WHEN VISUAL PROGRAMS FAIL SILENTLY: AN EXECUTION-GUIDED REPAIR AGENT FOR SPATIAL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Visual program synthesis makes spatial reasoning inspectable, but an incorrect perception output can pass through a well-formed program without raising an exception. We present **PitCrew**, an execution-guided repair framework that turns failed perception calls and inconsistent intermediate results into actionable program revisions. A question-conditioned Scene Captioner provides supplementary visual context for synthesis. The Program Executor records tool calls, and a deterministic Consistency Checker organizes answer-validity failures and disagreements with the scene context into feedback for a Repair Agent. The Repair Agent uses the failed program, execution trace, and scene context to localize the issue and revise perception queries or program logic within a bounded budget. Mask-aware depth and size operators support both initial and repaired executions. PitCrew achieves 49.8% on Omni3D-Bench and 89.4% on CLEVR, improving on the original VADAR configuration by 9.4 and 35.8 percentage points without task-specific training. Matched-backbone controls and component ablations support the utility of the combined system.

1 INTRODUCTION

Visual program synthesis (Gupta & Kembhavi, 2023; Surís et al., 2023; Subramanian et al., 2023; Lu et al., 2023; Marsili et al., 2025) decomposes a visual question into perception calls and explicit computation. Intermediate detections, depths, and measurements can be inspected, making the computation easier to diagnose than a final answer alone. This transparency, however, does not establish that the intermediate visual observations are correct.

The distinction matters for single-view spatial reasoning. Estimating an object's size relative to another may require locating both objects, identifying their surfaces, estimating depth, and combining measurements. A missed target prevents the computation; a spurious box or inaccurate depth can instead produce a plausible but wrong result. These difficulties complement the spatial-reasoning limitations documented for end-to-end VLMs (Kamath et al., 2023; Tong et al., 2024; Chen et al., 2024a; Cheng et al., 2024).

VADAR (Marsili et al., 2025) provides a useful foundation through question-specific helper APIs and program synthesis. Its exception-based retry can address Python execution errors, but a type-correct perception error need not raise an exception. For example, a failed localization can be converted into a default answer, while an extra box can silently change a count. Figure 1 illustrates this gap between successful execution and a correct visual answer.

A question and its operations often provide expectations against which execution can be assessed. A size comparison requires grounded operands; selecting a particular object requires resolving competing candidates. An empty required operand or unresolved selection is therefore informative even without knowing the final answer. Other expectations, such as the number of visible objects, need additional visual evidence. We use a question-conditioned scene description as supplementary context for program synthesis and as a fallible reference for those checks. The distinction is essential: a discrepancy suggests where to investigate, but does not identify which source is correct.

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

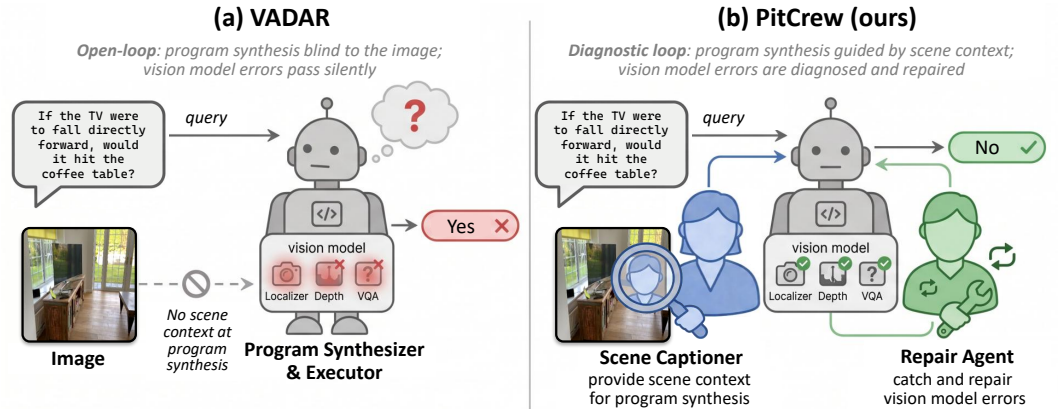


Figure 1: **Motivation: a plausible return can conceal a perception error.** Exception-based retry need not activate when an incorrect detection still produces a valid answer type. PitCrew uses scene context and execution feedback to diagnose the failing perception step and revise the program within a bounded repair loop.

We instantiate this approach as **PitCrew**. Its Scene Captioner supplies relevant entities, suggested query terms, spatial cues, and auxiliary estimates before code generation. The Program Synthesizer uses this context to construct a program, and the Program Executor records perception calls in a trace. Answer-validity checks and context-dependent discrepancies support a bounded repair loop. The Consistency Checker is deterministic code, while the Repair Agent is an LLM that proposes a revised program. It grounds each proposed repair in the failed program code, execution logs, and scene context, using the observed failure to guide changes to perception queries or program logic. A separate Spatial Perception API improves the depth and size values used by either execution path. A terminal fallback handles questions unresolved within the repair budget.

PitCrew connects three elements: execution evidence that identifies an unresolved computation, targeted query or logic edits conditioned on that evidence, and re-execution that tests the proposed repair. Missing objects, spurious detections, and cardinality disagreements motivate the checks, making the interface applicable across questions and perception calls. We evaluate this design on natural photographs and synthetic scenes through matched-backbone controls, component ablations, and trace-level analysis of successful and unsuccessful repairs.

PitCrew improves Omni3D-Bench from 40.4% to 49.8% and CLEVR from 53.6% to 89.4% relative to the original VADAR configuration. At a matched GPT-5-mini synthesis backbone, the Omni3D gain is 8.8 percentage points. Component ablations quantify the contributions of repair, scene conditioning, and spatial operators, while trace analysis characterizes recovery behavior and inference cost.

2 Related Work

Visual program synthesis. VisProg (Gupta & Kembhavi, 2023), ViperGPT (Surís et al., 2023), CodeVQA (Subramanian et al., 2023), and Chameleon (Lu et al., 2023) compose pretrained tools through generated programs. VADAR (Marsili et al., 2025) extends this approach with question-specific API retrieval and construction. Visual Program Distillation (Hu et al., 2024) instead transfers program-based reasoning into a VLM. PitCrew builds on VADAR's executable pipeline and focuses on feedback when a perception return is plausible enough to avoid exception-based retry.

Feedback and program revision. ReAct (Yao et al., 2023), MM-ReAct (Yang et al., 2023b), AssistGPT (Gao et al., 2023), and IdealGPT (You et al., 2023) study reasoning with tools and intermediate feedback. HYDRA (Ke et al., 2024) uses feedback to adapt compositional reasoning, while NAVER (Cai et al., 2025) introduces explicit logical structure for visual grounding. Self-Debugging (Chen et al., 2024b), Reflexion (Shinn et al., 2023), Self-Refine (Madaan et al., 2023), LATS (Zhou et al., 2023), CRITIC (Gou et al., 2024), and CodeT (Chen et al., 2023) investigate execution, reflection, search, or external feedback for improving generated outputs. PitCrew exam-

ines a specific interface within this broader setting: explicit discrepancies between visual-program observations and available expectations, followed by query or logic revision.

Visual grounding and spatial reasoning. Hallucination evaluation and correction (Li et al., 2023; Yin et al., 2024) and visual search or grounding (Wu & Xie, 2024; Yang et al., 2023a) address related failures in visual evidence. What's Up (Kamath et al., 2023) and Cambrian-1 (Tong et al., 2024) motivate stronger spatial understanding; SpatialVLM (Chen et al., 2024a) and SpatialRGPT (Cheng et al., 2024) use 3D-grounded training. SQA3D (Ma et al., 2023) and 3D-LLM (Hong et al., 2023) consider richer 3D inputs, while VSI-Bench (Yang et al., 2025) evaluates spatial understanding from video. PitCrew operates at inference time on a single RGB image. Its evaluation uses Omni3D-Bench (Marsili et al., 2025) and CLEVR (Johnson et al., 2017).

3 METHOD

PitCrew extends a visual-program pipeline with execution checks and targeted repair (Figure 2). We first introduce the baseline and problem setting, then the supplementary scene context, the checking-and-repair loop, and the supporting spatial operators.

3.1 BASELINE PIPELINE AND PROBLEM SETTING

Visual programs. Given an image I and question q, a program P invokes perception operators $\mathcal{O}$ and produces an answer $\hat{a}$. We retain VADAR's pipeline: the Signature Agent specifies function interfaces, the API Construction Agent prepares helper functions, and the Program Synthesizer composes a Python program executed by the Program Executor. VADAR retries execution exceptions; PitCrew also targets failures exposed by program checks on perception returns. PitCrew additionally supplies question-conditioned scene context H and records the execution trace τ. The ground-truth answer $a^\star$ is used only for evaluation.

Silent failures and observable discrepancies. A silent failure occurs when a perception or program-logic error affects the answer without raising an execution exception. PitCrew makes unresolved computations explicit through program-level checks and failure sentinels. An answer-validity gate $G(\hat{a}, q)$ then tests whether the returned answer is admissible for the question. False-negative detections, false-positive detections, and count errors are general perception failure modes, but their ground-truth labels are unavailable at runtime. PitCrew instead uses observable proxies: missing required operands, failure sentinels, and disagreements with available scene references. Checks follow the requirements of the operation: empty localization is a failure for a required measurement operand, whereas zero counts and negative existence answers are admissible. Multiple detections require disambiguation when the computation needs a unique referent.

3.2 SCENE CAPTIONER

The Scene Captioner provides a question-conditioned description that focuses H on relevant entities and relations. Its object inventory includes visibility descriptions and suggested detector query terms; optional fields supply instance descriptions, a count estimate, relative depth ordering, and anticipated perception challenges. These are textual cues, not bounding-box coordinates. Scene-context instructions in the Program Synthesizer's prompt guide their use in choosing queries and operations (Figure 10); the captioning model is specified in Section 4.1.

For example, the captioner describes a white coffee table and a black side table as visible, suggests "table" as a query term for both, and flags perspective and reflections as measurement challenges (Figure 5). This context identifies relevant objects and possible ambiguities before tool execution; Appendix B.3 unpacks the example. A *count estimate* supplies a cardinality reference for the Consistency Checker. The counting configuration and answer routes are detailed in Appendix B.3.

Scene descriptions guide which visual evidence to seek, but can omit or misidentify objects. The synthesis prompt therefore asks the program to check object presence with localization and, when needed, VQA even when H suggests absence. Object-inventory and mismatched-context ablations support the utility of relevant descriptions (Table 4); the no-context configuration shows that the checking-and-repair pipeline remains usable without them (Table 2). Shared omissions can still escape both sources, as illustrated in Appendix D.4.

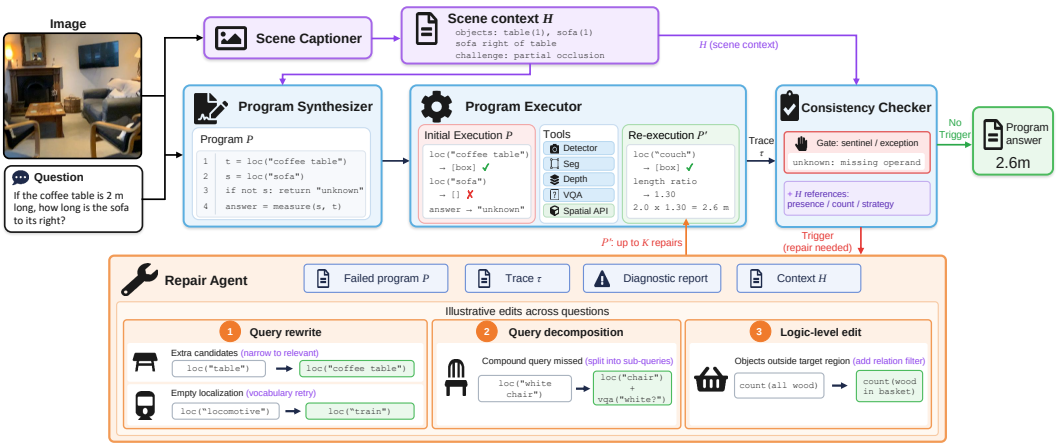


Figure 2: **PitCrew overview and illustrative program repairs.** Supplementary scene context H guides synthesis and diagnosis. The example connects a missing localization to repair and re-execution; the lower panel shows alternative query rewrites, query decomposition, and logic-level edits across questions. The Repair Agent uses the failed program, trace τ, and available context and diagnostic report to produce P'. "No Trigger" denotes acceptance by the available checks. Program excerpts and numerical values are schematic.

3.3 Consistency Checker and Repair Agent

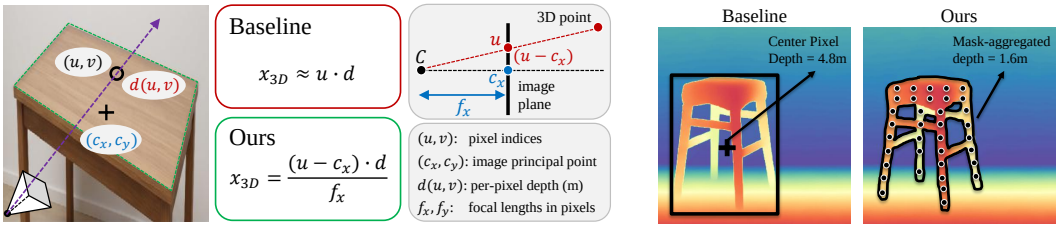
Execution evidence and expectations. The Program Executor records each perception call, its arguments and returned value, and cache status in τ. The question and the requested operation define what the program needs to compute an answer. For example, measuring an object's size requires its localization; if the required operand is missing, the generated program can return `unknown`. When an operation needs a unique referent, unresolved selection can similarly be exposed by a program-level check. These expectations concern the computation and do not require a scene description.

Consistency Checker. The core checking path connects program-level failures to repair. The answer-validity gate tests the returned answer against the configured answer type and failure sentinels; a failed gate or execution exception sends the program and trace to the Repair Agent. Thus, an empty required operand can initiate repair without H, provided the program exposes it as a failure. Legitimate zero counts and negative existence answers remain admissible. The trace then shows which calls and values led to the failure, allowing the Repair Agent to revise queries or logic even without supplementary scene context.

When H is available, the checker adds a diagnostic report comparing relevant localization records with the description. Its four report types are: *visibility miss*, when an object described as visible has no localization; *visibility false positive*, when an object described as absent has detections; *count mismatch*, when reference and localization counts disagree; and *strategy ignored*, when execution bypasses an indicated strategy. These comparisons supply additional leads for repair, such as a candidate missed object or a count discrepancy, rather than prerequisites for the repair loop. The strategy signal is advisory because an alternative strategy may also be valid.

The checks use deterministic rules and require no additional judging-model call. Program failures and context discrepancies are complementary evidence: the former can activate repair independently of H, while the latter help interpret a flagged execution when suitable references exist. Appendix B details the trigger and diagnostic stages, including the scope of each check.

Targeted program repair. The Repair Agent receives the question, current program, execution trace, answer or exception, and available context and discrepancy report. A fresh LLM call proposes a complete revised program, with a directive describing the intended change (Figure 9). The choice of edit follows the suspected failure. *Query rewrite* can narrow `loc('table')` to `loc('coffee table')` when the returned candidates appear to include irrelevant tables. Conversely, an empty `loc('locomotive')` can motivate a vocabulary retry with `loc('train')`. *Query decomposition* addresses a possible miss from a compound query: if `loc('white`



(a) Per-pixel 3D backprojection.

(b) Mask-aggregated depth.

Figure 3: **Supporting spatial operators.** (a) VADAR's image-plane size approximation versus calibrated per-pixel backprojection, as defined in Equation 2. (b) VADAR's center-pixel depth sample may fall on an occluder or background; mask aggregation uses the estimated object surface. Displayed depths are predictions, not ground truth.

`chair')` returns no required object, the agent can use `loc('chair')` followed by `vqa('is it white?')` on each candidate. *Logic-level edit* revises selection, control flow, or aggregation, such as restricting a count to objects inside the queried region. These are alternative edits, illustrated across questions in Figure 2, rather than a fixed sequence. Fresh prompting supplies the failed execution separately from the synthesis conversation; the revised program is then executed and checked within the same bounded loop.

Budget, cache, and terminal fallback. The initial execution is followed by at most K additional repair executions. A trace-aware cache reuses compatible perception results; changed queries create new calls, and affected entries may be invalidated. Once this budget is exhausted, a terminal fallback supplies a best-effort response. For this purpose, the Scene Captioner also precomputes a *draft answer*: a candidate response to the question. The draft block is also visible to synthesis and repair as evidence and strategy guidance, but the prompt instructs programs to compute their answer rather than copy the candidate. Direct integer-count fallback can reuse the count estimate; other questions use the configured image–question VLM, with the cached draft answer available when that call is unavailable or fails to return an answer (Appendix B.3). This model-dependent response closes the bounded execution loop; aggregate system results include the fallback path.

3.4 Spatial Perception API

The Spatial Perception API (Spatial API) refines two measurements in the VADAR pipeline: size estimation from an image-plane bounding box scaled by depth, and depth estimation from the bounding-box center. Even after repair recovers the intended object, these approximations can distort the values used by the program. We replace them with per-pixel backprojection and mask-aggregated depth while retaining the existing depth and segmentation models (Figure 3). Both initial and repaired programs use these operators.

Per-pixel backprojection. An image-plane bounding box summarizes projected extent and can include substantial area outside the visible object. We instead use calibrated pinhole backprojection (Hartley et al., 2003) on mask pixels with per-pixel depth, then measure quartile-trimmed spans along the camera-coordinate x and y axes. This uses the estimated object surface and its depth variation for size estimation. Equation 2 in Appendix B specifies the geometry.

Mask-aggregated depth. A bounding-box center can fall on background through a gap in the object, or on a foreground occluder. We replace this single sample with the median of in-mask depth samples after quartile trimming, reducing dependence on the center pixel.

These geometric operations complement query and logic repair by measuring the estimated visible surface. Their accuracy depends on segmentation, depth, and camera calibration. Section 4.3 evaluates their contribution separately from the repair mechanism.

Table 1: **Main results on Omni3D-Bench and CLEVR (%).** Bold marks the highest score in each column among non-grey entries, including ties. †InternVL's CLEVR results are greyed and excluded from highlighting because related CLEVR-family data appear in earlier InternVL training recipes (Appendix A.1). Evaluation settings are described in Section 4.1; scoring and variability are detailed in Appendices A.1 and F.

	Omni3D-Bench					CLEVR			
Method	Y/N	MC	Count	Other	Total	Y/N	MC	Count	Total
Proprietary VLMs									
GPT-4o (Hurst et al., 2024)	66.7	57.2	28.1	35.5	42.9	63.0	60.0	52.3	58.4
GPT-5-mini (Singh et al., 2025)	68.0	68.6	27.9	43.1	49.1	79.0	78.4	68.0	75.0
GPT-5.1 (Singh et al., 2025)	**74.7**	70.9	30.0	42.8	50.6	89.7	92.4	73.8	85.0
Claude Sonnet 4.6 (Anthropic, 2026)	68.0	59.3	32.9	37.8	45.3	92.7	**99.7**	**92.5**	94.8
Claude Opus 4.6 (Anthropic, 2026)	65.3	**72.1**	30.0	40.7	48.3	**94.5**	**99.7**	91.3	**95.0**
Gemini 2.5 Flash (Comanici et al., 2025)	60.0	69.8	25.7	33.9	42.8	68.2	63.2	55.0	62.1
Gemini 2.5 Pro (Comanici et al., 2025)	69.3	67.4	30.0	**45.5**	**50.7**	84.0	78.4	72.5	78.3
Open-weight VLMs									
InternVL-3.5-8B† (Wang et al., 2025)	64.0	59.3	28.6	25.9	37.7	92.2	98.9	91.8	94.1
Qwen3-VL-8B (Bai et al., 2025)	50.7	68.6	15.7	30.3	37.9	89.4	92.4	81.2	87.5
Qwen3-VL-32B (Bai et al., 2025)	62.6	68.6	31.4	32.9	43.3	92.9	93.5	89.7	92.0
Qwen3.5-35B-A3B (Qwen Team, 2026)	62.7	62.8	28.6	34.0	42.5	93.7	95.8	82.3	90.4
Qwen3.5-122B-A10B (Qwen Team, 2026)	64.0	67.4	**38.6**	37.3	46.7	93.7	97.2	83.0	91.1
Gemma-4-26B-A4B-it (Team et al., 2026)	57.3	57.0	18.6	31.9	38.1	56.9	51.7	48.0	52.2
Program-based methods									
VisProg (Gupta & Kembhavi, 2023)	54.7	25.9	2.9	0.9	13.5	48.4	28.3	16.7	31.2
ViperGPT (Surís et al., 2023)	56.0	42.4	20.0	15.4	26.7	43.4	13.4	20.5	26.2
VADAR (Marsili et al., 2025)	56.0	57.6	21.7	35.5	40.4	65.3	40.8	53.3	53.6
PitCrew (ours)	68.0	66.3	35.7	43.2	49.8	92.0	93.0	83.5	89.4

4 Experiments

4.1 Setup

Benchmarks and comparisons. Our Omni3D-Bench evaluation uses 500 questions from VADAR's benchmark (Marsili et al., 2025), built on Omni3D photographs (Brazil et al., 2023), covering Yes/No, multiple choice (MC), integer count, and other numerical spatial answers. CLEVR (Johnson et al., 2017) tests compositional reasoning on synthetic scenes. Table 1 compares program-based methods with proprietary and open-weight direct VLMs, including Qwen3.5 (Qwen Team, 2026) and Gemma-4 (Team et al., 2026). The CLEVR component study separately uses a fixed, type-balanced subset of 300 questions. Following VADAR's evaluation protocol, we report mean relative accuracy (MRA) for Omni3D's numerical "Other" answers, averaging accuracy over relative-error thresholds from 5% to 50%; the remaining categories use answer matching (Appendix A.1).

Implementation and reporting. The Signature Agent, API Construction Agent, Program Synthesizer, and Repair Agent use GPT-5-mini (Singh et al., 2025) in the canonical configuration; each is a separate prompted call. The Scene Captioner uses the pretrained VLM, Qwen3-VL-8B (Bai et al., 2025), to produce question-conditioned descriptions and auxiliary estimates. The perception stack uses GroundingDINO-T (Liu et al., 2024) for Omni3D localization, Molmo-7B (Deitke et al., 2025) for CLEVR localization, Qwen3-VL-4B (Bai et al., 2025) for VQA, UniDepthV2 (Piccinelli et al., 2025) for depth, and SAM2-B+ (Ravi et al., 2025) for segmentation. VADAR in Table 1 uses its original GPT-4o configuration; Table 3 provides matched-backbone comparisons. We use $K = 3$ additional repair attempts. Repeated-run results use the median of three runs; score variability is reported in Appendix F.

4.2 Main Results

On Omni3D-Bench, PitCrew reaches 49.8% against VADAR's 40.4% (+9.4 pp), ViperGPT's 26.7%, and VisProg's 13.5%. The gain over the original VADAR is present in all four categories: Yes/No +12.0, MC +8.7, Count +14.0, and Other +7.7 pp. On CLEVR, PitCrew reaches 89.4% versus

Table 2: **Leave-one-component-out ablation (%).** The code-synthesis backbone is GPT-5-mini. Omni3D uses 500 questions; CLEVR uses a fixed subset of 300 questions (100 per type, seed 42), distinct from Table 1. Full factorials and category scores are retained in Appendix C.2.

Configuration	Omni3D-Bench	CLEVR (300)
Full PitCrew	**49.8**	**91.3**
w/o Repair	44.7	80.0
w/o Spatial Perception API	47.1	87.3
w/o Scene Context	43.2	78.7

Table 3: **Backbone and configuration controls (%).** Matched synthesis backbones on Omni3D (500 questions), plus the all-off/full comparison on the fixed CLEVR subset (300 questions). The CLEVR row holds GPT-5-mini fixed; it is not a cross-backbone comparison.

Cohort	Synthesis backbone	VADAR / all-off	Full PitCrew	Gain (pp)
Omni3D (500)	GPT-4o	40.4	46.6	+6.2
Omni3D (500)	GPT-5-mini	41.0	49.8	+8.8
CLEVR (300)	GPT-5-mini	69.3	91.3	+22.0

53.6%; the category gains are +26.7, +52.2, and +30.2 pp for Yes/No, MC, and Count. The gains span object counting, spatial relations, and numerical measurement.

PitCrew has the highest total score among the evaluated program-based methods on both benchmarks. Its Omni3D score of 49.8 exceeds the listed open-weight direct VLMs (up to 46.7) and is within 0.9 pp of Gemini 2.5 Pro (50.7) and GPT-5.1 (50.6). Performance varies by category and benchmark: Qwen3.5-122B-A10B leads Omni3D Count at 38.6, and Claude Opus 4.6 leads CLEVR at 95.0. The Qwen3.5 models score 90.4/91.1 on CLEVR, also above PitCrew's 89.4. These comparisons position the full system among direct-answer alternatives; the following controls examine gains within a common program pipeline.

4.3 Component Ablation

Component removal. Table 2 shows that removing repair, the Spatial API, or scene context reduces the Omni3D score by 5.1, 2.7, or 6.6 pp. The corresponding CLEVR subset scores also decrease. The complete factorials in Appendix C.2 retain all single-component and all-off conditions. Scene+Repair reaches 47.1 on Omni3D without the Spatial API, showing that geometry improvements support the loop without being necessary for it. Removing repair costs 11.3 pp on the CLEVR subset; removing scene context costs 6.6 pp on Omni3D and 12.6 pp on CLEVR. The full combination yields the highest total on both cohorts.

Backbone controls. Table 3 shows gains with both backbones: PitCrew improves over the matched VADAR pipeline by 6.2 pp with GPT-4o and 8.8 pp with GPT-5-mini on Omni3D. Its benefit therefore extends beyond the canonical GPT-5-mini configuration. Changing the backbone adds only 0.6 pp within VADAR, compared with 3.2 pp within PitCrew. A stronger code generator alone does not add a response to perception errors that pass without exceptions. PitCrew additionally exposes failed executions to the language model for revision, providing another opportunity to use its capabilities. The larger backbone gain is consistent with this role for execution feedback, although the aggregate comparison does not separate synthesis and repair contributions. On the fixed CLEVR subset, the GPT-5-mini all-off and full configurations score 69.3 and 91.3 (+22.0 pp). Configuration and inference-cost distinctions are detailed in Appendix A.2.

What the scene context contributes. Table 4 examines the information supplied to synthesis, repair, and reference-based checks. Removing the object inventory reduces the Omni3D score from 49.8 to 44.8; removing counting analysis yields 47.6, with Count decreasing from 35.7 to 25.7. A mismatched-image context yields 43.1, close to the no-context score of 43.2. These results support the importance of relevant entities and estimates, rather than the presence of a structured prompt alone. In the evaluated integer-count configuration, the count-anchor program can return the cap-

Table 4: **Scene-context field ablations on Omni3D-Bench (%).** The modified context is shared by synthesis, repair, and reference-based checking. Removing the draft-answer block also removes its strategy/evidence but retains the separate counting prior. All rows use the same 500 questions.

Context	Y/N	MC	Count	Other	Total
Full H	68.0	66.3	35.7	43.2	49.8
w/o Object inventory	61.3	60.5	28.5	39.4	44.8
w/o Counting analysis	66.7	65.1	25.7	42.3	47.6
w/o Depth ordering	68.0	65.1	35.7	41.2	48.6
w/o Predicted challenges	68.0	66.3	34.3	42.8	49.4
w/o Draft-answer block	68.0	65.1	35.7	43.0	49.5
Mismatched-image H	62.7	57.2	24.3	38.1	43.1
No H	66.6	57.4	22.8	37.5	43.2

Table 5: **Component sensitivity on Omni3D-Bench.** (a) Program/Repair LLM; (b) Scene Captioner VLM; (c) Spatial API operators (Mask-Dep. = mask-aggregated depth, BP = per-pixel back-projection); (d) total score versus repair budget K. Canonical PitCrew in **bold**.

(a) Program/Repair

LLM	Total
GPT-4o	46.6
GPT-5-nano	44.3
GPT-5-mini	**49.8**
GPT-5.1	49.8

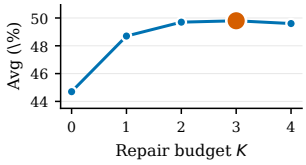
(b) Scene Captioner

VLM	Total
Qwen3-VL-4B	49.3
Qwen3-VL-8B	**49.8**
Qwen3-VL-32B	49.4
GPT-5-mini	49.7

(c) Spatial API

Mask-Dep.	BP	Total
		47.1
✓		48.7
	✓	48.8
✓	✓	**49.8**

(d) Repair budget K.



tioner's estimate directly; the counting-analysis ablation therefore changes an answer source as well as a consistency reference (Appendix B.3).

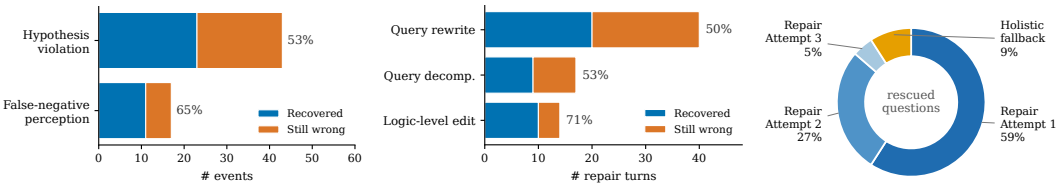
Removing the explicit draft-answer block yields 49.5, only 0.3 pp below full H, whereas removing the object inventory costs 5.0 pp. This contrast supports the value of scene descriptions for program construction and repair beyond access to a candidate answer. The intervention also removes the block's strategy and evidence while retaining the separate count prior. Field ablations thus measure information utility across the pipeline. Appendix C.1 relates these controls to captioner capacity and shared-error limitations.

Sensitivity and inference cost. Table 5 reports synthesis/repair backbone sensitivity and a narrow 49.3–49.8 range across the tested Scene Captioners. Both spatial operators improve the 47.1 configuration, reaching 49.8 together. Repair gains largely saturate within a few attempts: $K = 2$ and $K = 3$ yield 49.7 and 49.8, respectively; we use $K = 3$. The captioner adds a local pass per question (8±2 seconds), while the analyzed run averages 0.12 repair calls per question. Summing reported stage means gives approximately 19 seconds per question for PitCrew versus 5 for the VADAR pipeline, before separately unreported fallback overhead (Appendix E).

4.4 Diagnostic Coverage and Repair Strategies

In the analyzed Omni3D run ($N = 500$), repair activates on 9.6% of questions, while 90.4% complete without repair. Among the flagged questions, 45.8% are rescued (22/48). Figure 4 reports trigger and directive summaries over 60 repair attempts. The original "hypothesis violation" group accounts for 71.7% of attempts and includes visibility-FP, count-mismatch, and strategy-ignored reports; "false-negative perception" accounts for 28.3% and groups visibility-miss and gate-sentinel events. These operational labels describe diagnostic signals, not ground-truth error classifications.

Among rescued questions ($n = 22$), 90.9% are resolved through revised programs: 59.1%, 27.3%, and 4.5% at repair attempts 1, 2, and 3, respectively. Terminal fallback contributes the remaining 9.1%. These shares describe successful rescues. Fallback is reserved for unresolved questions after the repair budget; its call rate is not separately logged. Appendix D.1 retains the underlying event counts.



(a) Trigger groups. (b) Repair-directive distribution. (c) Resolution depth.

Figure 4: **Observed repair activity on Omni3D-Bench.** Repair activates on 9.6% of questions ($N = 500$), producing 60 attempts. (a) Attempt groups, using the original diagnostic labels. (b) Directive shares and reported per-emission recovery summaries; these are not a partition of unique rescued questions. (c) Resolution shares among rescues ($n = 22$): 59.1/27.3/4.5% at repair attempts 1/2/3 and 9.1% via terminal fallback, rounded in the plot. The fallback slice is a share of rescues, not its invocation rate.

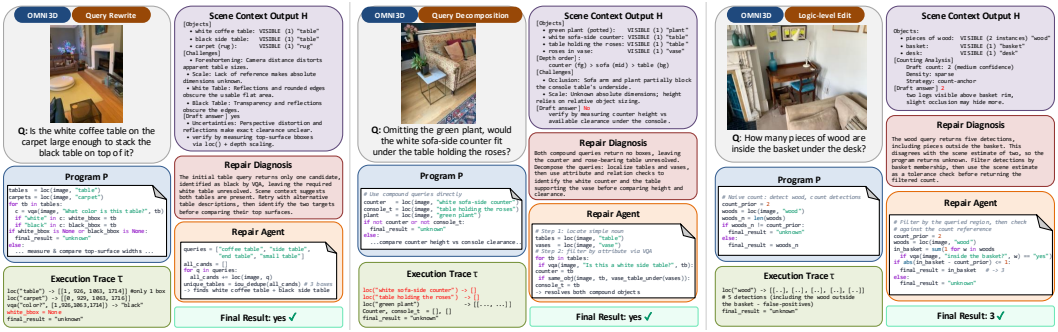


Figure 5: **Three illustrative repair strategies on Omni3D-Bench:** query rewrite, query decomposition, and logic-level edit. Each panel connects the failing program, diagnostic reasoning, and revised execution. The third panel illustrates a region-filtered count; Appendix B.3 explains its relation to the count-prior instructions.

Figure 5 illustrates how query and logic edits recover the evidence needed for an answer. A complementary pilot annotates 50 residual errors per benchmark (Appendix D.2). A concrete failure in Appendix D.4, Figure 7(c), has both the captioner and localizer report eight knobs when the ground truth is eleven. Their agreement leaves the count comparison without a discrepancy. In panel (a), two real armchairs are localized, but selecting the first box chooses the wrong referent. Together, these cases distinguish count consistency from correct object grounding: repair needs evidence that exposes the particular error, not simply more attempts.

5 Conclusion and Limitations

PitCrew connects execution evidence, supplementary scene descriptions, and targeted program repair for single-view spatial reasoning. The combined system improves the original VADAR results on Omni3D-Bench and CLEVR, with gains retained under matched-backbone controls. Component ablations support distinct roles for scene conditioning, repair, and mask-aware spatial operators. By grounding program revision in execution traces, PitCrew addresses perception failures beyond Python exceptions and makes the recovery process inspectable.

Limitations. PitCrew's repair quality depends on the evidence exposed by its perception tools and scene context. The residual cases in Appendices D.2 and D.4 highlight challenges in referent selection, per-instance attributes, shared captioner–localizer errors, and depth estimation. These cases motivate richer object-level evidence and improved spatial measurements alongside program repair. Scene captioning and conditional repair also add inference cost, quantified in Appendix E.

REFERENCES

Anthropic. Claude, 2026. URL https://claude.ai.

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

Garrick Brazil, Abhinav Kumar, Julian Straub, Nikhila Ravi, Justin Johnson, and Georgia Gkioxari. Omni3d: A large benchmark and model for 3d object detection in the wild. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13154–13164. IEEE, 2023.

Zhixi Cai, Fucai Ke, Simindokht Jahangard, Maria Garcia de la Banda, Reza Haffari, Peter J Stuckey, and Hamid Rezatofighi. Naver: A neuro-symbolic compositional automaton for visual grounding with explicit logic reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 24078–24089, 2025.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. In *The Eleventh International Conference on Learning Representations*, 2023.

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

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*, 2024b.

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

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. In *Advances in Neural Information Processing Systems*, 2024.

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.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 91–104, 2025.

Difei Gao, Lei Ji, Luowei Zhou, Kevin Qinghong Lin, Joya Chen, Zihan Fan, and Mike Zheng Shou. Assistgpt: A general multi-modal assistant that can plan, execute, inspect, and learn, 2023. *URL https://arxiv. org/abs/2306.08640*, 2023.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*, 2024.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14953–14962, 2023.

Richard Hartley, Andrew Zisserman, et al. *Multiple view geometry in computer vision*, volume 2. Cambridge university press Cambridge, 2003.

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

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9590–9601, 2024.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2901–2910, 2017.

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

Fucai Ke, Zhixi Cai, Simindokht Jahangard, Weiqing Wang, Pari Delir Haghighi, and Hamid Rezatofighi. Hydra: A hyper agent for dynamic compositional visual reasoning. In *European Conference on Computer Vision*, pp. 132–149. Springer, 2024.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 292–305, 2023.

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*, pp. 38–55. Springer, 2024.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 43447–43478, 2023.

Xiaojian Ma, Silong Yong, Zilong Zheng, Qing Li, Yitao Liang, Song-Chun Zhu, and Siyuan Huang. Sqa3d: Situated question answering in 3d scenes. In *The Eleventh International Conference on Learning Representations*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36: 46534–46594, 2023.

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

Luigi Piccinelli, Christos Sakaridis, Yung-Hsu Yang, Mattia Segu, Siyuan Li, Wim Abbeloos, and Luc Van Gool. Unidepthv2: Universal monocular metric depth estimation made simpler. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

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

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*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. Modular visual question answering via code generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 747–761, 2023.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11888–11898, 2023.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

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. In *Advances in Neural Information Processing Systems*, 2024.

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

Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13084–13094, 2024.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*, 2023a.

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 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Shukang Yin, Chaoyou Fu, Sirui Zhao, Tong Xu, Hao Wang, Dianbo Sui, Yunhang Shen, Ke Li, Xing Sun, and Enhong Chen. Woodpecker: Hallucination correction for multimodal large language models. *Science China Information Sciences*, 67(12):220105, 2024.

Haoxuan You, Rui Sun, Zhecan Wang, Long Chen, Gengyu Wang, Hammad Ayyubi, Kai-Wei Chang, and Shih-Fu Chang. Idealgpt: Iteratively decomposing vision and language reasoning via large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 11289–11303, 2023.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

APPENDIX

A EVALUATION PROTOCOL AND CONFIGURATION CONTROLS

We first specify scoring and configuration controls, then give implementation details, complete ablations, and diagnostic analyses. Runtime, variability, and prompt interfaces complete the reproducibility information.

A.1 EVALUATION DETAILS

Categorical and integer-count answers use the benchmark's answer-matching protocol. For the numerical "Other" category, we retain the mean relative accuracy (MRA) used by VADAR (Marsili et al., 2025):

$$s(\hat{y}, y) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \mathbf{1}\left[\frac{|\hat{y} - y|}{|y|} < t\right], \qquad \mathcal{T} = \{0.05, 0.10, \ldots, 0.50\}, \tag{1}$$

for nonzero numerical references. The total aggregates category contributions with the benchmark's question weighting. Consequently, Omni3D total score is not simply a fraction of binary-correct answers. Repair-rescue counts are reported separately from this aggregate score.

The InternVL2.5 training-data inventory includes CLEVR-Math and Super-CLEVR (Chen et al., 2024c). We therefore distinguish the InternVL-family CLEVR results in Table 1. This family-level training-data note does not establish test-set overlap for the evaluated InternVL3.5 checkpoint.

Section 4.1 lists the canonical models, Appendix B.3 specifies the evaluated answer routes, and Appendix G documents the prompt interfaces.

A.2 BACKBONE AND CONFIGURATION CONTROLS

The matched results are reported in main Table 3. The all-off configuration retains the baseline pipeline and retry behavior while removing scene context, the added repair mechanism, and the Spatial API together. Direct GPT-5-mini inference (49.1 in Table 1) is a separate image–question baseline, not a matched program-pipeline control. The comparisons also do not match total inference cost: PitCrew adds scene captioning and conditional repair, as quantified in Appendix E.

B METHOD AND IMPLEMENTATION DETAILS

B.1 ROLES AND CHECK SCOPE

The core loop links program execution to checking and repair: the Program Executor exposes tool returns, the Consistency Checker connects failure signals to diagnosis, and the Repair Agent uses the code and trace to propose changes. The Scene Captioner supplements this loop with context for synthesis and reference-based diagnosis; the loop can operate without it. The Spatial API improves measurements consumed by both initial and revised programs.

Scene captioning provides question-relevant entities and alternative localization terms before a detector query is fixed. The deterministic checker organizes execution feedback without a generative judging call, and a fresh repair prompt supplies the current program, trace, and available context. Table 6 summarizes the evidence supplied by each check.

Operation-specific checks are exposed by the generated program; the supplementary discrepancy routine parses H and localization records. Expectations follow the requested computation rather than a separate grammatical parser: `loc("chair")` can validly return several chairs, and "How many chairs?" may have answer zero or one. Unique-reference questions instead require candidate selection.

B.2 EXECUTION, DIAGNOSIS, AND BOUNDED REPAIR

Algorithm 1 distinguishes three stages in the checking-and-repair loop: checks within the generated program, the runtime answer gate, and discrepancy reporting for repair. An embedded count

Table 6: **What each check can establish.** Expectations and reference comparisons are evidence for repair, not ground-truth error labels.

Check source	Observation	Interpretation and boundary
Program / operation	Required operand is missing; a failure sentinel reaches the gate	Can initiate repair without H; legitimate absence and zero counts require answer-type context.
H: visibility	Visible reference with no localization, or absent reference with detections	Suspected miss or spurious detection; the captioner can be wrong.
H: cardinality	Localization count differs from the reference estimate	Disagreement, not identification of the correct count.
H: strategy	Indicated operation is not present in the trace	Auxiliary strategy discrepancy; other valid programs may use different operations.

disagreement can produce `unknown`, so an otherwise type-correct perception return may lead to a repair trigger. Conversely, an undetected error can produce a valid answer and return without repair. The H–trace routine supplements diagnosis for flagged executions. Without applicable context it returns an empty report, while the Repair Agent can still use the question, program, trace, and answer or exception. The no-context configurations in Tables 8 and 9 evaluate the pipeline with this supplementary source removed.

Algorithm 1 PitCrew execution and repair (K additional repair attempts).

Require: Image I, question q, perception operators $\mathcal{O}$, repair budget K
$H \leftarrow \textsc{SceneCaptioner}(I, q)$ (or $\emptyset$ without scene context)
$P \leftarrow \textsc{ProgramSynthesizer}(q, H)$
Initialize trace-aware cache $\mathcal{C} \leftarrow \emptyset$
for $k = 0$ **to** K **do**
  $(\hat{a}, \tau, e) \leftarrow \textsc{ProgramExecutor}(P, I, \mathcal{O}, \mathcal{C})$ {Program checks may return a failure sentinel; e is an execution exception.}
  if $e = \emptyset$ and $G(\hat{a}, q) =$ pass **then**
    return $\hat{a}$
  end if
  if $k < K$ **then**
    $\Delta \leftarrow \textsc{ContextDiscrepancies}(\tau, H)$ {Deterministic; empty without applicable context.}
    $(P', d) \leftarrow \textsc{RepairAgent}(q, P, \tau, \hat{a}, e, \Delta, H)$
    $\mathcal{C} \leftarrow \textsc{UpdateCache}(\mathcal{C}, d)$
    $P \leftarrow P'$
  end if
end for
return $\textsc{TerminalFallback}(I, q, H)$

The cache preserves compatible tool results across attempts. A synonym query or decomposition introduces new arguments and may invoke new perception calls; the fact that a program is regenerated does not require every unchanged call to run again. Each repair returns a complete program whose edits are guided by the observed failure.

B.3 Scene Context, Count Estimates, and Draft Answers

H is structured text rather than a ground-truth scene graph. Its fields include `[Visual Scene Analysis]`, optional `[Counting Analysis]`, `Challenges`, `Depth order`, and `[Draft Answer]` (Figure 8). We distinguish the *count estimate*, labeled "Draft count" in the counting block, from the *draft answer*, a candidate response to the full question. They may coincide for a direct counting question but have different roles: the former supplies a cardinality reference and count prior; the latter provides a cached candidate for terminal fallback. The shared text, including the draft-answer block, is visible to synthesis and repair. Thus, this distinction describes intended

uses, not an information barrier. The discrepancy routine parses applicable visibility, count, and strategy fields.

Scene-context instructions for program synthesis. The *scene-context bridge* in Figure 10 is a set of instructions appended to the Program Synthesizer's prompt, not an additional agent or model call. It connects the captioner's output to object queries, attribute checks, and spatial computations. Rule A specifies the direct integer-count prior route; Rule B uses the draft block's evidence and strategy to guide program construction. These instructions also ask the generated program to expose unresolved required operands as failure values, providing evidence for the subsequent checking-and-repair loop.

Table 7: **Information and answer routes in the evaluated prompts.** A draft candidate's availability at termination is distinct from the count prior and strategy fields consumed earlier.

Information / path	Consumer	Role and answer source
Descriptions, depth, challenges, strategy	Program Synthesizer; Repair Agent; applicable checker rules	Supplementary visual context, query terms, and guidance for operations.
Draft answer	Terminal fallback; also visible in synthesis and repair prompts	Cached candidate response; programs are instructed to compute rather than copy it.
Count estimate ("Draft count")	Consistency Checker; count-anchor program path; terminal fallback	Cardinality reference; can also supply the reported count under the existing heuristic.
Other program answers	Program Executor	Computed from the invoked tools and program logic, subject to available checks.
Non-count terminal fallback	Configured image–question VLM	Best-effort answer after the repair budget; model-dependent and not program-verified.

Integer counts versus numerical ratios. Rule A in Figure 10 applies only to direct integer counting with a counting-analysis block. In sparse scenes, its prompt compares localization count to the prior and returns the prior when the absolute difference is at most one; larger disagreement returns `unknown`. Ratio and stacking questions are excluded and must compute a numerical ratio from measurements. Yes/no comparison-counting uses a separate route that compares `count_objects()` results. These count-prior heuristics are part of the evaluated program-generation policy.

The report routine in Figure 11 supplies diagnosis for flagged executions; its asymmetric count-prior guidance is distinct from Rule A's sparse-scene acceptance test. The repair instructions also prefer the count prior after targeted localization retries fail. Thus, a reported answer may depend directly on the count estimate. Removing `[Draft Answer]` does not remove this separate counting block.

Fallible context and terminal fallback. The draft answer is precomputed during scene captioning and retained to support bounded termination. The same block also supplies evidence and strategy for earlier synthesis and repair; its candidate can serve as a sanity check in the original prompt. Integer-count fallback can reuse the separate count estimate, avoiding a new inference for that path. Other questions use the configured holistic VLM; if that generator is unavailable or its call fails or returns no answer, the cached draft answer is used when available. Cached reuse avoids a new model call; a fresh holistic answer adds inference cost. The logs identify two successful fallback outcomes but do not separately enumerate cached-draft uses, fresh fallback calls, or conditional fallback accuracy.

A qualitative Scene Captioner example. The first panel of Figure 5 asks whether a white coffee table can accommodate a black table on top. Its existing captioner output illustrates the interface:

> **Objects:** white coffee table: visible (1), "table"; black side table: visible (1), "table"; carpet: visible (1), "rug".
> **Challenges:** perspective affects apparent size; reflections and rounded or transparent edges obscure the usable surfaces.
> **Suggested operation:** measure the top surfaces using localization and depth scaling.

This excerpt condenses the output shown in the figure. "Table" and "rug" are suggested text queries; localization tools still produce the bounding boxes. The two table descriptions help diagnose unresolved selection when a broad query returns only one candidate. The displayed diagnostic prose summarizes the repair reasoning, which can combine VQA and code evidence beyond the deterministic localization report. The accompanying draft answer is "yes", but the successful program measures and compares the surfaces. In the third panel, "Draft count: 2" differs from the repaired region-filtered result of three. That program uses the estimate as a tolerance check and returns the filtered count; the default count-anchor instructions in Figure 10 instead return the prior. The panel illustrates a program-specific logic edit, rather than the default answer route.

B.4 Spatial Operator Geometry

Figure 3 in the main paper illustrates both supporting operators. Using the pinhole camera model (Hartley et al., 2003), the calibrated backprojection for mask pixels is

$$x_{3D}(u,v) = \frac{(u-c_x)d(u,v)}{f_x}, \qquad y_{3D}(u,v) = \frac{(v-c_y)d(u,v)}{f_y}. \tag{2}$$

Here $d(u,v)$ is the per-pixel metric depth estimate from UniDepthV2, (u,v) and (c_x,c_y) are pixel coordinates, and f_x, f_y are focal lengths in pixels. Width and height estimates use quartile-trimmed spans of these coordinates over the object mask. They describe the visible estimated surface along camera axes, not an object-aligned, rotation-invariant reconstruction.

Mask depth is the median of in-mask samples after discarding the upper and lower quartiles. This reduces reliance on a center pixel that may lie on background or an occluder, but does not remove segmentation or depth bias. Standard backprojection and aggregation are supporting operators rather than new geometric principles.

C Scene-Context Utility and Component Ablations

C.1 Field Ablations

Main Table 4 reports the complete field ablations over 500 Omni3D questions. Each intervention modifies the context text shared by synthesis, repair, and discrepancy reporting while retaining the other components. The mismatched-image control substitutes another image's context; no-context removes the scene channel. The draft-answer intervention removes its evidence and strategy together with the candidate answer, while leaving the separate counting-analysis block present. The results characterize the utility of each information block across the pipeline.

Table 5b provides a separate capacity control: four Scene Captioners yield 49.3–49.8. Among candidates with direct-answer results, Qwen3-VL-8B, Qwen3-VL-32B, and GPT-5-mini span 37.9–49.1 in direct Omni3D performance but 49.4–49.8 as PitCrew captioners. Their role in the pipeline therefore differs from direct answering. The mismatched-context control separately tests the effect of irrelevant descriptions.

C.2 Complete Factorials

The main paper presents full PitCrew and one-component removals. Tables 8 and 9 retain all eight configurations. The CLEVR cohort is fixed across rows: 300 questions sampled with seed 42, with 100 each for Yes/No, MC, and Count. This subset gives 91.3 for full PitCrew and 69.3 for all-off, whereas the main CLEVR comparison reports 89.4 and 53.6 under its own cohort and configurations. The same synthesis backbone and component definitions are used across the subset study; the localizer is Molmo-7B.

For a descriptive interaction measure, let $A(r,s,h)$ denote the score with Repair r, Spatial s, and Scene h enabled or disabled. The Scene–Repair difference-in-differences, averaged over Spatial, is

$$I_{RH} = \frac{1}{2}\sum_{s\in\{0,1\}} \big[A(1,s,1) - A(0,s,1) - A(1,s,0) + A(0,s,0)\big]. \tag{3}$$

Table 8: **Complete Omni3D component factorial (%, 500 questions).** Repair, Spatial API, and scene context are toggled with the GPT-5-mini backbone and perception models fixed. The all-off condition is the additional VADAR configuration, not the original Table 1 baseline.

Repair	Spatial	Scene	Y/N	MC	Count	Other	Total
✓	✓	✓	68.0	66.3	35.7	43.2	49.8
	✓	✓	61.3	59.3	21.4	41.4	44.7
✓		✓	69.3	63.9	27.1	40.7	47.1
✓	✓		66.6	57.4	22.8	37.5	43.2
✓			65.3	56.9	25.7	36.1	42.6
	✓		65.3	57.4	22.8	36.7	42.6
		✓	60.0	61.6	22.0	38.3	43.3
			58.6	56.9	22.8	35.7	41.0

Table 9: **Complete CLEVR component factorial (%, fixed 300-question subset).** Each answer type has 100 questions. All rows use GPT-5-mini for synthesis, with Qwen3-VL-8B when scene context is enabled. Scores are percentages on the fixed subset.

Repair	Spatial	Scene	Y/N	MC	Count	Total
✓	✓	✓	94.0	94.0	86.0	91.3
	✓	✓	74.0	88.0	78.0	80.0
✓		✓	88.0	90.0	84.0	87.3
✓	✓		80.0	84.0	72.0	78.7
✓			78.0	64.0	74.0	72.0
	✓		80.0	60.0	74.0	71.3
		✓	82.0	80.0	80.0	80.7
			78.0	56.0	74.0	69.3

Using the reported table values gives +3.35 pp on Omni3D and +3.90 pp on CLEVR. These descriptive interactions support complementary scene and repair roles. Individual components also improve over all-off. The Spatial API contributes 2.7 pp on Omni3D and 4.0 pp on the CLEVR subset when added to scene context and repair.

D Diagnostic Coverage and Failure Analysis

D.1 Repair Attempts and Resolution Depth

In the analyzed 500-question Omni3D run, 48 questions enter repair and 60 repair calls are issued. The 22 rescues are unique questions: 20 are resolved through revised programs (13/6/1 at attempts 1/2/3) and 2 through fallback. Attempt-level trigger groups and directives in Figure 4 can include several emissions for one question. Their displayed recovery percentages must not be multiplied and summed as if they partitioned the 22 unique rescues.

The budget sweep in Table 5(d) evaluates separate configurations: $K = 0, 1, 2, 3, 4$ yield 44.7, 48.7, 49.7, 49.8, and 49.6, respectively. Most of the improvement comes from the first two repair attempts. These configuration scores are separate from the resolution-depth counts above.

D.2 Residual-Error Pilot

A closed-label pilot assigned 50 residual wrong answers from each benchmark to one of six error causes. These cause labels differ from the four runtime discrepancy signals. For the Omni3D pilot, 47/50 residual cases emitted none of those signals and 3/50 remained wrong despite a signal. The 94% without a discrepancy is conditional on residual errors; successful repairs are excluded from this pilot.

Interface, context, and program-logic labels account for 24/50 cases, while perception-capability and confident-but-wrong labels account for 26/50. These annotations identify where better queries,

Table 10: **Omni3D residual-error pilot (50 cases).** Cause labels summarize the sampled residual errors; possible responses identify directions for improving evidence or repair.

Cause label	Cases	Share	Possible response
Vocabulary mismatch	9	18%	Alternative localization vocabulary
Compound query	6	12%	Decomposed localization and attribute checks
Scene-caption error	5	10%	Better or additional scene evidence
Program-logic error	4	8%	Better revision or operation checks
Perception-capability limit	14	28%	Improved perception or geometry
Confident-but-wrong	12	24%	Additional evidence exposing the disagreement

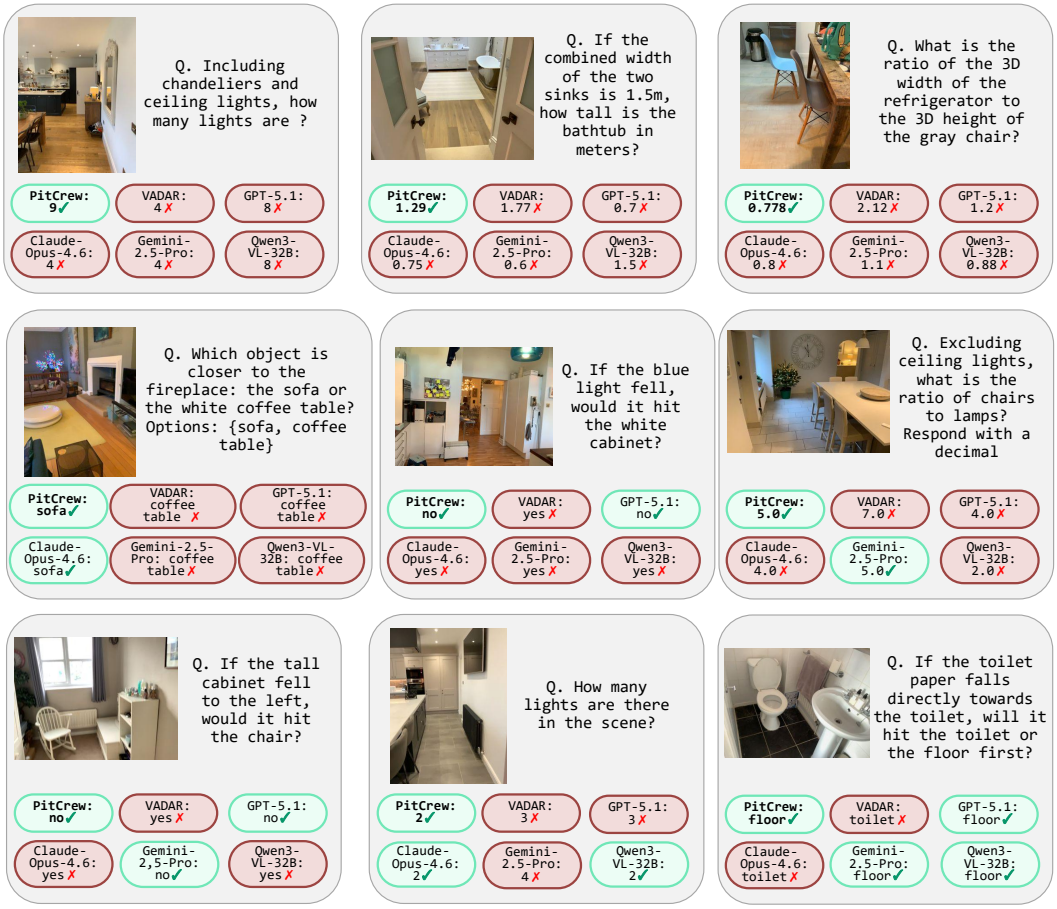


Figure 6: **Selected qualitative comparisons on Omni3D-Bench.** PitCrew, VADAR, and the displayed direct VLMs answer the same questions. Correctness symbols apply to the shown outputs under the evaluation protocol. Aggregate comparisons, including the Qwen3.5 and Gemma-4 baselines, appear in Table 1.

evidence, or operators may help; they are not measured recovery rates. We report the observed Omni3D sample directly rather than extrapolating its mixture to the full benchmark.

D.3 Qualitative Comparison

Figure 6 complements Table 1 with selected counts, relations, and measurements. PitCrew's explicit computation provides an inspectable alternative to direct answering; individual examples show both shared successes and differences between methods.

D.4 Failure Cases and Fallible Scene Context

Figure 7 organizes six failure examples by the evidence needed to resolve them: referent identity, per-instance attributes, counts, and geometry.

Referent ambiguity and concordant errors (a, c). In (a), `loc("armchair")` returns two real candidates and selecting `boxes[0]` commits to the wrong instance. Matching the scene's object count does not resolve which object the question refers to. In (c), the captioner and localizer both report eight knobs although the ground truth is eleven. The count comparison cannot expose an error when its two sources agree. Detecting this shared omission requires evidence beyond the two agreeing counts.

Missing per-instance evidence (b, d). In (b), a spurious chair box survives a geometric relation filter. In (d), a VQA attribute decision incorrectly removes one of six localized chairs. The existing visibility and count fields need not constrain those individual downstream decisions. Resolving these errors requires per-instance relation or attribute evidence.

Depth and geometry (e, f). In (e), a depth-magnitude error propagates into a ratio of 2.15 instead of 2.73. In (f), the displayed image-plane approximation estimates a TV-stand height of 1.79 rather than 0.62. PitCrew's backprojection addresses that approximation, but still depends on depth, masks, and visible surfaces. Query repair alone cannot supply missing metric or occluded-surface evidence.

Incorrect context and occlusion. When H is wrong, it can misdirect both the initial program and the repair. The count-anchor and trust directives can prefer a mistaken caption over a correct detection. Occlusion, small scale, truncation, and ambiguous categories can affect both sources. This motivates context that localizes uncertainty to individual objects and checks tied to the specific operation, rather than relying only on global agreement.

E Runtime and Resource Use

The reported stage measurements use an NVIDIA V100 for local inference and the GPT-5-mini API for synthesis and repair. Signature and API construction are run once per benchmark in this evaluation and amortized over questions. The Scene Captioner runs per question, even when no repair or fallback occurs. Repair is conditional: 60 calls across 500 questions give 0.12 calls per question. Re-execution of perception tools contributes to the larger execution-time variance.

Table 11: **Stage runtime on Omni3D-Bench (seconds).** Local inference uses an NVIDIA V100; synthesis and repair use the GPT-5-mini API. Signature and API construction are amortized per benchmark. Scene captioning and synthesis are per question; repair is per call (0.12 calls/question in the analyzed run). Execution includes re-executions. Entries retain the reported mean and standard deviation. Terminal fallback has no separately reported timing row.

	Signature (per bench)	API (per bench)	Captioner (per Q)	Synthesis (per Q)	Repair (per call)	Execution (per Q)
PitCrew (full)	$12_{\pm 4}$	$65_{\pm 20}$	$8_{\pm 2}$	$3_{\pm 1}$	$3_{\pm 1}$	$8_{\pm 18}$
PitCrew (K=0)	$12_{\pm 4}$	$65_{\pm 20}$	$8_{\pm 2}$	$3_{\pm 1}$	N/A	$4_{\pm 5}$
VADAR (baseline)	$12_{\pm 4}$	$65_{\pm 20}$	N/A	$3_{\pm 1}$	N/A	$2_{\pm 2}$

Adding per-question means gives $8 + 3 + 8 + 0.12 \times 3 \approx 19.4$ seconds for PitCrew, versus $3 + 2 = 5$ seconds for the VADAR pipeline, excluding the small amortized setup contribution and any separately unreported terminal-fallback overhead. This stage-sum estimate includes local captioning and conditional repair; it is not an independently measured end-to-end latency. The Consistency Checker uses deterministic code and adds no judging-model call.

One synthesis call plus an average of 0.12 repair calls gives 1.12 calls per question for those stages in the analyzed run. This excludes amortized Signature/API construction, any model calls inside the perception stack, and the terminal fallback route.



Figure 7: **Six illustrative failure cases on Omni3D-Bench.** (a) Unresolved referent selection; (b) a spurious detection surviving a relation filter; (c) a concordant captioner–localizer miscount; (d) an attribute error; (e) depth-magnitude error; (f) an inaccurate geometric extent. Panel (f) shows the image-plane size approximation, not the backprojection operator used by full PitCrew. "Analyst" denotes the Scene Captioner.

F Experimental Variability and Reproducibility

Reproducibility Statement. Section 4.1 specifies the model stack and repair budget. Algorithm 1 and Appendix B.3 detail execution, caching, and answer routing. Appendices A.1, C.2, and E document scoring, component cohorts, and runtime; Appendix G summarizes the prompt interfaces.

Score variability. Table 12 summarizes score variability for Table 1. Repeated-run results report medians and standard deviations across three runs. Zero entries denote the reported dispersion at the displayed precision.

G Prompt Templates and Checker Pseudocode

Figures 8–10 summarize the prompt interfaces using edited, condensed templates. They clarify the inputs, outputs, and illustrative repair edits; reported results use the evaluation configuration in Section 4.1 and the answer routes in Appendix B.3.

The base Signature/API and program-synthesis prompts follow VADAR. Figure 8 shows the Scene Captioner's descriptions, estimates, and candidate-answer fields. Figure 9 organizes evidence-based program revision; its edit families are distinct from the runtime directive names. Figure 10 shows the scene-context instructions supplied to the Program Synthesizer, including the direct-count and strategy routes. This bridge is part of synthesis, not a separate component of the repair loop.

Figure 11 gives explanatory pseudocode for the deterministic Consistency Checker. Its answer gate can initiate repair without H; the context-dependent routine supplies additional diagnosis for flagged executions. The legacy `TRUST_ANALYST` and `TRUST_PERCEPTION` fields express heuristic re-

Table 12: **Reported score variability for Table 1.** Dispersion of total scores in percentage points; repeated-run entries summarize three runs.

	Method	Omni3D-Bench	CLEVR
Proprietary VLMs	GPT-4o	0.6	1.4
	GPT-5-mini	0.5	1.0
	GPT-5.1	0.4	2.1
	Claude Sonnet 4.6	0.2	0.3
	Claude Opus 4.6	0.3	0.1
	Gemini 2.5 Flash	0.3	0.2
	Gemini 2.5 Pro	0.6	0.5
Open-weight VLMs	InternVL-3.5-8B	0.0	0.0
	Qwen3-VL-8B	0.0	0.0
	Qwen3-VL-32B	0.0	0.0
	Qwen3.5-35B-A3B	0.0	0.0
	Qwen3.5-122B-A10B	0.0	0.0
	Gemma-4-26B-A4B-it	0.0	0.0
Program-based methods	VisProg	0.3	0.9
	ViperGPT	0.3	1.1
	VADAR	1.8	2.9
	PitCrew (ours)	0.4	0.8

pair guidance, not validated source-reliability judgments. The complete budget and trigger order appear in Algorithm 1.

H Broader Impacts

Execution traces and explicit repair reports can aid debugging and make some visual reasoning failures easier to inspect. Their presence can also create unjustified confidence if users equate a no-trigger result with verification. Deployment should communicate the remaining error modes and the distinction between program and fallback answers. The method adds local inference, API use, and re-execution costs; its empirical accuracy improvements do not establish suitability for high-stakes decisions or lower overall resource consumption.

I AI Use Statement

Generative AI tools assisted manuscript and reference checking.

```
You are the Scene Captioner for a programmatic spatial reasoning system.
Describe the image with respect to the question. Supply concise visual
context for choosing objects, text queries, and spatial operations.
Report uncertainty where the image is ambiguous.

For each question-relevant object, provide:
- Visibility and estimated number of visible instances.
- Brief instance descriptions: appearance and approximate image location.
- A short text query for localization, usually a simple object noun.
  For example, suggest "chair" and check "white" separately with vqa().
  The legacy field "DINO hint" contains text, not bounding-box coordinates.
- Disambiguation cues when several candidates may match the question.
- Relative depth order when relevant to a size or ratio comparison.
- Perception challenges: occlusion, small size, or similar instances.

Distinguish two kinds of "how many" questions:
A. Direct integer count: "How many chairs are visible?"
   Include [Counting Analysis]: an instance list, count estimate, density,
   and suggested counting strategy. Zero is a valid count.
B. Ratio / stacking: "How many X would reach the height of Y?"
   Omit [Counting Analysis]. Treat the answer as a numerical ratio of the
   requested dimensions, not as a discrete object count.

End with [Draft Answer]: a candidate response with evidence and a strategy.
The candidate is retained for terminal fallback; its evidence and strategy
also guide synthesis and repair. Programs should compute their answer
rather than copy this candidate. The separate integer-count estimate
follows the count-anchor instructions in the bridge.

Choose the draft strategy according to the question:
- trust-direct: factual presence; confirm with perception.
- verify-direct: attributes, spatial comparisons, or measurements.
- decompose-uncertain: hypothetical or multi-step spatial computations.

Output format:
[Visual Scene Analysis]
Objects:
- <object>: VISIBLE (<N> instance(s)) | NOT VISIBLE
  - <brief instance description>
  DINO hint: "<short text query>"
  [Disambiguation: <candidate-selection cue, if needed>]
  [Counting: <len(loc()) | count-anchor> - <reason, if relevant>]
Challenges:
- <perception challenge>
Depth order: <foreground to background; omit if irrelevant>

[Counting Analysis]  # Direct integer-count questions only
Instances:
- <instance, image location, visibility or occlusion>
Draft count: <N> (confidence: <high|medium|low>)
Density: <sparse|dense>
Recommended strategy: <count-anchor|loc-direct>

[Draft Answer]
- Best answer: <candidate> (confidence: <high|medium|low>)
- Key evidence: <specific observations>
- Uncertain because: <ambiguity or N/A>
- Question type: <factual-presence|factual-attribute|
                  spatial-comparison|measurement|hypothetical>
- Recommended strategy: <trust-direct|verify-direct|decompose-uncertain>

Keep the entire output under 300 words.
```

Figure 8: **Scene Captioner: structured scene-context template.** Question-conditioned descriptions provide object-query terms, instance cues, and anticipated perception challenges. The integer-count estimate and full-question draft candidate occupy separate fields; their uses are detailed in Appendix B.3.

```
You are the Repair Agent for a programmatic spatial reasoning system.
An execution has been flagged by an exception or an inadmissible answer.
Diagnose the issue using the failed program and trace, then return a
complete revised program. Tie each edit to the observed execution evidence.

Question: {question}
Previous synthesis notes: {scratchpad}
Executed program (attempt {attempt}):
{program}
Execution trace, including tool arguments, returns, and cache status:
{exec_log}
Result: {final_result!r}
Expected answer type: {answer_type}
Observed failure information:
{violation_context}

When available, the runtime appends:
- Scene context H: descriptions, estimates, evidence, and strategy hints.
- Diagnostic report: applicable H–trace discrepancies and repair guidance.
Without H, use the question, program, trace, and observed failure.
With H, use descriptions to suggest what to inspect; consider conflicting
descriptions and tool returns in light of the execution evidence.

Choose an edit appropriate to the suspected failure:
- Query rewrite: narrow an ambiguous query or retry alternative vocabulary.
  Examples: loc("table") -> loc("coffee table");
            loc("sofa") -> loc("couch").
- Query decomposition: separate category localization from an attribute.
  Example: loc("white chair") -> loc("chair"), followed by
           vqa("Is it white?") on each candidate.
- Logic-level edit: revise selection, filtering, control flow, or aggregation.
  Example: count all pieces -> count only pieces inside the basket.

For empty required localizations, try a simpler/synonymous query before
decomposition. Runtime directives, when supported by the configuration:
  synonym_loc: <alternative text query>
  decompose_loc: <simple noun>
  rephrase_vqa: <more specific question>
  escalate_loc: <configured localization-cascade step>
Follow the configured escalation policy after unsuccessful query repairs.
Reuse compatible cached calls; changed queries may require new calls.

Program rules:
- Return a complete program and assign final_result at module scope.
  Do not define helper functions in the repaired program.
- Initialize the configured failure value before computation:
  "unknown" for categorical or unresolved required-object questions;
  0.0 for the configured ratio/size failure path.
  Legitimate zero counts and negative existence answers are admissible.
- If H says an object is absent but loc() was never called, inspect it.
  Retry a suitable alternative query before treating a required object
  as unresolved. Missing required operands should expose a failure value.
- For direct integer-count disagreement, use instance descriptions and
  query retries to recover candidates. The retained count-anchor policy
  prefers count_prior if targeted retries still cannot reach that estimate.
- For ratio questions, compute the requested dimension ratio from tools.
  Do not copy the full-question draft candidate as the program answer.
- Collect candidates passing vqa() before attribute-based selection;
  the retained template selects the largest box when several pass.
- Leave unresolved execution to bounded repair and terminal fallback.
  Do not insert a holistic fallback call into the numeric program.

Return:
<diagnosis>Evidence and the suspected failing operation.</diagnosis>
<repair_directive>Applicable runtime directives.</repair_directive>
<program>The complete revised program.</program>
```

Figure 9: **Repair Agent: evidence-based program revision template.** The question, failed program, execution trace, and result guide a diagnosis, applicable directives, and a complete revised program. Scene context and its diagnostic report supplement these inputs when available. Query and logic edits are alternatives selected for the suspected failure, rather than a fixed sequence.

```
SCENE-CONTEXT BRIDGE FOR THE PROGRAM SYNTHESIZER

Use descriptions to select question-relevant object queries and operations.
The question determines the computation; H supplies visual hints.
If H reports absence, still inspect with loc() and, when needed, vqa().
Use short localization queries and candidate-level attribute checks.
Expose unresolved required operands as failure values so the runtime can
attempt repair. Ordinary absence or zero count is not automatically failure.

RULE A — INTEGER-COUNT ESTIMATE
Apply only to direct integer-count questions with [Counting Analysis].
Read "Draft count" as count_prior. This is the separate count estimate,
not the "Best answer" field of the full-question [Draft Answer] block.
In this path, use the prior instead of calling count_objects().

For density=sparse under count-anchor:
    count_loc = len(loc(image, "X"))
    if abs(count_loc - count_prior) <= 1:
        final_result = count_prior
    else:
        final_result = "unknown"  # disagreement exposes a repair trigger

For density=dense under count-anchor:
    final_result = count_prior

For strategy=loc-direct:
    final_result = len(loc(image, "X"))

These are the retained counting heuristics. Do not apply them to ratio or
stacking questions: compute size_Y_dim / size_X_dim as a numerical ratio.
Yes/no comparison-counting (e.g., "Are there more X than Y?") has a separate
route: compare count_objects(image, "X") with count_objects(image, "Y").

RULE B — EVIDENCE AND STRATEGY FOR PROGRAM CONSTRUCTION
Use [Draft Answer]'s evidence and recommended strategy to plan the program.
Its candidate can serve as a sanity check; the computed answer takes
precedence over the candidate.

- trust-direct: inspect with loc() and verify any required attribute with
  vqa(); infer presence/absence from the observed evidence.
- verify-direct: compute attributes, comparisons, and measurements using
  the relevant localization, VQA, depth, and size operators.
- decompose-uncertain: compute the constituent spatial subproblems, then
  combine their results according to the question.

Do not hardcode the full-question draft candidate as a numeric answer or
substitute its boolean for an uncomputed hypothetical relationship.
Rule A explicitly governs the separate integer-count estimate.
Do not append a draft-based fallback inside a generated numeric program.
Terminal fallback is controlled by the runtime after the repair budget.
```

Figure 10: **Program Synthesizer: scene-context instructions.** This prompt extension specifies how supplementary H informs program construction. Rule A describes the integer-count prior route, including sparse-scene tolerance, dense-scene prior use, and the localization-based alternative. Rule B uses the draft block's evidence and strategy to plan tool computations. These instructions belong to synthesis; terminal fallback is controlled by the runtime.

```
# (a) Execution gate: independent of scene context H.
# Program checks expose unresolved operands as failure values.
# Admissibility uses the question and configured answer rules;
# legitimate zero counts and negative existence answers remain valid.
def needs_repair(answer, question, exception):
    return (exception is not None
            or not answer_is_admissible(answer, question))

# (b) Supplementary report for a flagged execution; no judging-model call.
# Labels denote observed disagreements, not ground-truth error classes.
def context_discrepancies(tau, H):
    objects = parse_scene_objects(H) if H else []
    if not objects:
        return []
    loc_results = parse_loc_results(tau)  # query -> list[str] of returns
    signals = []
    EMPTY = {"[]", "", "None", "''", '""'}

    for obj in objects:
        name, hint = obj.name, obj.dino_hint
        keys = dict.fromkeys([name, hint])  # avoid duplicate trace entries
        matching = [r for k in keys for r in loc_results.get(k, [])]
        if not matching:
            continue  # localization was not called for this object
        found = any(r.strip() not in EMPTY for r in matching)

        if obj.visible and not found:
            signals.append(Signal(
                type="VISIBILITY_MISS", object=name,
                trust="TRUST_ANALYST",
                action="retry vocabulary, then decompose the query"))

        elif not obj.visible and found:
            signals.append(Signal(
                type="VISIBILITY_FP", object=name,
                trust="TRUST_PERCEPTION",
                action="verify the returned region with vqa()"))

        elif obj.visible and found:
            loc_n = first_positive_count(matching)
            if loc_n is not None and loc_n != obj.count:
                action = ("prefer count_prior"
                          if 0 < loc_n <= obj.count + 1
                          else "inspect duplicate / irrelevant detections")
                signals.append(Signal(
                    type="COUNT_MISMATCH", object=name,
                    trust="UNCERTAIN", action=action))

        # Legacy per-object field, distinct from the bridge's
        # count-anchor / loc-direct recommendation.
        if ("count_objects" in obj.counting_strategy
                and not used_count_objects(tau, name, hint)):
            signals.append(Signal(
                type="STRATEGY_IGNORED", object=name,
                trust="TRUST_ANALYST",
                action="confirm with loc(); apply the count-prior route"))
    return signals

# Within the remaining repair budget:
# if needs_repair(answer, q, exception):
#     report = context_discrepancies(tau, H)  # [] without applicable H
#     P_new = RepairAgent(q, P, tau, answer, exception, report, H)
#     execute_and_check(P_new)
# Otherwise return the program answer. On budget exhaustion, use fallback.
```

Figure 11: **Consistency Checker: execution gate and supplementary diagnostics.** (a) Exceptions and answer-admissibility failures can initiate repair without H. (b) When context is available, localization records are compared with parsed visibility, count, and strategy fields to guide diagnosis. The report labels denote disagreements, and its trust and action fields are heuristics. Both stages are deterministic and use no judging-model call.