3D Primitives are a Spatial Language for VLMs

Anonymous authors
Paper under double-blind review

Abstract

Vision-language models (VLMs) exhibit a striking paradox: they can generate executable code that reconstructs a 3D scene from geometric primitives with correct object counts, classes, and approximate positions, yet the same models fail at simpler spatial tasks when directly answering natural-language questions on the same image. We show that 3D geometric primitives (cubes, spheres, cylinders, expressed in executable code) serve as a powerful intermediate representation for spatial understanding, and we exploit this insight through three contributions. First, we introduce **SpatialBabel**, a benchmark that evaluates fourteen VLMs on primitive-based 3D scene reconstruction across six *scene-code languages* (programming languages and declarative formats for expressing 3D primitive scenes), revealing that a model's object-detection F1 on identical images swings sharply with the target scene-code language: aggregated with class- and scale-fidelity into our Reconstruct Score, the best-versus-worst-language gap reaches 0.40 (on a 0–1 scale) within a single model. Second, we propose **Code-CoT** (Code Chain-of-Thought), a training-free inference strategy that routes spatial reasoning through primitive-based code generation. Code-CoT improves the SpatialBabel-QA-Score by up to +8.9% on primitive scenes and lifts real-photo CV-Bench-3D accuracy by up to +5.9% for VLMs with strong coding capabilities. Third, we propose **S^3-FT** (Self-Supervised Spatial Fine-Tuning), which self-supervisedly distills primitive spatial knowledge into general visual reasoning by parsing the model's own primitive-reconstruction code (generated in Three.js) into structured annotations and fine-tuning on the result, requiring *no human labels and no teacher model*. Training on primitive images alone, S^3-FT improves Qwen3-VL-8B by +5.5 to +10.3% on SpatialBabel-Primitive-QA, +9.7% on CV-Bench-2D, and +17% on HallusionBench; the recipe transfers across model families. These results establish geometric primitives in executable code as both a diagnostic tool for VLM spatial understanding and a transferable spatial vocabulary. We will release the benchmark, training data, checkpoints, and evaluation toolkit upon publication.

1 Introduction

Vision-language models (VLMs) possess more spatial knowledge than their standard outputs reveal. Show a VLM a scene of geometric primitives (a cube and a cylinder) and ask "how many objects?"; it may hallucinate, answering "three" or "four." Yet ask the same model to write scene-reconstruction code (e.g., Three.js) from the same image, and it produces a program that places the correct number of primitives at roughly the correct 3D coordinates (Figure 1A). The same model, given the same pixels, contradicts itself. Across both proprietary APIs and open-weights models, VLMs generate compilable, spatially faithful 3D primitive-reconstruction programs while simultaneously failing to count, localize, or compose spatial relations when directly answering natural-language questions.

Geometric primitives are uniquely suited for diagnosing this gap: procedurally generated scenes have zero label noise, a no-occlusion constraint removes visibility ambiguity, and simple shapes eliminate object-recognition difficulty, so any failure on these scenes is a failure of spatial reasoning. Primitive-reconstruction code is also deterministically parseable into scene graphs, enabling automatic evaluation without human annotation.

This paradox points to a *task-routing failure*: scene-code generation is a strict superset of direct QA, as every visible object must be classified, sized, positioned, and named in a single coherent program. So, a model that

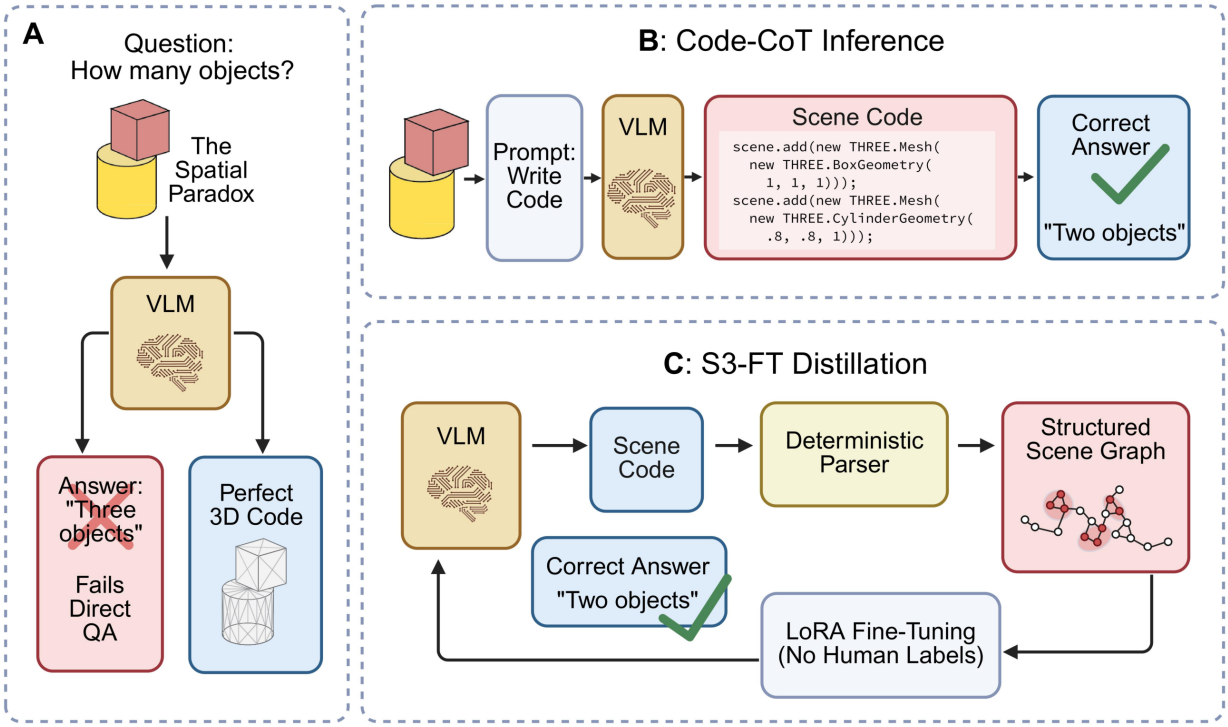

Figure 1: Geometric primitives as a spatial language for VLMs: the paradox and two approaches to exploit it. **(A) The paradox:** A VLM produces a faithful 3D primitive-reconstruction of the input image in code, yet fails simple spatial questions on the same scene **(B) Code-CoT** (Section 4) closes the gap *at inference time*: the model first reconstructs the scene as primitive-based executable code, then reasons over its own structured output. **(C) S^3-FT** (Section 5) closes the gap *at training time*: the model's own primitive-reconstruction outputs are deterministically parsed into spatial annotations, which serve as self-supervised labels for fine-tuning, requiring no human labels. Training on primitives alone yields gains on real-photo benchmarks.

emits accurate scene code demonstrably possesses the spatial understanding needed to answer the simpler question, yet fails to transfer it to natural-language QA.

We propose a methodology to test this hypothesis, quantify the phenomenon, and close the gap. Our investigation yields four progressive findings:

Finding 1: A multi-scene-code-language benchmark surfaces the routing failure (Section 3). We introduce SpatialBabel, a benchmark that evaluates VLM spatial understanding through 3D scene reconstruction across six scene-code languages (programming languages and declarative formats for expressing 3D primitive scenes) and a complementary spatial-QA task. Reconstruction quality varies sharply with the scene-code language, and each model exhibits a distinct pattern.

Finding 2: Primitive-based Code-CoT is a useful inference scaffold for capable VLMs (Section 4). We introduce Code-CoT (Code Chain-of-Thought): the model first reconstructs the scene as primitive-based executable code, then answers spatial questions conditioned on its own reconstruction. Code-CoT lifts accuracy of the spatial-QA task derived from primitive scenes for mid-tier proprietary APIs and lifts real-photo CV-Bench-3D Tong et al. (2024) accuracy for VLMs with strong coding capability.

Finding 3: 3D geometric primitives serve as a compositional spatial vocabulary for VLMs (Section 4.3). The Code-CoT lifts on primitive scenes and CV-Bench-3D suggest that VLM code training has implicitly tokenized 3D space through geometric primitive shapes. Unlike coordinate-based scaffolds, 3D

primitives encode shape, scale, and orientation in a single structured representation that aligns with how VLMs are trained to write 3D scene code.

Finding 4: Self-supervised fine-tuning on primitive scenes distills spatial knowledge to general vision (Section 5). S^3-FT (Self-Supervised Spatial Fine-Tuning) parses each VLM's own Three.js primitive-reconstruction code into structured spatial annotations and fine-tunes on the resulting multi-task mixture, requiring *no human labels and no teacher model*. Training on primitive images alone yields gains on real-photo benchmarks; the recipe transfers across five base VLMs.

We will release the benchmark, evaluation code, S^3-FT training datasets, and checkpoint adapters upon publication.

2 Related Work

Spatial understanding in VLMs. Prior work investigates VLM 3D reasoning through targeted training (SpatialVLM Chen et al. (2024), SpatialRGPT Cheng et al. (2024), SpatialLadder Li et al. (2025)) or single-image probing (Theory of Space Zhang et al. (2026)); mechanistic analyses identify structural causes of VLM spatial limitations including vision-token-norm suppression of positional encoding Qi et al. (2025), 1D flattening of 2D features Alam et al. (2026), attention imbalance Chen et al. (2025), and rigid linear spatial bindings Kang et al. (2026). SpatialBabel complements these by evaluating through *generative primitive-reconstruction code*, a stricter task than qualitative QA that externalizes the model's spatial representation into a parseable form.

3D scene generation and spatial-reasoning benchmarks. Scene-code generation methods (Scene Language Zhang et al. (2025), SceneScript Avetisyan et al. (2024), SceneCraft Hu et al. (2024), BlenderGym Gu et al. (2025), Design2Code Si et al. (2025)) and spatial-reasoning benchmarks (SpatialBench Xu et al. (2025), VSI-Bench Yang et al. (2025), 3DSRBench Ma et al. (2025), the spatial slices of MMMU Yue et al. (2024) / MathVista Lu et al. (2023)) each fix a single scene-code language or output channel, conflating spatial competence with format familiarity; code-generation benchmarks (HumanEval Chen et al. (2021), MBPP Austin et al. (2021), SWE-Bench Jimenez et al. (2023)) evaluate programming in isolation from spatial reasoning. SpatialBabel disentangles the two by holding spatial content constant while varying the scene-code language across six formats.

Primitives for spatial reasoning. Concurrent work by Lu et al. (2026) (DeepSeek-AI) also uses "primitives" for VLM spatial reasoning, but with a different definition: it interleaves *2D points and bounding boxes* (in pixel coordinates) into the chain of thought as referencing markers, while SpatialBabel uses *3D geometric primitives* as compositional scene elements with 3D position, rotation, and scale, expressed in executable code. 2D markers describe *where* an object sits in the image plane; 3D primitives additionally encode *what shape* and *how oriented*. Primitives play three roles in our work: the diagnostic stimulus of SpatialBabel, the inference-time reasoning scaffold of Code-CoT, and the supervisory signal of S^3-FT.

3 The SpatialBabel Benchmark

Existing benchmarks evaluate VLM spatial reasoning through a single output channel. Most of them ask the model to answer natural-language questions about a scene, and a few ask it to write code in one fixed scene-code language. Both choices mix two different factors: whether the model truly understands the spatial structure of the scene, and whether it is familiar with the output format it is asked to use. For example, a model may place objects correctly in Three.js but fail in Open3D, even when the scene is the same. To tell these two factors apart, SpatialBabel evaluates each scene with two complementary tasks: 3D scene reconstruction in six scene-code languages, and natural-language spatial question answering. This design lets us measure spatial understanding without tying it to any single output format.

3.1 Scene domains, tasks, and scene-code languages

Two scene domains with distinct diagnostic roles. The benchmark spans two complementary scene domains, chosen to enable systematic study of primitive-to-complex transfer (Figure 2). The *primitive* split

Figure 2: SpatialBabel's two scene domains. **Top:** four primitive scenes (T1, T2, T3, T5) from the 1,000-scene procedural split (fixed 10-color palette, no-major-occlusion placement). **Bottom:** four Hypersim photo-realistic indoor scenes (median 26 ground-truth objects per scene).

(1,000 scenes) contains procedurally-generated Blender scenes with 3–20 geometric primitives (cubes, spheres, cylinders, cones, tori, pyramids) organized into five complexity tiers (T1 = 3 objects through T5 = 20). Colors are drawn from a fixed 10-color palette, and a no-major-occlusion placement constraint ensures that every object is clearly visible and identifiable. Primitives serve as the *primary diagnostic tool* for three reasons: (i) procedural generation with known object placements provides exact ground truth with zero label noise; (ii) the no-occlusion constraint eliminates visibility ambiguity, so that any spatial failure is purely a reasoning failure rather than a perception one; (iii) primitive-reconstruction code can be deterministically parsed into structured scene graphs, enabling fully automatic evaluation. The *Hypersim* split (405 scenes, 200 evaluated) contains photo-realistic indoor scenes from Roberts et al. (2021), from which we extract per-object ground truth from the original 3D meshes. Hypersim serves as the *harder validation domain*: 60 object classes, complex geometries, and substantial inter-object occlusion. We evaluate on a representative 200-scene subset stratified across scene complexity levels to balance evaluation cost with statistical coverage. All renders are 512×512 from a single frontal viewpoint.

Tasks. SpatialBabel exposes two complementary task families derived from the same underlying scenes. The primary diagnostic is *cross-scene-code-language reconstruction* (**Reconstruct**): given a single-view image, the model must emit executable code in a specified scene-code language that reconstructs the depicted scene. The secondary task family, *programmatic spatial question answering* (**QA**), provides natural-language questions drawn from four categories: relationship, counting, existence, and comparison. Directional relationship questions (left/right, above/below, in front of/behind, and nearer/farther) are defined relative to the single fixed frontal viewpoint from which all scenes are rendered, which removes viewpoint ambiguity; the matching coordinate convention is supplied to the model in the prompt (Appendix A.1). Each task can be evaluated on either scene domain, yielding the four settings: SpatialBabel-Primitive-Reconstruct, SpatialBabel-Primitive-QA, SpatialBabel-Hypersim-Reconstruct, and SpatialBabel-Hypersim-QA.

Scene-code languages. We evaluate reconstruction across six scene-code languages spanning different programming paradigms and varying levels of pre-training prevalence: Three.js (imperative web 3D via JavaScript), Unity C# (game-engine API), Blender Python (procedural scripting), Open3D Python (mesh-creation API), Canonical JSON (a declarative schema specifying `position/rotation/scale/class_name`, introduced by this work), and a Scene Language DSL Zhang et al. (2025). The multi-scene-code design is the key diagnostic lever: a scene-code-agnostic spatial representation should yield roughly invariant performance across scene-code languages, whereas spatial knowledge entangled with a particular code-generation prior should produce model-specific cross-scene-code gaps. Pre-training exposure proxies for each scene-code language are reported in Appendix B.

3.2 Evaluation protocol and metrics

For each (model, scene-code language) pair, the generated code is deterministically parsed into a structured scene representation consisting of geometric primitives. For Hypersim scenes, whose ground-truth objects are real furniture and fixtures rather than simple shapes, each object is represented by the axis-aligned bounding box that encloses it, recording that box's center as the object position and its side lengths as the object scale; predictions and ground truth are therefore compared at the level of these box approximations. Predicted objects are matched one-to-one to ground-truth objects via the Hungarian algorithm Kuhn (1955), minimizing Euclidean position distance. Let O and $\hat{O}$ denote the ground-truth and predicted object sets, $\mathcal{M}$ the set of matched pairs, and e the per-scene spatial extent (the largest side of the ground-truth bounding box). Each (model, scene-code language) cell is scored on four $[0, 1]$ components:

- *Parse rate*: fraction of outputs that parse to a `Scene` with at least one object.
- *Detection F1*: a matched pair $(i, j) \in \mathcal{M}$ is a true positive at tolerance τ iff the normalized position error $\|p_i - \hat{p}_j\|/e \leq \tau$. With TP_τ such pairs, FP_τ unmatched predictions, and FN_τ unmatched ground-truth objects, $\mathrm{F1}_\tau = 2\,\mathrm{TP}_\tau/(2\,\mathrm{TP}_\tau + \mathrm{FP}_\tau + \mathrm{FN}_\tau)$; we report $\overline{\mathrm{F1}} = \frac{1}{|T|}\sum_{\tau \in T} \mathrm{F1}_\tau$ averaged over a tolerance sweep $T = \{0.05, 0.10, \ldots, 0.50\}$, a localization-aware analog of COCO average precision Lin et al. (2014). Detection F1 thus jointly rewards predicting the right number of objects *and* placing them correctly.
- *Class accuracy*: $\big|\{(i, j) \in \mathcal{M} : \mathrm{class}(i) = \mathrm{class}(j)\}\big|/|O|$, synonym-aware and normalized by the ground-truth count so omitted objects are penalized.
- *Scale fidelity*: $\frac{1}{|O|}\sum_{(i,j)\in\mathcal{M}} \max\big(0,\, 1 - \min(d^{ij}_{\mathrm{scale}}, 1)\big)$, where d^{ij}_{scale} is the root-mean-square error between predicted and ground-truth (s_x, s_y, s_z) scale vectors, likewise normalized by $|O|$.

These four components are aggregated into the *SpatialBabel-Reconstruct-Score*:

$$\text{Reconstruct Score} = \tfrac{1}{4}\big(\text{Parse Rate} + \text{Detection F1} + \text{Class Accuracy} + \text{Scale Fidelity}\big) \in [0, 1]. \quad (1)$$

Localization enters through detection F1 rather than as a separate component: a predicted object earns credit only when placed within tolerance of a ground-truth object, and class and scale credit are normalized by the ground-truth object count, so that neither omitting objects nor mislocating them is rewarded.

We evaluate 14 VLMs (8 proprietary APIs and 6 open-weights models) under two prompting modes (Direct and natural-language chain-of-thought). Per-scene-code Reconstruct Scores on the primitive split (SpatialBabel-Primitive-Reconstruct) are reported in Table 1, with per-model sample counts and prompting details in Appendix A.

3.3 Analysis

Table 1 reports per-scene-code Reconstruct Scores for fourteen VLMs on primitive scenes. *No model performs consistently across scene-code languages*: max–min gaps within a single model range from 0.09 (GPT-4o) to 0.40 (Qwen2.5-VL-3B). The best/worst language is itself model-specific, so spatial knowledge is entangled with each model's scene-code-specific code-generation prior. The gaps are largest on the clean primitive domain; on photo-realistic Hypersim scenes the mean Reconstruct Score falls by up to 0.25 and the cross-language gaps persist but shrink as the harder scenes depress reconstruction in every scene-code language (Appendix C).

Reconstruction skill correlates with QA across the capability range, but the two decouple among the strongest models. Across all fourteen VLMs, the mean Reconstruct Score (0.28–0.57) and relationship-category QA

Table 1: Per-scene-code SpatialBabel-Reconstruct-Score on primitive scenes (SpatialBabel-Primitive-Reconstruct) for fourteen VLMs under *Direct* prompting. Each cell is a $[0, 1]$ aggregate of parse rate, detection F1, class accuracy, and scale fidelity (Section 3.2). **Bold**: best per row. Red : lowest per row. Coding languages are ordered by approximate pre-training exposure (most → least; proxies in Appendix B). $\Delta_{\text{max-min}}$: max–min Reconstruct-Score gap within a row. Results under natural-language chain-of-thought prompting are reported in the per-component breakdown (Appendix Table 7).

Model	Three.js	Unity	Blen.Py	Open3D	JSON	DSL	$\Delta_{\text{max-min}}$
Proprietary APIs							
Claude Opus 4.7	**.662**	.541	.598	.566	.591	.474	.188
Claude Sonnet 4.6	.517	.547	.581	.581	**.587**	.349	.238
GPT-5	.484	.302	.445	.386	**.527**	.514	.224
GPT-4o	**.581**	.499	.514	.531	.565	.490	.091
Gemini-2.5-Pro	.576	.322	.430	.557	**.589**	.425	.267
Gemini-2.5-Flash	.487	.390	.508	.523	**.636**	.275	.361
Gemini-3-Flash	.501	.393	.495	.554	**.668**	.525	.275
Gemini-3-Pro	**.715**	.525	.571	.537	.688	.400	.315
Open weights							
Qwen3-VL-8B	.477	.504	.310	.398	**.541**	.483	.231
Qwen2.5-VL-32B	.497	.502	.283	**.551**	.461	.329	.268
Qwen2.5-VL-7B	.392	.189	.114	.382	**.459**	.137	.345
Qwen2.5-VL-3B	.327	.461	**.496**	.103	.429	.095	.401
LLaVA-OV-7B	.485	.517	.503	.438	.227	**.537**	.309
InternVL3-8B	.335	.392	.494	.388	.462	**.501**	.165

accuracy (30.9–38.2%) are moderately correlated (Pearson $r = 0.48$, Spearman $\rho = 0.58$; overall QA similar, $r = 0.50$), reflecting that more capable models tend to be better at both tasks. The association is driven by the weaker open-weights models; among the most capable models QA-relationship saturates within a 2.5-pp band (35.7–38.2%) while the Reconstruct Score still varies by more than 2×, so stronger reconstruction does not translate into higher QA at the top end (per-model breakdown in Appendix Table 8). The per-model rankings motivate the two interventions that follow: a per-model best/worst scaffold for inference-time Code-CoT (Section 4) and a structured pseudo-ground-truth signal for self-supervised training (Section 5).

4 Code-CoT: Primitive Reconstruction as a Generic Spatial Reasoning Tool

We exploit the model's spatial knowledge pathway of primitive-reconstruction with Code Chain-of-Thought (Code-CoT), a training-free strategy where the model first emits primitive-based scene code and then answers spatial questions conditioned on it, across three domains: SpatialBabel primitives, Hypersim photo-realistic indoors, and CV-Bench-3D real photos.

4.1 Setup

We compare four inference modes on the same SpatialBabel-QA prompts: (i) **direct** (image + question, no reasoning instruction); (ii) **NL-CoT**, a natural-language chain-of-thought scaffold; (iii) **best-language Code-CoT**, which reconstructs the scene in the model's primitive-best scene-code language (Table 1) before answering; (iv) **worst-language Code-CoT**, same procedure with the primitive-worst language. The NL-CoT prompt, the proprietary-API direct-mode caveat, and the mode-agnostic QA evaluator are described in Appendix G.

4.2 Results across three domains

We evaluate Code-CoT on three domains: SpatialBabel-Primitive-QA, SpatialBabel-Hypersim-QA, and CV-Bench-3D (Tong et al., 2024) (1,200 real-photo 3D spatial-relation MCQs). Per-cell tables for the latter two are in Appendices H and I.

Table 2: SpatialBabel-QA-Score (%) under four inference modes on SpatialBabel-Primitive-QA (100 held-out primitive scenes, 652 questions per cell). `best_cc` / `worst_cc` use each model's primitive-best / primitive-worst scene-code language from Table 1, selected over all six scene-code languages. Bold = row max. All cells use the mode-agnostic evaluator (Appendix G).

Model	direct	nl_cot	best_cc	worst_cc	$\Delta_{\text{best-direct}}$
Proprietary APIs					
Claude Opus 4.7	59.8	**60.4**	58.0	58.1	(−1.8)
Claude Sonnet 4.6	52.9	52.3	**57.2**	57.1	(+4.3)
GPT-5	55.4	55.2	**55.7**	55.2	(+0.3)
GPT-4o	52.8	51.4	54.8	**57.1**	(+2.0)
Gemini-2.5-Pro	55.8	**64.6**	62.7	**64.6**	(+6.9)
Gemini-2.5-Flash	**59.0**	56.7	58.9	**59.0**	(−0.2)
Gemini-3-Flash	58.1	**60.9**	**60.9**	60.3	(+2.8)
Gemini-3-Pro	**58.0**	**58.0**	56.1	53.5	(−1.8)
Open weights					
InternVL3-8B	46.0	50.5	**54.9**	49.5	(+8.9)
LLaVA-OV-7B	49.1	45.6	52.0	**60.3**	(+2.9)
Qwen3-VL-8B	**48.0**	47.2	46.0	36.0	(−2.0)
Qwen2.5-VL-7B	55.7	53.8	**58.0**	53.8	(+2.3)
Qwen2.5-VL-3B	44.6	**51.2**	22.9	40.2	(−21.8)

Table 2 reports SpatialBabel-Primitive-QA accuracy. Primitive-best Code-CoT helps a broad range of models (InternVL3-8B +8.9%, Gemini-2.5-Pro +6.9%, Sonnet +4.3%, LLaVA-OV-7B +2.9%, Gemini-3-Flash +2.8%, Qwen2.5-VL-7B +2.3%, GPT-4o +2.0%); the remaining proprietary APIs stay within ∼2% of direct (GPT-5 +0.3%, Gemini-2.5-Flash −0.2%, Opus 4.7 −1.8%, Gemini-3-Pro −1.8%); and the weakest coders regress (Qwen3-VL-8B −2.0%, Qwen2.5-VL-3B −21.8%), which can be addressed with S^3-FT (Section 5) by lifting reconstruction quality across all scene-code languages. Qwen2.5-VL-3B illustrates why a model's reconstruction-best language need not be its QA-best one: both `best_cc` and `worst_cc` are keyed to each model's reconstruction-best and reconstruction-worst language (Table 1), and this model's reconstruction-best language is the verbose Blender Python, on which greedy decoding runs into repetition before committing a final answer, so here `worst_cc` ends up ahead of `best_cc`. Beyond primitive scenes, Code-CoT's benefit is domain-dependent. On Hypersim (Appendix Table 9), primitive-best Code-CoT is sign-mixed, indicating a domain ceiling once cluttered scenes break the primitive-shape vocabulary. On CV-Bench-3D (Appendix Table 10), primitive-best Code-CoT lifts the strongest code generators (Sonnet +5.9%, Opus 4.7 +5.0%, GPT-5 +3.2%) while the ceiling-saturated Gemini-3 APIs stay within ±1% and weak coders regress (Qwen3-VL-8B −9.7%, Qwen2.5-VL-7B −4.6%). Two qualifications recur across all three domains. NL-CoT often matches or exceeds primitive-best Code-CoT on raw accuracy (e.g., Gemini-2.5-Pro NL-CoT 64.6% vs. best_cc 62.7% on primitive QA). The point of Code-CoT is therefore not to beat NL-CoT on accuracy, but to expose how sensitive a model is to the scene-code language it reasons through: because Code-CoT answers via a chosen language, the gap between a model's `best_cc` and `worst_cc` measures how much that choice alone shifts QA accuracy, which NL-CoT cannot reveal. And primitive-best struggles on cluttered indoor and real-photo scenes, an observation we discuss further in Section 6.

4.3 Why 3D geometric primitives work: code as spatial tokenization

Where Code-CoT helps the most (primitive scenes and a subset of CV-Bench-3D; Section 4.2, Appendix I) suggests that VLM code training has tokenized 3D space through geometric primitive shapes. Writing Three.js code to place a `BoxGeometry` at $(2, 0, -3)$ with scale $(1, 2, 1)$ expresses spatial structure in a vocabulary that the language-model backbone has been extensively trained on. Three properties make primitives a useful scaffold beyond natural-language and 2D-coordinate alternatives: they are *compositional* (any scene decomposes into canonical shapes parameterized by position, rotation, scale); they are *richer than only 2D coordinates*; and the primitive approximation preserves the spatial relations that drive QA (above, left-of, in-front-of, closer-than) even when geometry is simplified.

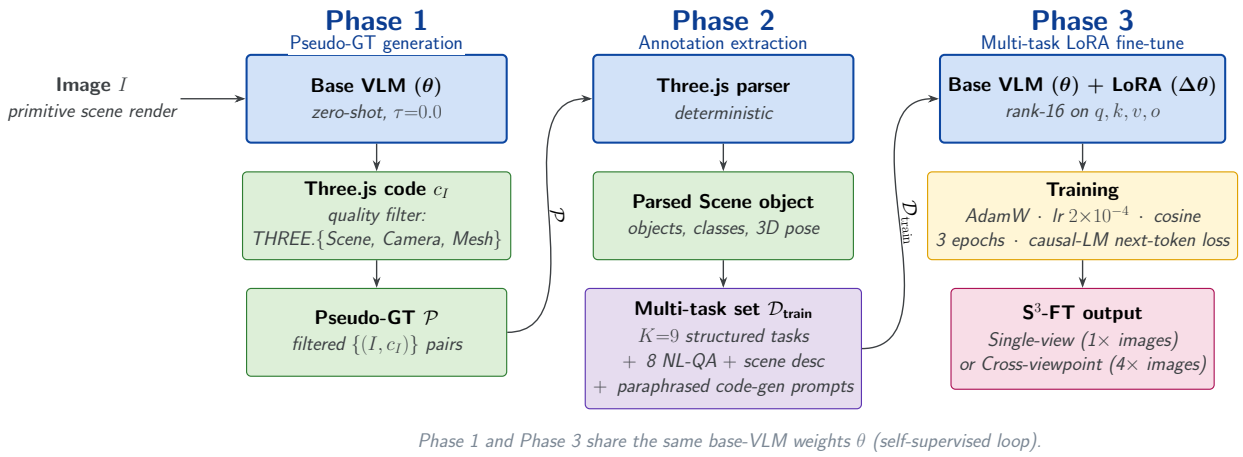

Figure 3: The three-phase S^3-FT pipeline (Phase 1: pseudo-GT generation; Phase 2: structured annotation extraction; Phase 3: multi-task LoRA fine-tuning).

The qualifier is a domain ceiling: when scenes cannot be tokenized into a small primitive vocabulary (cluttered Hypersim indoors, weak coders), the same reconstruction step that helps elsewhere hurts, and NL-CoT, which avoids the primitive commitment, is the safer scaffold. Code-CoT is therefore a primitive-vocabulary scaffold: it helps where a scene can be relatively well represented by a primitive vocabulary, and offers no advantage over the direct and NL-CoT baselines where it does not.

5 S^3-FT: Self-Supervised Distillation of Primitive Spatial Knowledge

S^3-FT targets the weak-coder regime that Code-CoT cannot address by distilling primitive spatial knowledge into the same VLM that generated the pseudo-ground-truth, with no human labels and no teacher model.

5.1 Pipeline

From a set of primitive scene images $\mathcal{I}$, S^3-FT proceeds in three phases (Figure 3). **Phase 1** queries the base VLM zero-shot for a Three.js primitive-reconstruction per image and applies a syntactic quality filter (the Three.js code declares `THREE.Scene`/`Camera`/`Mesh`; object count in $[1, 50]$; positions in $[-30, 30]$) to form a pseudo-ground-truth set $\mathcal{P} = \{(I, c_I)\}$. **Phase 2** parses each c_I deterministically into a `Scene` and derives K=9 structured-task targets (object count, classes, 3D positions, 3D / 2D bounding boxes, spatial relations, depth ordering, scene graph, code generation) plus rule-generated natural-language signals (8 QA pairs, a scene-description paragraph, and two paraphrases of the code-gen prompt per scene); the NL signals are essential: in an ablation, removing them causes the model to answer every prompt with scene-graph JSON, losing its ability to respond in natural language (Appendix J.1). **Phase 3** fine-tunes the same VLM with LoRA (Hu et al., 2022) (rank 16) on the attention query, key, value, and output (`q/k/v/o_proj`) projections (cosine schedule, 3 epochs); we call this configuration Single-view S^3-FT; a Cross-viewpoint S^3-FT variant additionally trains on each scene's other rendered viewpoints (Appendix J.2). Every supervised signal originates from the VLM's own output in $\mathcal{P}$, so the procedure requires no human labels and no teacher model.

5.2 In-distribution results

Figure 4 summarizes the three-axis evaluation. Mean SpatialBabel-Primitive-Reconstruct Score (Figure 4(a)) rises from 0.28 to **0.40** (Qwen2.5-VL-7B) and from 0.45 to **0.50** (Qwen3-VL-8B) under Single-view S^3-FT. On SpatialBabel-Primitive-QA (Figure 4(b)), S^3-FT lifts Qwen3-VL-8B across all four inference modes (the three non-collapse modes by +5.5 to +10.3%; mean over all four +11.0%; Table 3).[1] On Qwen2.5-VL-7B three of four modes improve (+3.2%, +3.8%, +3.1%) but primitive-best Code-CoT (Canonical JSON) regresses

[1]The fourth mode, primitive-worst Blender Code-CoT, gains +18.6% (36.0 → 54.6); most of this is S^3-FT repairing the greedy-decoding repetition collapse that depressed the base model in this verbose language (Section 4), so we do not headline it.

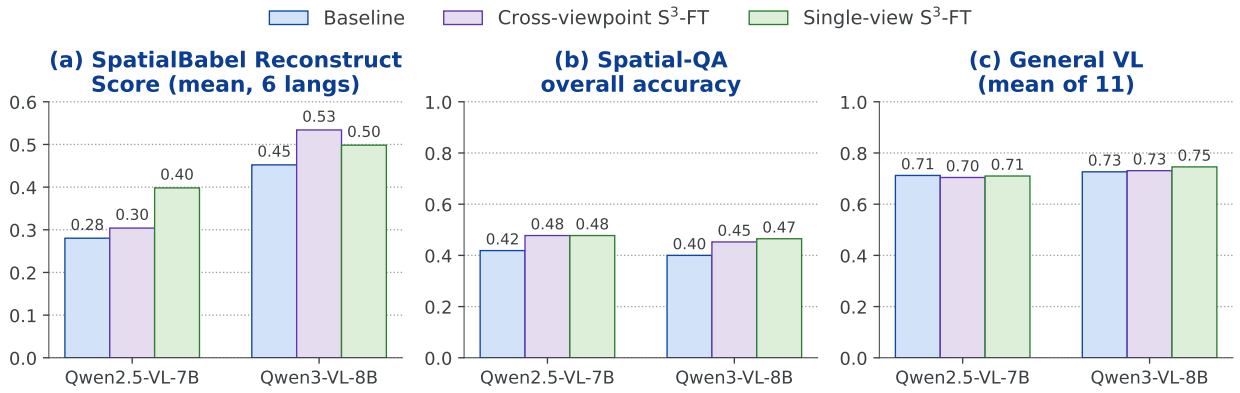

Figure 4: S^3-FT vs. Baseline on two base VLMs. **(a)** SpatialBabel-Primitive-Reconstruct mean Reconstruct Score (in-distribution). **(b)** SpatialBabel-Primitive-QA accuracy (in-distribution). **(c)** mean across 11 standard VLM benchmarks (out-of-distribution).

Table 3: SpatialBabel-Primitive-QA-Score (%) under four inference modes after Single-view S^3-FT, on the same 652 primitive-scene questions per cell as Table 2 (100 held-out primitive scenes). Δ denotes the per-cell improvement over the base model.

	Qwen3-VL-8B		Qwen2.5-VL-7B	
Mode	base	Single-view S^3-FT	base	Single-view S^3-FT
direct	48.0	**57.4** (+9.4)	55.7	**58.9** (+3.2)
nl_cot	47.2	**57.5** (+10.3)	53.8	**57.7** (+3.8)
best_code_cot (canonical_json)	46.0	**51.5** (+5.5)	**58.0**	47.2 (−10.7)
worst_code_cot (blender_py)	36.0	**54.6** (+18.6)	53.8	**56.9** (+3.1)

(−10.7%) because same-distribution distillation overwrites that base's already-clean best-language CoT, leaving the four-mode mean roughly flat. The Qwen3 gain is largest in the *comparison* category, where accuracy rises from ∼4% to ∼42% (Appendix J.4), and is seed-robust (±0.4–2.7% over 3 seeds on the non-collapse modes). Cross-viewpoint augmentation (∼3.5× data) helps the stronger Qwen3-VL-8B base but destabilizes Qwen2.5-VL-7B; cross-base validation on Qwen2.5-VL-3B, InternVL3-8B, and LLaVA-OV-7B is in Appendix J.7.

5.3 Out-of-distribution generalization: from primitives to real photos

A critical question is whether spatial gains from primitive-only training transfer to real-photo benchmarks (summarized in Figure 4(c)). Table 4 evaluates both base and S^3-FT configurations on eleven standard VLM benchmarks via VLMEvalKit (Duan et al., 2024) (text-only instruction-following via IFEval is in Appendix J.6). The results demonstrate that **primitive spatial knowledge transfers to general visual reasoning**.

For Qwen3-VL-8B under Single-view S^3-FT (trained *only* on primitive images), most benchmarks are preserved or improved, with the spatial benchmarks showing the largest gains: HallusionBench +17%, CV-Bench-2D +9.7%, CV-Bench-3D +3.3%, and BLINK +2.1%. These are real-photo benchmarks not seen during training, and the CV-Bench and BLINK improvements constitute the primary out-of-distribution evidence that spatial knowledge learned from primitives transfers to real-photo visual reasoning. Minor regressions on POPE (−2.8%) and OCRBench (−5.1%) reflect the absence of OCR and hallucination supervision in the primitive-only training mixture.

Table 4: General VLM performance across 11 standard benchmarks. Numbers are accuracy (%); parenthesized values are signed deltas versus the corresponding base model (green = improvement, red = regression). **Bold** = best per row per base model. † MMMU re-scored with a regex letter extractor (validation split, n=900) instead of VLMEvalKit's default LLM-judge for base→S^3-FT consistency; small deltas on this row are within re-scoring noise. S^3-FT is trained *only* on primitive images, yet gains transfer to real-photo benchmarks.

	Qwen3-VL-8B			Qwen2.5-VL-7B		
Benchmark	base	Cross-viewpoint	Single-view	base	Cross-viewpoint	Single-view
Hallucination						
POPE	**88.2**	86.2 (−2.0)	85.4 (−2.8)	**86.5**	85.2 (−1.3)	85.4 (−1.1)
HallusionBench (aAcc)	55.8	72.5 (+16.7)	**72.9** (+17.1)	65.2	**69.4** (+4.2)	68.8 (+3.6)
General reasoning						
$\text{MMBench}_{\text{dev-EN}}$	**84.4**	81.3 (−3.1)	81.9 (−2.5)	80.5	80.0 (−0.5)	**81.7** (+1.2)
MMStar	**64.6**	63.7 (−0.9)	64.1 (−0.5)	59.7	60.6 (+0.9)	**61.2** (+1.5)
MMMU_{val}†	**56.8**	54.9 (−1.9)	56.0 (−0.8)	**52.4**	49.6 (−2.8)	51.0 (−1.4)
AI2D	**84.1**	82.6 (−1.5)	83.6 (−0.5)	**80.9**	80.6 (−0.3)	80.9 (±0.0)
OCR / real-world						
OCRBench	**90.0**	84.4 (−5.6)	84.9 (−5.1)	**88.6**	85.3 (−3.3)	86.6 (−2.0)
RealWorldQA	71.0	69.0 (−2.0)	**72.2** (+1.2)	68.2	66.7 (−1.5)	**68.8** (+0.6)
Spatial reasoning						
BLINK	57.1	57.5 (+0.4)	**59.2** (+2.1)	**54.4**	53.4 (−1.0)	53.7 (−0.7)
CV-Bench-2D	70.3	78.7 (+8.4)	**80.0** (+9.7)	75.2	**77.3** (+2.1)	76.6 (+1.4)
CV-Bench-3D	76.8	73.2 (−3.6)	**80.1** (+3.3)	**71.8**	66.7 (−5.1)	66.6 (−5.2)

For the weaker Qwen2.5-VL-7B, the same recipe yields directionally consistent but smaller gains (HallusionBench +3–4%, CV-Bench-2D +1.4%), consistent with a capacity-dependent ceiling on the spatial knowledge extractable from noisier primitive pseudo-ground-truth.

The recipe is most effective on primitive scenes, where the base model's pseudo-ground-truth is high-quality. On Hypersim photo-realistic indoor scenes, the base model's own pseudo-ground-truth covers only ∼38% of objects per scene (measured on Qwen3-VL-8B and Qwen2.5-VL-7B). S^3-FT still lifts direct and NL-CoT modes on Hypersim (+3.8 and +7.6% on Qwen3, 0.0 and +9.5% on Qwen2.5; Appendix Table 15), but the Code-CoT gains seen on primitives do not carry over to this domain (Three.js Code-CoT is flat on Qwen3 at −0.9% and regresses −11.5% on Qwen2.5), because the low pseudo-ground-truth coverage on cluttered indoor scenes adds substantial label noise to the structured-task targets that Code-CoT depends on. Improving pseudo-ground-truth quality through a stronger base model or a multi-pass aggregation strategy is a promising path forward; details are discussed in Appendix J.8.

6 Limitations

S^3-FT is bounded by base-VLM primitive pseudo-GT quality: on Hypersim (∼38% object coverage on Qwen-VL; Appendix J.8) the pipeline does not transfer, and Code-CoT struggles once scene complexity exceeds what a primitive vocabulary can capture. The most promising application domain is therefore stylized, low-texture scenes (architectural visualizations, video-game-style graphics, robot simulators); photo-realistic transfer will likely be more challenging. The cross-language ranking from SpatialBabel-Primitive-Reconstruct (Table 1) also does not transfer cleanly to real photos, so the ranking is an informative default rather than a deterministic Code-CoT or S^3-FT language selector; we use Three.js as a single-language S^3-FT anchor and leave per-model language ablation to future work.

7 Conclusion

VLMs reconstruct 3D scenes from geometric primitives with high code-level fidelity yet fail at simpler spatial questions on the same images. SpatialBabel benchmarks this task-routing failure across six scene-code

languages and fourteen VLMs, with model-specific Reconstruct-Score gaps up to 0.40 across languages; Code-CoT routes around the failure at inference time for capable code generators, with a clear domain ceiling on cluttered photo-realistic scenes; S^3-FT closes it at training time for noisier coders, distilling primitive-reconstruction code into general visual reasoning with no human labeling and no teacher model. Together, 3D geometric primitives, as expressed in executable code, function as both a diagnostic instrument and a transferable spatial vocabulary for VLMs. For future work, we plan to enrich the scene-code primitive set to extend transfer into photo-realistic indoor scenes, sweep procedural complexity (occlusion, stacking, unconstrained palettes) to bridge the difficulty spectrum, and explore S^3-FT on stronger base VLMs. We will release the benchmark, training data, checkpoints, and evaluation toolkit upon publication. Broader impacts are discussed in Appendix K.

Reproducibility Statement

Full methodology, prompts, training and evaluation protocols, and per-table data sources are in Appendices A, G, and L. Upon publication we will release the benchmark, code, and checkpoints used to produce every result in the paper.

References

Nahid Alam, Leema Krishna Murali, Siddhant Bharadwaj, Patrick Liu, Timothy Chung, Drishti Sharma, Kranthi Kiran, Wesley Tam, Bala Krishna S Vegesna, et al. The spatial blindspot of vision-language models. *arXiv preprint arXiv:2601.09954*, 2026.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Armen Avetisyan, Christopher Xie, Henry Howard-Jenkins, Tsun-Yi Yang, Samir Aroudj, Suvam Patra, Fuyang Zhang, Duncan Frost, Luke Holland, Campbell Orme, et al. Scenescript: Reconstructing scenes with an autoregressive structured language model. In *European Conference on Computer Vision*, pp. 247–263. Springer, 2024.

Jinze Bai et al. Qwen2.5-vl: Seeing the world more clearly. 2025.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Shiqi Chen, Tongyao Zhu, Ruochen Zhou, Jinghan Zhang, Siyang Gao, Juan Carlos Niebles, Mor Geva, Junxian He, Jiajun Wu, and Manling Li. Why is spatial reasoning hard for vlms? an attention mechanism perspective on focus areas. *arXiv preprint arXiv:2503.01773*, 2025.

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

Google DeepMind. Gemini 2.5: A family of highly capable multimodal models. 2025.

Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, et al. Vlmevalkit: An open-source toolkit for evaluating large multi-modality models. In *Proceedings of the 32nd ACM international conference on multimedia*, pp. 11198–11201, 2024.

Yunqi Gu, Ian Huang, Jihyeon Je, Guandao Yang, and Leonidas Guibas. Blendergym: benchmarking foundational model systems for graphics editing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 18574–18583, 2025.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In *Forty-first International Conference on Machine Learning*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Raphi Kang, Hongqiao Chen, Georgia Gkioxari, and Pietro Perona. Linear mechanisms for spatiotemporal reasoning in vision language models. *arXiv preprint arXiv:2601.12626*, 2026.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2 (1-2):83–97, 1955.

Bo Li et al. Llava-onevision: Easy visual task transfer. 2024.

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.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pp. 740–755. Springer, 2014.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

Ruijie Lu, Yiyang Ma, Xiaokang Chen, Lingxiao Luo, Zhiyu Wu, Zizheng Pan, Xingchao Liu, Yutong Lin, Hao Li, Wen Liu, Zhewen Hao, Xi Gao, Shaoheng Nie, Yixuan Wei, Zhenda Xie, Ting Chen, and Gang Zeng. Thinking with visual primitives, 2026. DeepSeek-AI technical report.

Wufei Ma, Haoyu Chen, Guofeng Zhang, Yu-Cheng Chou, Jieneng Chen, Celso de Melo, and Alan Yuille. 3dsrbench: A comprehensive 3d spatial reasoning benchmark. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6924–6934, 2025.

Jianing Qi, Jiawei Liu, Hao Tang, and Zhigang Zhu. Beyond semantics: Rediscovering spatial awareness in vision-language models. *arXiv preprint arXiv:2503.17349*, 2025.

Mike Roberts, Jason Ramapuram, Anurag Ranjan, Atulit Kumar, Miguel Angel Bautista, Nathan Paczan, Russ Webb, and Joshua M Susskind. Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10912–10922, 2021.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3956–3974, 2025.

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai Charitha Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, Austin Wang, Rob Fergus, Yann LeCun, and Saining Xie. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. *NeurIPS*, 2024.

Peiran Xu, Sudong Wang, Yao Zhu, Jianing Li, Gege Qi, and Yunjian Zhang. Spatialbench: Benchmarking multimodal large language models for spatial cognition. *arXiv preprint arXiv:2511.21471*, 2025.

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*, pp. 10632–10643, 2025.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9556–9567, 2024.

Pingyue Zhang, Zihan Huang, Yue Wang, Jieyu Zhang, Letian Xue, Zihan Wang, Qineng Wang, Keshigeyan Chandrasegaran, Ruohan Zhang, Yejin Choi, Ranjay Krishna, Jiajun Wu, Li Fei-Fei, and Manling Li. Theory of space: Can foundation models construct spatial beliefs through active exploration? In *International Conference on Learning Representations (ICLR)*, 2026.

Yunzhi Zhang, Zizhang Li, Matt Zhou, Shangzhe Wu, and Jiajun Wu. The scene language: Representing scenes with programs, words, and embeddings. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 24625–24634, 2025.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

A Models and prompting setup

We evaluate **fourteen VLMs**: eight proprietary APIs (Anthropic Claude Opus 4.7 / Sonnet 4.6, OpenAI GPT-5 / GPT-4o, Google Gemini-2.5-Pro/Flash and Gemini-3-Pro/Flash (preview) (DeepMind, 2025)) and six open-weights bases: Qwen3-VL-8B-Instruct (the S^3-FT headline base), Qwen2.5-VL-7B-Instruct (second S^3-FT base) (Bai et al., 2025), Qwen2.5-VL-3B-Instruct, Qwen2.5-VL-32B-Instruct, LLaVA-OneVision-7B (Li et al., 2024), and InternVL3-8B (OpenGVLab). Open-weights models run on 4×NVIDIA A100 80GB with greedy decoding (`temperature=0.0`) for deterministic per-scene predictions; max-new-tokens 2,048 for code outputs (Direct reconstruction and Code-CoT), 1,024 for NL-CoT (which interleaves multi-sentence reasoning with a final committed answer; the lower 128-token cap used for direct answers truncated 68% of Qwen3-VL-8B NL-CoT responses on CV-Bench-2D before answer commitment), and 256 for short direct answers. The cross-language benchmark uses two prompting modes: **Direct** (image + language-specific reconstruction instruction listing required fields) and **NL-CoT** (a natural-language reasoning prefix enumerating objects, classes, and approximate 3D positions before emitting scene code; distinct from *Code-CoT* introduced later). We evaluate $n = 100$ primitive scenes per cell; Hypersim uses $n = 200$ per cell.

A.1 Prompt examples

We list the verbatim prompt templates used throughout the paper. The reconstruction prompt is composed of a fixed system instruction, a per-language body, and dataset-specific scene-content / coordinate hints; the QA prompts (Direct, NL-CoT, Code-CoT) are constructed at evaluation time and shown below in full.

Reconstruction, system prompt (all languages)

```
You are an expert 3D scene reconstruction assistant.  Given an image of a 3D scene, you must reconstruct
it as executable code in the specified programming language.  Output ONLY the code, no explanations.
```

Reconstruction, per-language body (Three.js example)

`Reconstruct this 3D scene as Three.js JavaScript code.` *[primitive hint]* `Use the appropriate geometry (BoxGeometry, SphereGeometry, CylinderGeometry, ConeGeometry, TorusGeometry) for each object.  Set exact positions, rotations, and scales.  Set MeshStandardMaterial color for each object.` *[coordinate hint]* *[camera hint]*

The five remaining bodies (Blender Python, Unity C#, Open3D Python, Scene Language DSL, Canonical JSON) follow the same template, swapping language-specific API verbs (e.g. `primitive_*_add()`, `GameObject.CreatePrimitive()`, `create_*`, `(entity ...)`). The Canonical JSON body inlines an explicit schema:

Reconstruction, per-language body (Canonical JSON)

```
Reconstruct this 3D scene as a JSON object with this exact schema:  {"scene_id":  "predicted", "objects":
[{"class_name":  "shape", "position":  [x,y,z], "rotation":  [rx,ry,rz], "scale":  [sx,sy,sz], "material":
"color_name"}], "camera":  {"position":  [x,y,z], "target":  [x,y,z], "fov":  60.0}}
```

Reconstruction, dataset hints (appended to body). Primitive hint: "`Objects are geometric primitives:  cube, sphere, cylinder, cone, torus, pyramid.  Each has a color from:  red, blue, green, yellow, purple, orange, cyan, white.`" **Hypersim hint:** "`This is a photorealistic indoor scene.  Objects are real furniture and items in a room.  Use descriptive class names for each object (e.g., chair, table, bathtub, lamp, bed).`" **Coordinate hint:** a Y-up right-handed convention plus per-axis spatial extent (e.g. "`The ground plane extends from` -3.0 `to` $+3.0$ `in both X and Z.`" for primitives; per-scene bounds for Hypersim) and a ground-plane convention (`y=0` with each object's y position equal to half its height for primitives; floor at $Y \approx 0$ in meters for Hypersim).

Spatial QA, Direct mode (Tables 2 to 4, `direct` column)

```
{question}
Answer concisely.
```

Spatial QA, NL-CoT mode (Tables 2 to 4, `nl_cot` column)

```
{question}

Think step-by-step about the objects in the scene, their classes, colours, and approximate 3-D positions.
Then give a concise final answer.
```

Spatial QA, Code-CoT mode (Tables 2 to 4, `best_cc` / `worst_cc` columns)

```
First, write a complete {language} reconstruction of the 3-D scene you see.  Include each object's class,
colour, and position.  Then, using your reconstruction, answer the following question.

Question:  {question}
Finish with a line starting with 'Final answer:'  followed by the concise answer.
```

`{language}` is replaced with a human-readable rendering of each model's primitive-best or primitive-worst scene-code language (e.g. "`Three.js JavaScript`", "`canonical JSON with the schema {...}`").

B Scene-code language exposure proxies

A key contribution of SpatialBabel is evaluating spatial understanding across six diverse 3D scene-code languages, spanning different paradigms and syntax families: **Three.js** (imperative web 3D), **Canonical JSON** (declarative structured), **Blender Python** (procedural scripting), **Unity C#** (engine API with transform components), **Open3D Python** (mesh-creation API), and a **Scene Language DSL** Zhang et al. (2025). They differ substantially in public footprint:

Table 5: Public proxy metrics for pre-training exposure (rounded order-of-magnitude figures, collected 2026); SO = Stack Overflow tagged-question count, GH = GitHub stars. Languages ordered by approximate pre-training prevalence (most → least). Canonical JSON is introduced by this work, so specific-schema exposure is zero, though generic JSON priors remain available to all models.

Language	Pkg downloads	SO questions	GH stars
Three.js	∼8M/wk (npm)	∼21k	∼110k
Unity C#	closed-source	∼78k	∼13k
Blender Python	∼16k/wk (PyPI)	∼3k	∼18k
Open3D Python	∼370k/wk (PyPI)	∼400	∼13k
Canonical JSON	n/a	n/a	n/a
Scene Lang. DSL	n/a	0	∼260

We treat pre-training exposure as a continuous axis in our analysis (Section 3.3) rather than binning the six scene-code languages into high- versus low-exposure groups, because which languages a model finds easy or hard is itself model-specific (Appendix F.1): no single high/low split holds across all models.

C Per-scene-code Reconstruct Score on Hypersim

Table 6 reports per-scene-code Reconstruct Scores on the Hypersim indoor split for the six models with full Hypersim coverage. It is the Hypersim counterpart of the primitive results in Table 1, and Figure 5 plots the primitive and Hypersim per-language scores side by side for the six models evaluated on both splits.

The Reconstruct Score drops substantially from the primitive to the Hypersim domain. Averaged over the six scene-code languages under Direct prompting, the mean falls from 0.572 to 0.322 for Claude Opus 4.7, 0.573 to 0.313 for Gemini-3-Pro, 0.443 to 0.305 for GPT-5, 0.483 to 0.269 for Gemini-2.5-Pro, and 0.452 to 0.277

for Qwen3-VL-8B; Qwen2.5-VL-7B is unchanged at 0.279, already near the primitive floor. Per-language gaps persist on Hypersim, though smaller than on primitives (for example, 0.17 for Gemini-2.5-Pro and 0.21 for Qwen3-VL-8B between their best and worst languages). The gaps are smaller because Hypersim is uniformly harder: with a median of 26 cluttered objects per scene, few objects are localized within tolerance in any scene-code language, so all Reconstruct Scores are low and closer together. The clean, occlusion-free primitive split therefore resolves the per-language differences most precisely, while Canonical JSON is the Hypersim-best for both Opus and GPT-5, matching GPT-5's primitive ranking (Opus's primitive-best is Three.js).

Table 6: Per-scene-code SpatialBabel-Reconstruct-Score on **Hypersim** indoor scenes (FOV-filtered, $n = 200$ scenes per row) for the six models with full Hypersim coverage. Format mirrors Table 1 for direct primitive↔Hypersim comparison: **bold** = best per model-row, red = lowest per model-row, Δ = max minus min Reconstruct Score gap. Hypersim numbers are FOV-filtered (GT objects whose 3D center falls outside the rendered camera frustum are excluded; Section 6); mean GT object count is 65.7/scene unfiltered, 61.3/scene FOV-filtered. The Reconstruct Score is dragged down on Hypersim primarily by detection F1 (objects are rarely localized within tolerance on cluttered scenes with a median of 26 objects) and class accuracy (60-class indoor taxonomy vs. primitive's 6 shapes).

Model	Three.js	Unity	Blen.Py	Open3D	JSON	DSL	Δ
Proprietary APIs							
Claude Opus 4.7	.320	.309	.322	.322	**.331**	.329	.022
GPT-5	.323	.270	.290	.286	**.337**	.324	.067
Gemini-2.5-Pro	.231	.332	.213	**.383**	.245	.212	.170
Gemini-3-Pro	**.330**	.312	.325	.287	.298	.327	.043
Open weights							
Qwen2.5-VL-7B	.294	.283	**.307**	.266	.256	.272	.051
Qwen3-VL-8B	.304	.306	.317	.304	.112	**.318**	.206

D SpatialBabel-Reconstruct-Score per-component breakdown

Table 7 reports the four Reconstruct Score components averaged across the six scene-code languages for each (model, mode) cell. The Reconstruct Score is their arithmetic mean (last column).

What the breakdown reveals. The table makes the failure modes explicit. (i) *Parse rate* (fraction of outputs that yield ≥ 1 object after parsing) ranges from ~ 0.60 on the weakest 7B/3B Qwen bases to near 1.0 on the strongest proprietary APIs (Claude Opus 4.7, Gemini-3-Pro); Gemini-2.5-Pro sits at 0.81 owing to a small fraction of refusals on JSON output. The dominant failure on the weak Qwen bases is empty parses (valid syntax, zero objects extracted), not malformed code. (ii) *Detection F1*, the localization-gated and count-aware component, is low across the board under the strict tolerance sweep and separates models that place objects within tolerance (Gemini-3-Pro 0.26, Gemini-2.5-Pro 0.23) from those that do not (Qwen2.5-VL-7B 0.05). (iii) *Class accuracy*, normalized by the ground-truth object count, is the largest cross-model gap: the strongest proprietary APIs reach 0.39–0.46 (Claude Opus 4.7, Gemini-3-Pro) while open-weights bases sit between 0.17 and 0.30. (iv) *Scale fidelity*, likewise normalized by the ground-truth object count, ranges from 0.30 on the weakest base to $\sim$0.65 on Claude Opus 4.7, reflecting both size accuracy and how many objects were recovered.

E Reconstruct Score vs QA Score correlation

Section 3.3 reports that reconstruction skill (Reconstruct Score) and direct spatial answering (QA Score) are moderately correlated across models, driven by overall capability, but decouple among the strongest models. Table 8 reports the per-model values behind these correlations, computed as follows. Each model contributes one data point, pairing its mean Reconstruct Score (averaged over the six scene-code languages

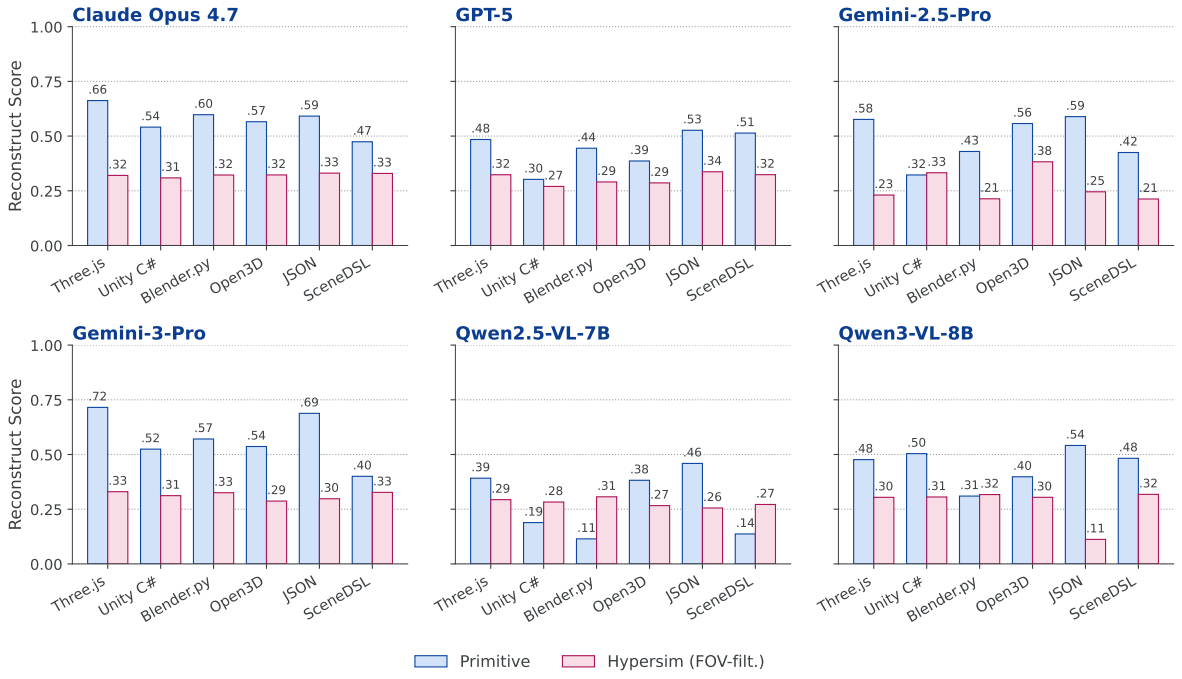

Figure 5: Primitive vs. Hypersim per-scene-code Reconstruct Score (Direct prompting) for the six models with both splits. Hypersim numbers are FOV-filtered (full per-component metrics computed against FOV-matched pairs), matching Table 6. Hypersim scenes are substantially harder across all languages, and the cross-language gaps compress relative to the primitive split (Section 3.3).

under Direct prompting) with its accuracy on the *relationship* category of SpatialBabel-QA: the questions that test spatial relations between objects (whether one object is above, left of, in front of, or closer than another), as distinct from the counting, existence, and comparison categories. We single out this category because it is the most spatially demanding and the one Code-CoT most affects, and we report overall QA accuracy beside it. We summarize each association two ways: Pearson r for the strength of the linear relationship between the two scores, and Spearman ρ for the monotonic (rank) relationship, which is less sensitive to individual outliers; both lie in $[-1, 1]$, where values near zero indicate no association and positive values indicate that stronger reconstructors tend to answer relational questions better. The per-scene-code rows at the bottom repeat the same correlation but substitute a model's Reconstruct Score in a *single* scene-code language for the six-language mean, so a large value flags a language whose reconstruction quality most closely tracks relational-QA ability across models.

Two observations refine this correlation. **(i) It is largely capability-driven.** The moderate $r \approx 0.5$ comes mainly from the weaker open-weights models: InternVL3-8B and Qwen2.5-VL-3B score lowest on both reconstruction and QA. Among the most capable models QA-relationship saturates within a 2.5-pp band while the Reconstruct Score still varies by more than 2×, so reconstruction skill does not predict QA at the top end. **(ii) Reconstruction skill in some scene-code languages tracks relational QA more closely than in others.** Computed per scene-code language, Open3D Python (+0.79) and Three.js (+0.73) track relational QA most closely, Canonical JSON moderately (+0.49), while Unity C#, Blender Python, and Scene Language DSL are near zero or rank-negative. That stronger reconstruction does not lift relational QA among capable models suggests the bottleneck there is not reconstruction fidelity but the step from spatial *evidence* to a directly-stated relational *conclusion*, the step that Code-CoT (Section 4) targets by routing QA through reconstruction.

Table 7: SpatialBabel-Reconstruct-Score components, per (model, mode), averaged across the six scene-code languages on primitive scenes. **Parse** = parse rate; **Det.F1** = localization-gated detection F1 averaged over the tolerance sweep; **ClsAcc** = class accuracy normalized by the ground-truth object count; **ScaleFid** = scale fidelity normalized by the ground-truth object count.

Model	Mode	Parse	Det.F1	ClsAcc	ScaleFid	Mean
Proprietary APIs						
Claude Opus 4.7	base	0.993	0.251	0.392	0.652	**0.572**
	NL-CoT	1.000	0.240	0.411	0.677	**0.582**
GPT-5	base	0.852	0.171	0.303	0.447	**0.443**
	NL-CoT	0.817	0.145	0.239	0.418	**0.405**
Gemini-2.5-Pro	base	0.814	0.227	0.373	0.519	**0.483**
	NL-CoT	0.931	0.267	0.406	0.590	**0.549**
Gemini-3-Pro	base	0.957	0.262	0.457	0.615	**0.573**
	NL-CoT	0.953	0.275	0.441	0.585	**0.564**
Open weights						
Qwen2.5-VL-32B	base	0.856	0.154	0.298	0.440	**0.437**
	NL-CoT	0.922	0.156	0.286	0.443	**0.452**
Qwen3-VL-8B	base	0.908	0.131	0.260	0.509	**0.452**
	NL-CoT	0.797	0.121	0.261	0.491	**0.417**
Qwen2.5-VL-7B	base	0.597	0.054	0.168	0.296	**0.279**
	NL-CoT	0.802	0.091	0.208	0.452	**0.388**
LLaVA-OV-7B	base	0.851	0.107	0.247	0.599	**0.451**
	NL-CoT	0.761	0.085	0.188	0.522	**0.389**
Qwen2.5-VL-3B	base	0.657	0.088	0.172	0.357	**0.318**
	NL-CoT	0.842	0.098	0.176	0.403	**0.380**

F Additional benchmark visualizations and findings

F.1 Language-difficulty rankings and scale findings

Figure 6 ranks the six scene-code languages within each model (1 = best), plotting the per-model best and worst languages from Table 1, so that each model's best and worst languages can be read off directly. Models disagree on which languages are easy; we then examine how this ranking pattern changes with model scale.

The benchmark scales with model size. Within the Qwen2.5 family (a controlled three-point sweep at 3B, 7B, and 32B parameters), the mean Reconstruct Score across all six languages rises with parameter count. Under Direct: 0.318 (3B), 0.279 (7B), 0.437 (32B); under NL-CoT: 0.380 / 0.388 / 0.452. The 32B model is well clear of both smaller bases (the 3B and 7B scores are within noise of each other under Direct), and the 10× 3B→32B step lifts the mean Reconstruct Score by ∼0.12 (Direct) and ∼0.07 (NL-CoT). That the largest model clearly tops the sweep is a sanity check for the benchmark itself: a benchmark that did not separate model sizes would suggest its signal was dominated by something other than spatial-understanding capacity.

Scale changes *where* the gaps appear, not just their magnitude. Smaller models (Qwen2.5-VL-3B, LLaVA-OV-7B) underperform on less-exposed languages such as Open3D and Scene Language DSL, the intuitive failure mode. Larger models (Qwen2.5-VL-7B with NL-CoT, Gemini-2.5-Pro) instead tend to fail on specific widely-exposed APIs while excelling on novel formats: Qwen2.5-VL-7B scores 0.16 on Blender Python under NL-CoT while reaching 0.46 on Scene Language DSL; Gemini scores 0.32 on Unity C# (its lowest Reconstruct Score) while reaching 0.59 on Canonical JSON. Small models are exposure-bound; large models have absorbed all six APIs but still show model-specific weak spots driven by API *semantics* rather than aggregate public code frequency. In neither regime does scale eliminate cross-language variance.

Table 8: Per-model comparison of the Reconstruct Score (averaged across the six scene-code languages, Direct prompting) against SpatialBabel-Primitive-QA accuracy on the relationship category and overall (100 scenes, Direct prompting; overall is over 652 questions across four categories). Rows are ordered by Reconstruct Score (descending). Across all fourteen VLMs, the Reconstruct Score spans 0.279 to 0.573 (a 2.05× ratio) while QA-relationship spans 30.9 to 38.2%. Pearson r and Spearman ρ (defined in the text) are reported at the bottom, both for the six-language mean and per scene-code language.

Model	Reconstruct Score (mean of 6 langs)	QA-relationship (%)	QA-overall (%)
Gemini-3-Pro	**.573**	37.9	58.0
Claude Opus 4.7	.572	**38.2**	**59.8**
GPT-4o	.530	37.3	52.8
Claude Sonnet 4.6	.527	37.9	52.9
Gemini-3-Flash	.523	37.3	58.1
Gemini-2.5-Pro	.483	**38.2**	55.8
Gemini-2.5-Flash	.470	37.6	59.0
Qwen3-VL-8B	.452	35.7	48.0
LLaVA-OV-7B	.451	35.4	49.1
GPT-5	.443	34.7	55.4
Qwen2.5-VL-32B	.437	37.3	51.5
InternVL3-8B	.429	30.9	46.0
Qwen2.5-VL-3B	.318	31.5	44.6
Qwen2.5-VL-7B	.279	37.9	55.7
Reconstruct Score (mean) vs. QA-relationship		Pearson $r = 0.48$	Spearman $\rho = 0.58$
Reconstruct Score (mean) vs. QA-overall		Pearson $r = 0.50$	Spearman $\rho = 0.59$

Per-scene-code Reconstruct Score vs. QA-relationship (Pearson r / Spearman ρ, $n = 14$):
Open3D Py $+0.79/+0.73$ · Three.js $+0.73/+0.74$ · Canon. JSON $+0.49/+0.56$
Unity C# $+0.04/+0.23$ · Blender Py $-0.07/+0.27$ · Scene DSL $+0.05/-0.36$

F.2 Same-scene cross-language qualitative comparison

Figure 7 shows a single scene reconstructed by one model (Gemini-2.5-Pro) in four scene-code languages, illustrating how the same spatial understanding is expressed with different placement fidelity across languages.

G Code-CoT setup details and QA evaluator

This appendix expands on the Code-CoT setup paragraph (Section 4.1).

Direct mode for proprietary APIs is not literally "no reasoning". For proprietary APIs (Claude Opus 4.7, Claude Sonnet 4.6, GPT-5, GPT-4o, and the four Gemini variants), direct mode still executes internal reasoning tokens before the visible response; NL-CoT and Code-CoT therefore mainly change the *visible* reasoning scaffold for these models. For open-weights bases (Qwen3-VL-8B, Qwen2.5-VL-7B/3B, InternVL3-8B, LLaVA-OV-7B), sampled direct-mode outputs average 5 to 7 characters with no `<think>` traces, so chain-of-thought is genuinely absent under direct mode.

NL-CoT prompt. Our NL-CoT prompt asks the model to think step-by-step about object class, color, and approximate 3D position before producing a final answer; this is an information-matched prose analog of canonical-JSON Code-CoT, deliberately strong so that any Code-CoT advantage cannot be attributed to merely cuing the model to attend to object attributes.

mode-agnostic QA evaluator. Our QA scorer (the `evaluate_qa_answer` routine in the released code) applies the same per-category parsers uniformly across direct, NL-CoT, and Code-CoT, eliminating prompt-format asymmetries that earlier paper drafts inherited from full-response substring matching.

Per-category extraction. (i) *Counting:* prefer the integer following an explicit "Final answer:" / "the answer is:" / "total:" marker; otherwise take the last integer in the response (chain-of-thought scratchpads

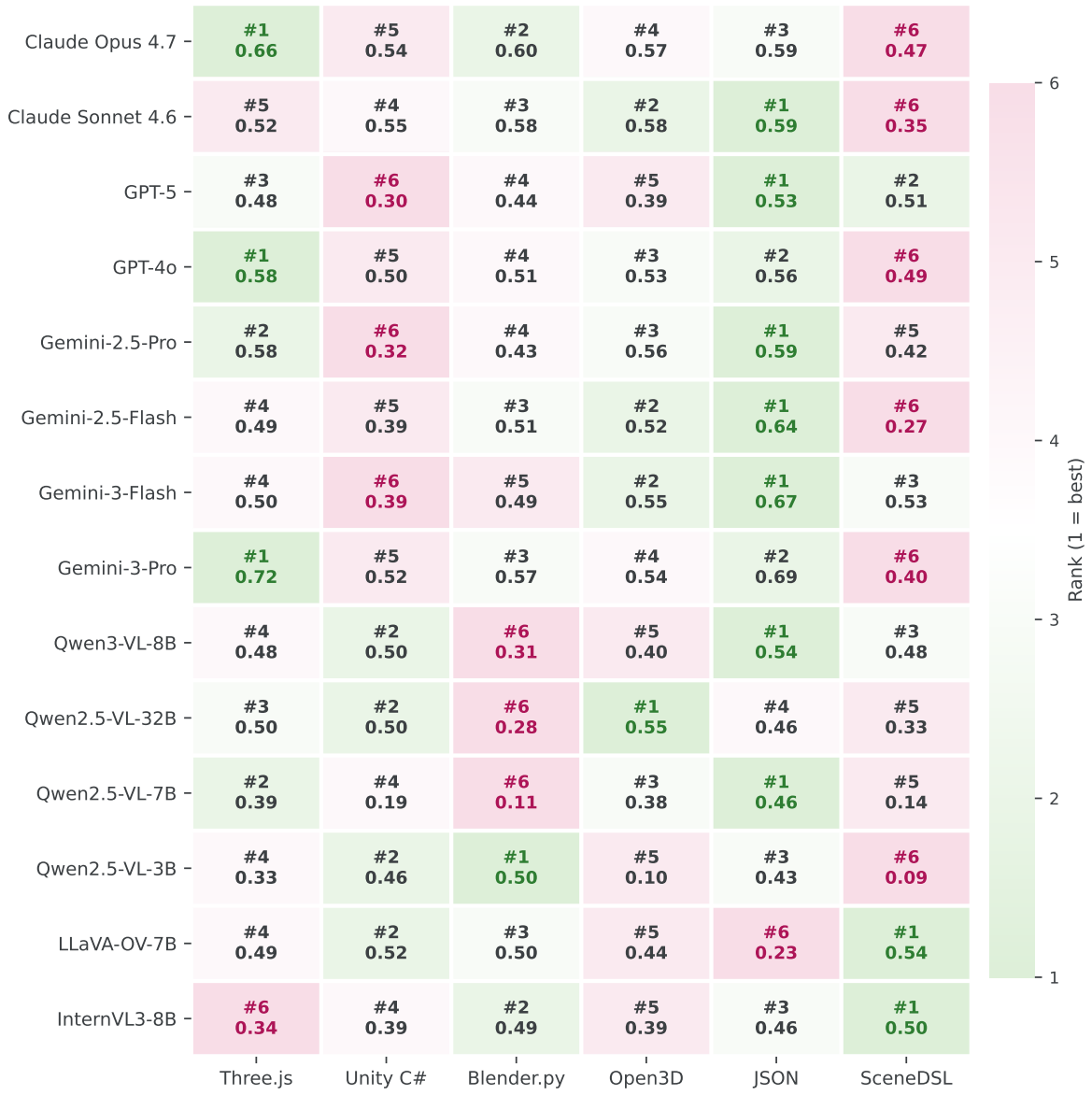

Figure 6: Language difficulty rankings per model (baseline, primitive scenes). Each cell shows the model's rank among the 6 languages (1 = best) and absolute Reconstruct Score. Models disagree on which languages are easy or hard, and the ranking pattern correlates with model scale: smaller models (Qwen2.5-VL-3B, LLaVA-OV-7B) rank high-exposure languages above low-exposure ones, whereas larger models show the opposite pattern driven primarily by Canonical JSON being structurally simple.

typically commit at the end). (ii) *Existence:* prefer the *yes/no* token following an explicit answer marker; otherwise take the last *yes/no* word-boundary token. (iii) *Relationship* and (iv) *comparison:* extract the trailing committed answer segment (the final "Final answer:" / "Answer:" line, or the last 1 or 2 non-empty lines) and substring-match the ground truth phrase against *that* segment, not the full response. **Code-CoT pre-extraction.** For Code-CoT modes, we additionally apply a markdown-tolerant "Final answer:" line extractor before passing the response to the scorer, so both Code-CoT and NL-CoT scoring see the model's committed answer rather than its scratchpad.

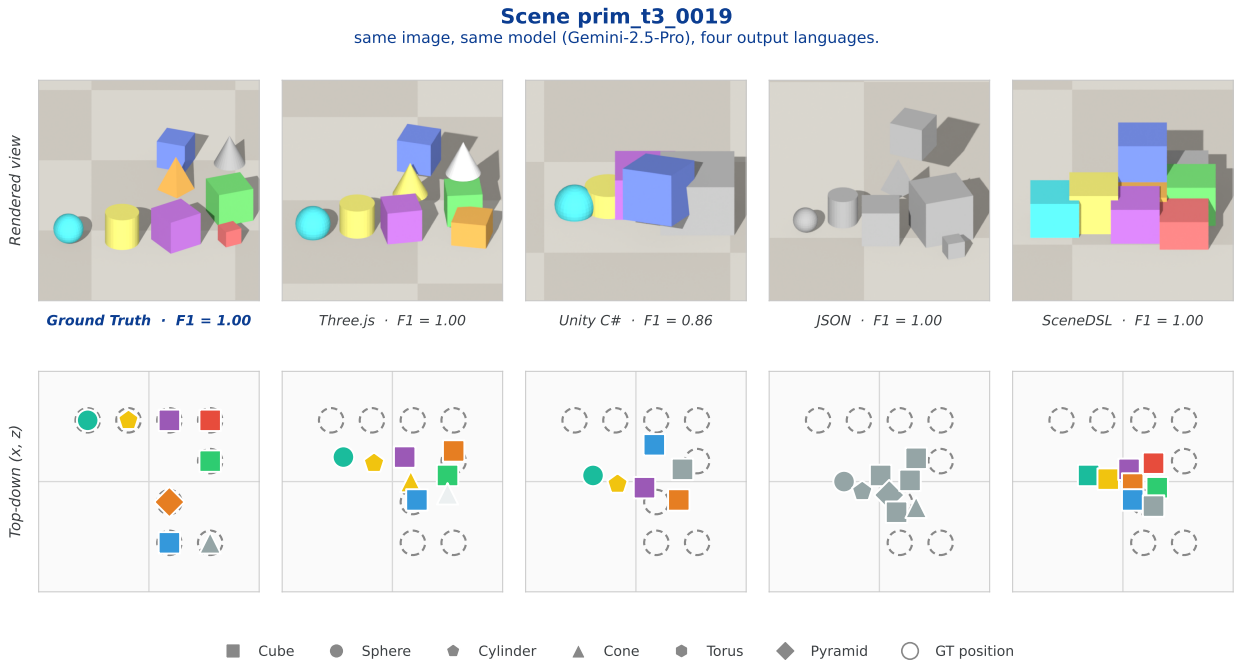

Figure 7: Same scene, same model (Gemini-2.5-Pro), four scene-code languages. **Top row:** 3D render of the ground truth and of each predicted scene, reconstructed from the model's generated code and rendered with the GT camera. **Bottom row:** top-down (x,z) layout; dashed rings = GT positions, filled markers = predicted positions. Gemini's Unity C# output recovers the scene with slightly different object placement than its Three.js output, despite both describing the same image.

The mode-agnostic variant brings NL-CoT scoring strictness in line with Code-CoT scoring strictness so the inference-mode comparison in Tables 2, 9, and 10 is apples-to-apples.

H Code-CoT on SpatialBabel-Hypersim-QA

Table 9 reports the four-mode SpatialBabel-Hypersim-QA evaluation (105 questions per cell, 20 held-out photo-realistic indoor scenes). Primitive-best Code-CoT is sign-mixed: four of ten models gain (Gemini-2.5-Flash +9.6%, Gemini-2.5-Pro +7.6%, Sonnet +6.6%, Gemini-3-Flash +5.8%) and six regress (Opus 4.7 −7.6%, Qwen2.5-VL-7B −7.6%, GPT-4o −5.7%, Gemini-3-Pro −5.7%, GPT-5 −4.8%, Qwen3-VL-8B −2.9%). Under the mode-agnostic evaluator the row maxima remain spread across direct, NL-CoT, and Code-CoT, with no single mode dominating. The primitive-best language is also a leaky predictor on cluttered indoor scenes (e.g., GPT-4o's primitive-worst DSL beats its primitive-best Three.js by 13.4pp, and Qwen2.5-VL-7B's primitive-worst Blender Python beats its primitive-best Canonical JSON by 5.7pp), an inversion that recurs on CV-Bench-3D and is discussed in Section 6.

I CV-Bench Code-CoT, 2D and 3D (out-of-distribution)

SpatialBabel's questions are programmatically generated, so they favor code-CoT by construction. To test whether the lift generalizes to real-photo spatial reasoning, we repeat the four-mode comparison on CV-Bench-2D and CV-Bench-3D (Tong et al., 2024) (1,438 and 1,200 real-photo MCQs respectively); the per-cell results are in Table 10. Two patterns separate the two splits. **(i) Code-CoT lifts on 3D where it does not on 2D.** On CV-Bench-2D, primitive-best Code-CoT moves accuracy only modestly relative to direct: it gains for Claude Sonnet 4.6 (+2.2%), Gemini-2.5-Pro (+1.1%), and Gemini-3-Flash (+0.4%), and is flat-to-negative for the rest (down to Qwen3-VL-8B −4.9%). On CV-Bench-3D the picture is more positive:

Table 9: SpatialBabel-Hypersim-QA-Score (%) under four inference modes. `best_cc` / `worst_cc` use each model's primitive-best / primitive-worst scene-code language from Table 1. $\Delta_{\text{best-direct}}$ = `best_cc` − `direct` (pp). Bold = row max.

Model	direct	nl_cot	best_cc	worst_cc	$\Delta_{\text{best-direct}}$
Proprietary APIs					
Claude Opus 4.7	**45.7**	42.9	38.1	40.0	(−7.6)
Claude Sonnet 4.6	32.4	43.8	39.0	**44.8**	(+6.6)
GPT-5	38.1	**39.0**	33.3	35.2	(−4.8)
GPT-4o	37.1	40.0	31.4	**44.8**	(−5.7)
Gemini-2.5-Pro	36.2	**43.8**	**43.8**	42.9	(+7.6)
Gemini-2.5-Flash	37.1	43.8	**46.7**	42.9	(+9.6)
Gemini-3-Flash	37.1	**49.5**	42.9	40.0	(+5.8)
Gemini-3-Pro	43.8	**45.7**	38.1	43.8	(−5.7)
Open weights					
Qwen3-VL-8B	**38.1**	32.4	35.2	21.0	(−2.9)
Qwen2.5-VL-7B	**41.9**	34.3	34.3	40.0	(−7.6)

best_cc is the row maximum for Claude Opus 4.7 (89.0%) and a strong second on most other rows, and nl_cot helps 7 of 8 proprietary APIs on 3D (gains up to +7.4% for Claude Sonnet 4.6; only Gemini-2.5-Pro slightly regresses by −0.4%). Depth/distance reasoning benefits from explicit reasoning trace in a way that 2D primitives do not. **(ii) Primitive-best is again leaky on real photos.** For Gemini-3-Pro (96.0% via Scene Lang DSL vs. 95.1% via Three.js on CV-Bench-3D), Gemini-3-Flash (94.3% via Unity C# vs. 93.6% via Canonical JSON on CV-Bench-3D), GPT-4o (83.0% DSL vs. 78.2% Three.js on CV-Bench-3D), and Claude Sonnet 4.6 (87.3% DSL vs. 86.3% JSON on CV-Bench-3D), the primitive-*worst* Code-CoT language exceeds the primitive-best on real photos, the same inversion observed on Hypersim (Section 4.2). The primitive Reconstruct-Score ranking is a useful default but not a tight predictor of Code-CoT effectiveness on cluttered real-photo benchmarks.

Table 10: Accuracy (%) under four inference modes on CV-Bench-2D (1,438 items) and CV-Bench-3D (1,200 items). `best_cc` and `worst_cc` use each model's primitive-best and primitive-worst scene-code language from Table 1, matching the convention of Tables 2 and 9. **Bold** = row maximum within each benchmark; Δ = `best_cc` − `direct` (pp), reported per benchmark.

	CV-Bench-2D (1,438)					CV-Bench-3D (1,200)				
Model	direct	nl_cot	best_cc	worst_cc	Δ	direct	nl_cot	best_cc	worst_cc	Δ
Proprietary APIs										
Claude Opus 4.7	80.4	80.6	80.3	**80.9**	(−0.1)	84.0	87.8	**89.0**	88.1	(+5.0)
Claude Sonnet 4.6	76.8	78.4	**79.0**	78.3	(+2.2)	80.4	**87.8**	86.3	87.3	(+5.9)
GPT-5	**80.9**	80.8	80.2	77.2	(−0.7)	83.8	**89.6**	87.0	80.4	(+3.2)
GPT-4o	74.4	78.4	70.6	**80.0**	(−3.8)	78.5	**83.7**	78.2	83.0	(−0.3)
Gemini-2.5-Pro	80.9	81.7	**82.0**	79.6	(+1.1)	**92.2**	91.8	89.6	89.9	(−2.6)
Gemini-2.5-Flash	**81.7**	81.4	80.8	80.5	(−0.9)	88.9	**90.9**	87.1	87.5	(−1.8)
Gemini-3-Flash	84.6	83.9	85.0	**85.3**	(+0.4)	92.8	93.6	93.6	**94.3**	(+0.8)
Gemini-3-Pro	84.1	84.8	84.1	**85.5**	(±0.0)	95.5	95.8	95.1	**96.0**	(−0.4)
Open weights										
Qwen3-VL-8B	**80.4**	79.7	75.5	70.0	(−4.9)	**92.9**	92.5	83.2	65.1	(−9.7)
Qwen2.5-VL-7B	**75.9**	53.3	74.5	73.6	(−1.4)	**82.7**	76.9	78.1	66.4	(−4.6)

J S^3-FT additional analyses

The main paper covers the pipeline (Section 5.1), the Single-view recipe with in-distribution gains (Section 5.2), and the out-of-distribution VLMEvalKit performance comparison (Section 5.3). This appendix reports the

remaining S^3-FT analyses: the structured-only format-collapse failure mode that motivates Single-view, the Cross-viewpoint augmentation variant, full per-language and per-category breakdowns, multi-seed reliability, an ablation of the design choices, the Hypersim-domain failure case, and cross-base/cross-family validation.

J.1 Format-collapse failure mode (Structured-only S^3-FT)

A naive version of the S^3-FT pipeline, which we call **Structured-only S^3-FT**, trains only on the $K = 9$ structured tasks derived in Phase 2 and uses no natural-language supervision. It improves structured outputs but *collapses* the model's language behavior: on 100 held-out primitive scenes, the mean Reconstruct Score drops from 0.280 to 0.224 for Qwen2.5-VL-7B (Three.js and Open3D fall to 0.00), and SpatialBabel-QA drops by 3.3% to 38.6%. Looking at the failures, the fine-tuned model emits scene-graph JSON for any prompt, even a free-form question like *"Is the cube above the sphere?"*, where it answers `"false"` instead of `"No"`. The 9 structured tasks dominate the training mixture and overwrite the model's prior for natural-language decoding. Single-view S^3-FT (Section 5.2) addresses this by adding rule-generated NL-QA pairs, scene-description paragraphs, and paraphrased code-gen prompts, restoring the natural-language prior while preserving the structured-task gains.

J.2 Cross-viewpoint augmentation

Several of our Phase-2 tasks (object count, classes, scene-coordinate 3D positions, scene description, NL-QA) have answers that do not depend on camera pose, so each such record can be replicated across the other three rendered viewpoints of the same scene without changing the supervision target. We call the resulting $\sim$3.5$\times$ larger dataset **Cross-viewpoint S^3-FT**. This is a supervised analog of a representation-level cross-view alignment objective: the model is shown that the same scene-coordinate answer must be produced regardless of which 2D projection it sees. Cross-viewpoint S^3-FT is the overall winner on Qwen3-VL-8B (mean Reconstruct Score **0.534**, +0.08 over baseline), but on the weaker Qwen2.5-VL-7B base it destabilizes training (canonical JSON Reconstruct Score drops from 0.48 baseline to 0.19, with only 39/100 outputs yielding ≥ 1 parsed object). For the stronger base the extra cross-view signal pays off; for the weaker one Single-view is the safer choice. We adopt Single-view as the headline recipe in the main body for that reason; cross-viewpoint numbers appear alongside in Figure 4, Table 4, and Figure 8.

J.3 Per-language S^3-FT results

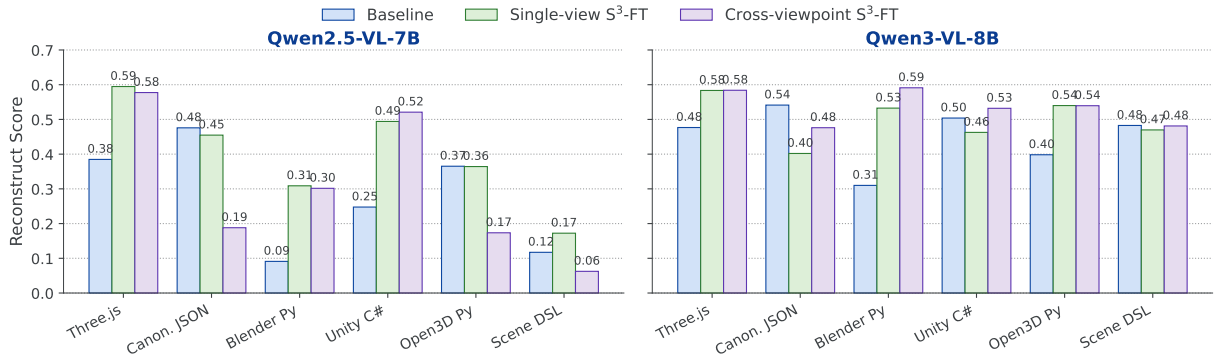

Figure 8: Per-language Reconstruct Score on 100 *primitive* held-out scenes for the two base VLMs, comparing Baseline, single-view S^3-FT (our self-supervised recipe) and cross-viewpoint S^3-FT.

The largest gains show up on languages where the base model has the most room to improve. For Qwen3-VL, cross-viewpoint S^3-FT lifts Blender Python from 0.31 to 0.59 (+0.28), Open3D Python from 0.40 to 0.54 (+0.14), and Three.js from 0.48 to 0.58 (+0.11). S^3-FT is not a universal win: for the stronger base both recipes slightly lower the Canonical JSON score (single-view 0.54 $\rightarrow$ 0.40), and for the weaker Qwen2.5-VL-7B cross-viewpoint destabilizes training (Canonical JSON collapses from 0.48 to 0.19 and Open3D from 0.37 to

0.17), so Single-view is the safer choice there. On Qwen2.5-VL's Scene Language DSL, where the base sits near the floor (0.12), single-view recovers only to 0.17.

J.4 Per-category breakdown of SpatialBabel-QA

Aggregate SpatialBabel-QA accuracy (Table 2) hides where S^3-FT actually helps. Breaking the same 652 questions per cell down by category shows that the gains are more significant in one category.

Table 11: Per-question-category accuracy (%) for Qwen3-VL-8B base vs single-view S^3-FT on SpatialBabel-Primitive-QA (100 held-out primitive scenes, 652 questions per cell across four categories). Δ is in percentage points. Bold marks rows where S^3-FT improves by more than 5 percentage points.

	direct			nl_cot			code_cot (threejs)		
Category	base	single-view	Δ	base	single-view	Δ	base	single-view	Δ
relationship	35.7	41.7	(+6.0)	29.0	43.0	(+14.0)	35.0	44.9	(+9.9)
counting	72.0	79.3	(+7.3)	61.0	79.3	(+18.3)	74.4	76.8	(+2.4)
existence	97.9	92.2	(−5.7)	88.7	91.5	(+2.8)	83.0	80.9	(−2.1)
comparison	3.5	41.7	(+38.2)	36.5	40.0	(+3.5)	12.2	39.1	(+26.9)

Two observations from the breakdown. (i) Across direct, NL-CoT, and Three.js Code-CoT, the single-view S^3-FT gains on Qwen3-VL-8B are largest in a single category: **comparison** questions go from 3.5% to 41.7%, a +38% lift on direct answering and +27% on Three.js Code-CoT. (ii) NL-CoT *counting* (61.0% → 79.3%, +18.3%) and *relationship* (29.0% → 43.0%, +14.0%) improve substantially, suggesting structured `object_count` and spatial-relation supervision in Phase 2 transfers to free-form NL reasoning under CoT prompting.

Qwen2.5-VL-7B follows the same direct/cc pattern at smaller magnitude. *Comparison under direct* lifts ∼31% →∼41% (+10%, smaller because the Qwen2.5-VL-7B baseline already starts much higher on this category, leaving less room to improve); *counting under direct* stays near 79% across base and S^3-FT. Qwen2.5-VL-7B's NL-CoT improves modestly (+3.8% overall) under the mode-agnostic evaluator, smaller than Qwen3-VL-8B's +10.3% lift, consistent with Qwen2.5-VL-7B's lower-quality base reconstructions.

J.5 Multi-seed reliability

We train Qwen3-VL-8B single-view with three random seeds (0, 42, 1337) under identical hyperparameters. All three runs converge to similar val losses (0.215 to 0.243).

Table 12: Single-view S^3-FT on Qwen3-VL-8B across three independent random seeds (0, 42, 1337), reported as SpatialBabel-Primitive-QA-Score (%) per inference mode (the same metric as Table 3). All four modes are 3-seed runs; seed std is the across-seed sample standard deviation.

Mode	base (%)	single-view S^3-FT seed mean (%) (n)	seed std (pp)	Δ over base (pp)
direct	48.0	57.7 (3)	0.4	$(+9.7 \pm 0.4)$
nl_cot	47.2	57.6 (3)	1.7	$(+10.4 \pm 1.7)$
best_code_cot (canonical_json)	46.0	52.1 (3)	2.7	$(+6.1 \pm 2.7)$
worst_code_cot (blender_py)	36.0	54.0 (3)	4.8	$(+18.0 \pm 4.8)$

Across-seed standard deviation stays at or below 2.7 pp on the three non-collapse modes and 4.8 pp on primitive-worst Blender Code-CoT. Under the mode-agnostic evaluator (Section 4.1), single-view S^3-FT lifts SpatialBabel-QA by +6.1 to +10.4% across those three modes on Qwen3-VL-8B (mean over all four ∼+11%); the primitive-worst Blender mode gains +18.0% by repairing the base model's repetition collapse in that verbose language (Section 4).

J.6 IFEval (text-only) performance comparison

To confirm S^3-FT does not trade spatial ability for general instruction-following, we run the models on IFEval (Zhou et al., 2023): 541 prompts with programmatic verifiers for format, length, keyword and language constraints (Table 13). Prompts are text-only.

Table 13: IFEval (text-only) instruction-following accuracy (%) for both S^3-FT bases, reported at prompt-level and instruction-level strict accuracy; Δ (in percentage points) is relative to the base model. S^3-FT is essentially free for the stronger Qwen3-VL-8B base and costs 3 to 6% for the weaker Qwen2.5-VL-7B base.

Base model	Baseline	cross-viewpoint S^3-FT	single-view S^3-FT
Qwen2.5-VL-7B (prompt)	62.7%	56.4% (−6.3)	57.3% (−5.4)
Qwen2.5-VL-7B (instr.)	71.3%	66.2% (−5.1)	67.6% (−3.7)
Qwen3-VL-8B (prompt)	83.5%	82.1% (−1.4)	83.0% (−0.5)
Qwen3-VL-8B (instr.)	89.0%	87.8% (−1.2)	88.5% (−0.5)

For the stronger base (Qwen3-VL-8B), S^3-FT is essentially free on IFEval. For the weaker base (Qwen2.5-VL-7B) the cost is larger (5 to 6% on single-view/cross-viewpoint) because the natural-language supervision targets are structured-but-not-quite-English and dampen the model's prior for long free-form prose. The IFEval delta is small relative to the spatial gain: Qwen2.5 single-view loses 5.4% on IFEval but lifts the mean Reconstruct Score by 0.12 (a 42% relative gain).

J.7 Cross-base and cross-family validation

To test how much of the recipe carries over to (i) a smaller capacity and (ii) different model families, we re-run the full pipeline on **Qwen2.5-VL-3B-Instruct**, **InternVL3-8B**, and **LLaVA-OV-7B**, alongside the two headline bases; Table 14 reports all five.

Table 14: Single-view S^3-FT across five base VLMs: two headline bases (Qwen3-VL-8B, Qwen2.5-VL-7B) and three cross-family / cross-scale validation bases. Each cell is "base → single-view (Δ)" on programmatic SpatialBabel-QA (100 *primitive* held-out scenes, 652 questions per cell), reported as accuracy in %; Δ is in percentage points. We report three Code-CoT scene-code languages for every base, Three.js, Canonical JSON, and Blender Python, chosen to span a compact format (Canonical JSON) and a verbose one (Blender Python).

Base VLM	direct	nl_cot	cc(threejs)	cc(canonical_json)	cc(blender_py)
Qwen3-VL-8B	48.0 → 57.4 (+9.4)	47.2 → 57.5 (+10.3)	46.3 → 55.7 (+9.4)	46.0 → 51.5 (+5.5)	36.0 → 54.6 (+18.6)
Qwen2.5-VL-7B	55.7 → 58.9 (+3.2)	53.8 → 57.7 (+3.8)	54.6 → 56.3 (+1.7)	58.0 → 47.2 (−10.7)	53.8 → 56.9 (+3.1)
Qwen2.5-VL-3B	44.6 → 47.7 (+3.1)	51.2 → 47.4 (−3.8)	45.6 → 46.0 (+0.5)	28.7 → 40.8 (+12.1)	22.9 → 48.2 (+25.3)
InternVL3-8B	46.0 → 53.2 (+7.2)	50.5 → 53.8 (+3.4)	49.5 → 41.4 (−8.1)	57.4 → 51.1 (−6.3)	46.3 → 52.6 (+6.3)
LLaVA-OV-7B	49.1 → 52.9 (+3.8)	45.6 → 52.9 (+7.4)	56.3 → 51.4 (−4.9)	60.3 → 52.5 (−7.8)	58.9 → 51.8 (−7.1)

Three patterns. (i) **Direct mode lifts uniformly across bases.** Every base in the table gains on direct: Qwen3-VL-8B +9.4%, InternVL3-8B +7.2%, LLaVA-OV-7B +3.8%, Qwen2.5-VL-7B +3.2%, Qwen2.5-VL-3B +3.1%. This is the cleanest S^3-FT signal in the cross-base sweep. (ii) **Code-CoT gains are largest where the base's Code-CoT was weakest.** The biggest lifts land on cells where the base scored low because it collapsed into verbose repetition before reaching a final answer: Qwen2.5-VL-3B canonical_json +12.1% (base 28.7%) and Blender Python +25.3% (base 22.9%), and Qwen3-VL-8B Blender Python +18.6% (base 36.0%); S^3-FT teaches these models to terminate their reconstructions and commit to an answer. Conversely, when the recipe hurts Code-CoT (LLaVA-OV-7B and InternVL3-8B), the drops fall on the language each base was already strongest at, where same-distribution distillation adds little and can overwrite what the model already did well. (iii) **NL-CoT lifts for nearly all bases.** S^3-FT improves NL-CoT for four of five bases (Qwen3-VL-8B +10.3%, LLaVA-OV-7B +7.4%, Qwen2.5-VL-7B +3.8%, InternVL3-8B +3.4%); only Qwen2.5-VL-3B regresses (−3.8%). The recipe transfers across model architectures: LLaVA-OV-7B uses a

different visual encoder and patch-merger from Qwen-VL or InternVL, yet the same pipeline lifts its direct- and NL-CoT-mode SpatialBabel-QA without changes.

J.8 Hypersim: when the self-generated training data is too incomplete

We rebuild the full single-view S^3-FT pipeline on SpatialBabel's 200 Hypersim indoor scenes: Phase-1 pseudo-GT generation with the same two base VLMs, Phase-2 annotation extraction, Phase-3 LoRA fine-tuning on 180 scenes with 20 held out for evaluation.

Table 15: Single-view S^3-FT on Hypersim (20 held-out scenes, 105 questions per cell), reported as SpatialBabel-Hypersim-QA-Score (accuracy, %) per inference mode; Δ is in percentage points. Under the mode-agnostic evaluator, S^3-FT lifts direct and NL-CoT for both Qwen models on Hypersim but does not help Code-CoT, shown here in the trained Three.js format. In Code-CoT formats absent from the S^3-FT training distribution (canonical_json, Blender Python) the reconstruction step degenerates into malformed or near-empty output and the answer collapses, so we report the trained-format Three.js Code-CoT as the fair comparison.

Qwen3-VL-8B (Hypersim)	Base	single-view S^3-FT	Δ
direct	38.1	41.9	(+3.8)
nl_cot	32.4	40.0	(+7.6)
code_cot (threejs)	33.3	32.4	(−0.9)
Qwen2.5-VL-7B (Hypersim)	Base	single-view S^3-FT	Δ
direct	41.9	41.9	(±0.0)
nl_cot	34.3	43.8	(+9.5)
code_cot (threejs)	44.8	33.3	(−11.5)

Code-CoT regresses because the pseudo-ground-truth is too incomplete. Single-view S^3-FT still helps on Hypersim where the supervision is reliable: direct improves for Qwen3-VL-8B (+3.8%), and nl_cot improves for both bases (+7.6% and +9.5%). Code-CoT, which first reconstructs the scene and then answers, does not improve in the trained Three.js format (−0.9% for Qwen3-VL-8B, −11.5% for Qwen2.5-VL-7B), and in formats absent from the training distribution (canonical_json, Blender Python) the reconstruction step degenerates into malformed or near-empty output and the answer collapses. The reason is the quality of the Phase-1 pseudo-ground-truth on cluttered indoor scenes, not a weakness of the recipe. The pipeline is the same as for primitives and the pseudo-GT pass rate is comparable (68% to 58%), but the pseudo-GT code is far less faithful to the scene: the base VLMs' Three.js reconstructions contain only about 9 to 10 objects per Hypersim scene, while the ground-truth scene graphs have a median of 26 objects (mean 66, IQR 11 to 58). A reconstruction that captures less than a third of the objects is too incomplete to use as a training label, so the structured Code-CoT tasks are trained on systematically incomplete targets. The bottleneck is therefore pseudo-ground-truth coverage on cluttered scenes, not the recipe itself: with more faithful reconstructions (from a stronger base model or multi-pass aggregation), the same pipeline should transfer to complex scenes.

This is a qualitative phase transition. On SpatialBabel primitives, pseudo-GT covers ∼89% of scene content (8.4 vs 9.4 objects/scene) and the recipe produces large gains. On Hypersim coverage is only ∼38% (9.8 vs 26) and the same recipe yields no gain or regresses. We conclude S^3-FT's headline result is *conditional on the teacher VLM's code-generation ability spanning the target domain's scene complexity*; beyond that threshold, the "oracle" in "pseudo-oracle" is no longer an oracle.

Two paths forward: (i) a stronger teacher at the cost of self-supervision; (ii) FOV-aware target derivation at training time so Phase-2 tasks only cover visible objects, which raises pseudo-GT coverage to ∼0.16 of the in-frustum portion (still low, but cleaner training signal). We list both as follow-ups.

K Broader Impacts

Stronger spatial understanding in VLMs is a foundational capability for embodied agents (robotic manipulation, autonomous navigation), assistive technology (scene description for blind and low-vision users), AR/VR

scene authoring, and educational tools that explain 3D structure from a single image. The self-supervised nature of S^3-FT (which requires *no human spatial labels, depth sensors, or teacher-model distillation*) lowers the cost and labour barrier for fine-tuning open-weights VLMs, broadening access for academic groups and small organizations that cannot afford large-scale annotation pipelines or proprietary teachers. Releasing the benchmark, training data, parser toolkit, and LoRA adapters under permissive licenses upon publication further democratizes spatial-VLM research. Potential misuses (e.g., surveillance) are mitigated by the benchmark containing no human imagery and the LoRA-only release allowing downstream users to opt-out by simply not loading the adapter.

L Reproducibility appendix

We include the following to support reproduction of every numeric result in the paper.

L.1 S^3-FT hyperparameters

The LoRA fine-tuning configuration, shared across both base models, is given in Table 16.

Table 16: S^3-FT LoRA fine-tuning hyperparameters (shared across both base models).

Hyperparameter	**Value**
Base models	`Qwen/Qwen2.5-VL-7B-Instruct`, `Qwen/Qwen3-VL-8B-Instruct`
LoRA	rank 16, $\alpha = 32$, dropout 0.05, target modules {q,k,v,o}_proj
Optimizer	AdamW, lr 2e−4, weight decay 0
Schedule	cosine decay, no warmup
Batch / accumulation	batch size 1, gradient accumulation 8 (effective 8)
Epochs	3 for structured-only (deprecated) and single-view S^3-FT; 2 for cross-viewpoint (larger data)
Sequence length	2048 tokens max
Mixed precision	bfloat16
Checkpoint selection	best validation loss over epochs; final model saved additionally

L.2 Phase-1 quality filters

Generated code is accepted only if it contains all three tokens {`THREE.Scene`, `THREE.PerspectiveCamera`, `THREE.Mesh`}, has 1 to 50 `new THREE.Mesh(` occurrences, and all `position.set(x,y,z)` values lie in $[-30, 30]$. Pass rates: Qwen3-VL-8B 82%, Qwen2.5-VL-7B 84%.

L.3 Phase-2 task mixture

single-view S^3-FT (per scene): 9 JSON task records + 1 scene description (free-form) + 8 NL-QA pairs (relationship, counting, existence, comparison via rule-based generator) + 2 extra code-generation prompt paraphrases. Dataset size after Phase-1 filtering: ∼3.5k records per model. Cross-viewpoint S^3-FT replicates every view-invariant record across the four rendered viewpoints, ∼3.5× data growth.

L.4 SpatialBabel eval protocol

The test set is 100 held-out primitive scenes (pre-filtered from the Phase-1 candidate pool to avoid leakage). Generation uses temperature 0.0, `max_new_tokens` 2048, and no beam search. Scoring is parser-based: each generated code string is parsed into a `Scene` object, Hungarian-matched against the ground truth on position, and graded with the SpatialBabel-Reconstruct-Score (parse rate, localization-gated detection F1, class accuracy, and scale fidelity; Section 3.2). Confidence intervals are 95% bootstrap over scenes with 1,000 resamples.

L.5 Public VLM benchmark protocol

VLMEvalKit, generation-based evaluation with `use_custom_prompt=True` (standard MCQ template) for every configuration. Baseline models use the same wrapper; S^3-FT adapters are loaded via PEFT at model-init time. Fallback to `sdpa` attention when flash_attn is unavailable; we verified no numeric difference on POPE / OCRBench. Full per-benchmark numbers in `results/vlmeval/final_summary.json` in the code release.

L.6 Software versions

transformers $\geq$4.46, peft $\geq$0.13, torch 2.x, VLMEvalKit main@2026-04. numpy pinned to $<$2 for compatibility with downstream pandas used in VLMEvalKit. IFEval scoring via the `lm_eval.tasks.leaderboard.ifeval` module (official Google-Research port).

L.7 Data splits

Primitive scenes are split deterministically: 1,000 total, 100 held out for evaluation (stratified across tiers 1 to 5), 467 used for Phase-1 pseudo-GT generation (the remaining $\sim$430 are kept unused as a safety margin). Train / val for Phase-3 LoRA is a further 90/10 split of the Phase-1 pseudo-GT survivors (seed 42), grouped by scene so no scene appears in both.

We commit two aggregated snapshot files: `results/benchmark/aggregate.json` (the wide table read by every figure script) and `results/vlmeval/final_summary.json` (the wide table read by Table 4). These contain the per-cell aggregate metrics needed to regenerate every table and figure in the paper.

Availability. Upon publication, we will publicly release the benchmark, the evaluation and training code, the S^3-FT checkpoints, and the aggregated results needed to reproduce the tables and figures in this paper.