. The custom coordinate system (+Y up, -Z forward) must be mapped to cam_0: $+Z_{\text{ref}} = -Z_{\text{cam0}}$, $+Y_{\text{ref}} = -Y_{\text{cam0}}$.

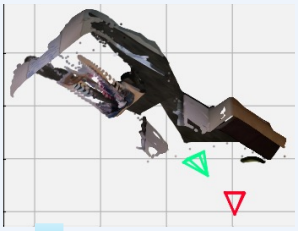
Objective Constraint
The sequence of axis rotations required to transform the camera viewpoint from the first image to the second image.

Stage 2: Geometric Computation — VLM as Constrained Task Solver

From Camera 0



BEV Perspective



Reconstruction

extrinsic

Code Generation

```python
def execute(extrinsic):
    R_0, R_1 = extrinsic[0][:3, :3], extrinsic[1][:3, :3]
    T = np.diag([1, -1, -1])  # y_user = -y_cam0, z_user = -z_cam0
    R_user = T @ R_1.T @ R_0 @ T

    options = {
        'A': {'order': 'zyx', 'signs': [1, 1]}, 'B': {'order': 'yxz', 'signs': [1, -1]},
        'C': {'order': 'yxz', 'signs': [1, 1]}, 'D': {'order': 'zyx', 'signs': [1, -1]}
    }
    for opt, params in options.items():
        angles = Rotation.from_matrix(R_user).as_euler(params['order'], degrees=False)
        sign1_match = (params['signs'][0] == 1 and angles[0] > 1e-6 ) or \
                      (params['signs'][0] == -1 and angles[0] < -1e-6 )
        sign2_match = (params['signs'][1] == 1 and angles[1] > 1e-6 ) or \
                      (params['signs'][1] == -1 and angles[1] < -1e-6 )
        if sign1_match and sign2_match:
            return opt
        return None
```

Execution

C

Figure E. **Case Study #4.** Camera rotation analysis.

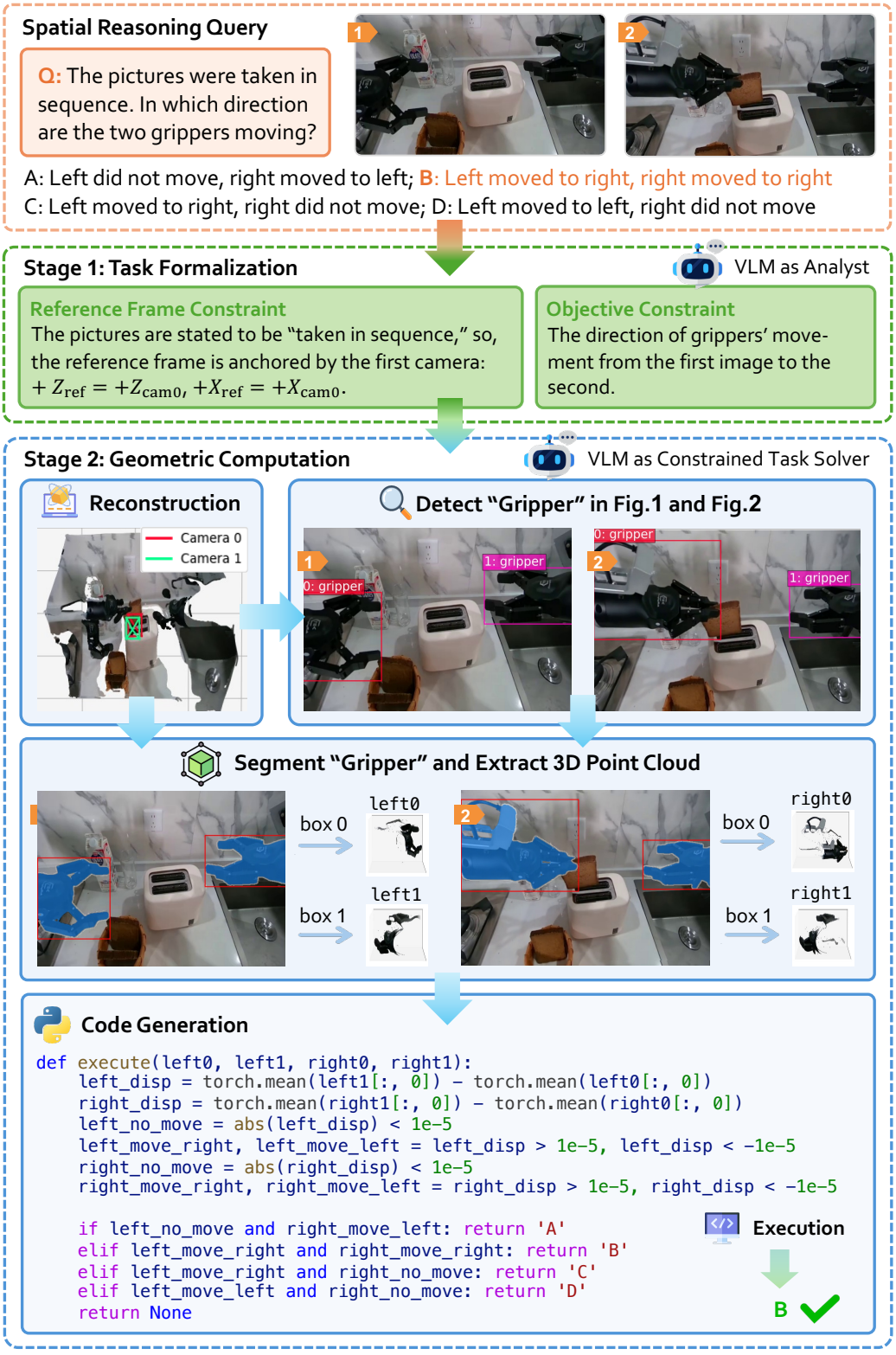


Figure F. **Case Study #5.** Object movement analysis.

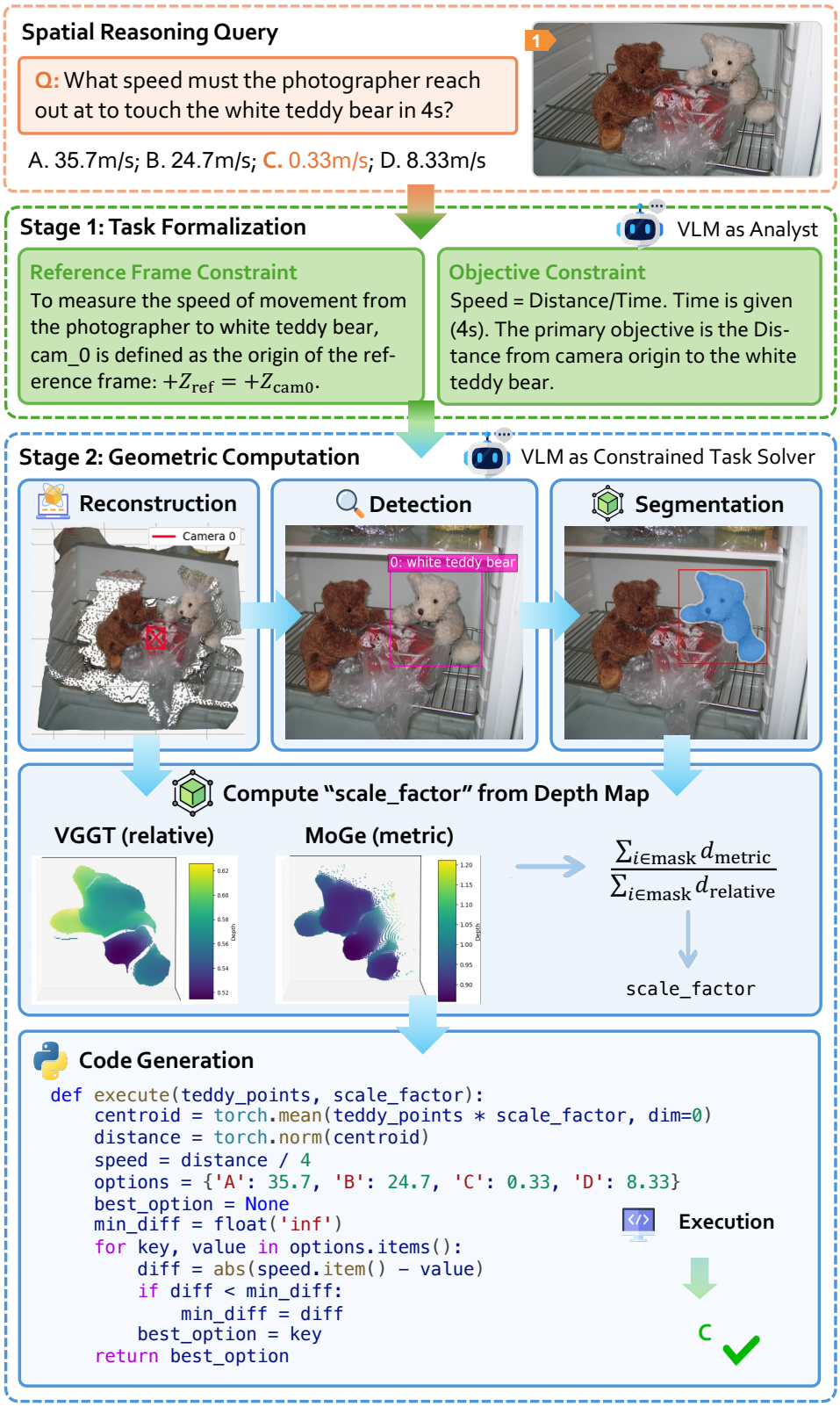


Figure G. **Case Study #6.** Metric-scale estimation.

References

[1] Jae-Woo Choi, Youngwoo Yoon, Hyobin Ong, Jaehong Kim, and Minsu Jang. Lota-bench: Benchmarking language-oriented task planners for embodied agents. In *The Twelfth International Conference on Learning Representations*, 2024. 1

[2] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. 4

[3] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003. 4

[4] Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In *Conference on Robot Learning*, pages 4573–4602. PMLR, 2025. 1

[5] Mengdi Jia, Zekun Qi, Shaochen Zhang, Wenyao Zhang, Xinqiang Yu, Jiawei He, He Wang, and Li Yi. Omnispatial: Towards comprehensive spatial reasoning benchmark for vision language models. *arXiv preprint arXiv:2506.03135*, 2025. 1, 4

[6] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023. 4

[7] Hongxing Li, Dingming Li, Zixuan Wang, Yuchen Yan, Hang Wu, Wenqi Zhang, Yongliang Shen, Weiming Lu, Jun Xiao, and Yueting Zhuang. Spatialladder: Progressive training for spatial reasoning in vision-language models. *arXiv preprint arXiv:2510.08531*, 2025. 1, 5

[8] Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Erran Li Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37: 100428–100534, 2024. 1

[9] Zefu Lin, Rongxu Cui, Chen Hanning, Xiangyu Wang, Junjia Xu, Xiaojuan Jin, Chen Wenbo, Hui Zhou, Lue Fan, Wenling Li, et al. Embodiedcoder: Parameterized embodied mobile manipulation via modern coding model. *arXiv preprint arXiv:2510.06207*, 2025. 1

[10] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European conference on computer vision*, pages 38–55. Springer, 2024. 3

[11] Xiaoya Lu, Zeren Chen, Xuhao Hu, Yijin Zhou, Weichen Zhang, Dongrui Liu, Lu Sheng, and Jing Shao. Is-bench: Evaluating interactive safety of vlm-driven embodied agents in daily household tasks. *arXiv preprint arXiv:2506.16402*, 2025. 1

[12] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018. 4

[13] QwenTeam. Qwen3-vl: Sharper vision, deeper thought, broader action. https://qwen.ai/blog?id=99f0335c4ad9ff6153e517418d48535ab6d8afef&from=research.latest-advancements-list, 2025. 2, 3, 4

[14] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. Sam 2: Segment anything in images and videos. In *The Thirteenth International Conference on Learning Representations*, 2024. 3

[15] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 5294–5306, 2025. 3

[16] Ruicheng Wang, Sicheng Xu, Yue Dong, Yu Deng, Jianfeng Xiang, Zelong Lv, Guangzhong Sun, Xin Tong, and Jiaolong Yang. Moge-2: Accurate monocular geometry with metric scale and sharp details. *arXiv preprint arXiv:2507.02546*, 2025. 3

[17] Zehan Wang, Ziang Zhang, Tianyu Pang, Chao Du, Hengshuang Zhao, and Zhou Zhao. Orient anything: Learning robust object orientation estimation from rendering 3d models. In *Forty-second International Conference on Machine Learning*, 2024. 3

[18] Jihan Yang, Shusheng Yang, Anjali W Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. Thinking in space: How multimodal large language models see, remember, and recall spaces. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 10632–10643, 2025. 1, 2

[19] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. In *Forty-second International Conference on Machine Learning*, 2025. 1

[20] Sihan Yang, Runsen Xu, Yiman Xie, Sizhe Yang, Mo Li, Jingli Lin, Chenming Zhu, Xiaochen Chen, Haodong Duan, Xiangyu Yue, et al. Mmsi-bench: A benchmark for multi-image spatial intelligence. *arXiv preprint arXiv:2505.23764*, 2025. 1, 4

[21] Baiqiao Yin, Qineng Wang, Pingyue Zhang, Jianshu Zhang, Kangrui Wang, Zihan Wang, Jieyu Zhang, Keshigeyan Chandrasegaran, Han Liu, Ranjay Krishna, et al. Spatial mental modeling from limited views. In *Structural Priors for Vision Workshop at ICCV'25*, 2025. 1, 4

[22] Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, et al. Vlabench: A large-scale benchmark

for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11142–11152, 2025. 1

[23] Enshen Zhou, Jingkun An, Cheng Chi, Yi Han, Shanyu Rong, Chi Zhang, Pengwei Wang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, et al. Roborefer: Towards spatial referring with reasoning in vision-language models for robotics. *Advances in Neural Information Processing Systems*, 2025. 1