SEE ONCE, QUERY MANY: 3D SCENE AFFORDANCE SEGMENTATION VIA AGENTIC MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Scene-level affordance segmentation is fundamental for embodied agents to perceive interaction possibilities in complex 3D environments. Existing MLLM-based methods face two key limitations: repetitive visual processing causes substantial redundancy, while insufficient modeling of remote functional dependencies and complex spatial relations limits their ability to ground intricate instructions. We present **MemAff**, a training-free agentic framework that converts a single scan into an evolving *Agentic Scene Memory*. Through incremental editing and on-demand visual recall, a vision-language agent anchors objects to frames and cross-links spatially separated functional pairs. For each instruction, MemAff traverses this memory to select the view best exposing the actionable region, followed by temporal propagation and cross-view 3D lifting. This agentic memory construction and retrieval enables "See Once, Query Many" without repeated visual processing. Experiments on SceneFun3D and our 3,149-instruction FunEnhance benchmark show state-of-the-art performance, reaching 30.5% AP_{50} and 24.9% mIoU while reducing per-instruction latency by over 20×.

1 INTRODUCTION

Affordances characterize the actions enabled by objects and their functional parts (Gibson, 2014). Understanding and localizing affordances in 3D scenes is fundamental for enabling embodied agents to interact effectively and purposefully with the physical world (Mo et al., 2021; Ning et al., 2023; Do et al., 2018). In realistic environments, an agent must consolidate evidence from long visual observations into reusable scene knowledge for grounding multiple natural-language instructions. This is challenging because relevant evidence may be distributed across distant views, functionally related objects may not co-occur, and the target affordance may be a small, context-dependent object part (Delitzas et al., 2024; Rotondi et al., 2025; He et al., 2026). For instance, grounding "Turn on the TV using the remote on the nightstand" requires identifying the correct remote through its spatial relation to the nightstand and its functional association with a TV observed elsewhere. Scene-level affordance grounding therefore requires: *(1) retaining knowledge across long observations and subsequent queries*, *(2) reasoning over complex spatial relations and remote functional dependencies*, and *(3) precisely localizing fine-grained actionable regions in 3D*.

Despite recent progress, existing 3D affordance understanding approaches fall short in three respects. First, most adopt a per-query perception paradigm (Corsetti et al., 2025; Wang et al., 2025), re-encoding the entire scan for every instruction. This incurs substantial redundancy and treats observations as disposable context rather than knowledge that can be accumulated, refined, and reused. Second, existing methods often rely on local proximity or visual co-occurrence to infer affordances (Rotondi et al., 2025; He et al., 2026; Zhang et al., 2025). MLLM-based pipelines further feed dense visual inputs directly into the reasoning model without a structured cross-view representation (Corsetti et al., 2025; Wang et al., 2025; Lin & Xu, 2026), making it difficult to connect spatially separated observations and recover remote functional relationships. Third, single-frame grounding may select context-rich views in which the actionable part is small, oblique, or occluded, while direct 2D-to-3D lifting can amplify mask leakage into off-affordance points. Consequently, existing pipelines do not jointly support persistent scene understanding, global relational reasoning, and precise 3D affordance localization.

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.

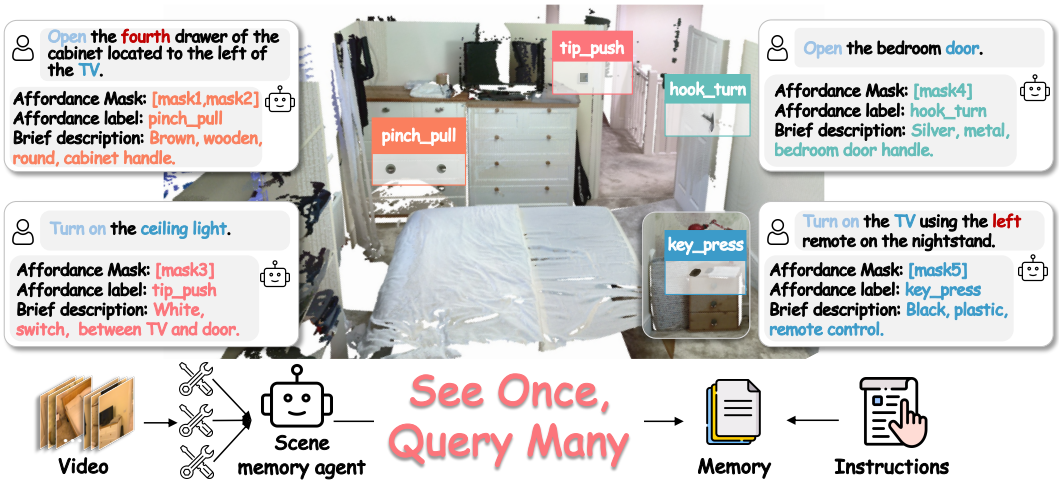

Figure 1: **The "See Once, Query Many" paradigm for 3D scene-level affordance segmentation.** MemAff compresses long visual scans into a persistent, structured Scene Memory, serialized in Markdown for readable and VLM-friendly querying. This memory enables efficient, training-free grounding of complex instructions and remote functional linkages without re-processing visual inputs for each task.

To address these limitations, we propose **MemAff**, a training-free framework that uses *agentic tools to organize reusable scene memory* for "**See Once, Query Many**" affordance segmentation. During a single scan pass, a vision-language agent uses atomic insertion and replacement tools to accumulate and refine a structured, language-readable memory. An on-demand visual recall tool resolves ambiguity when earlier frames have left the bounded visual context. Together, these tools turn observations into persistent object records, frame anchors, spatial descriptions, and cross-links between functionally related entities. Each subsequent instruction is resolved through memory retrieval and relation traversal; only the retrieved candidate views and a short temporal neighborhood undergo task-specific visual processing. An acting-part visibility check, point localization, and cross-view mask fusion then recover the 3D affordance. This separation amortizes scene understanding across instructions while retaining the visual evidence needed for fine-grained segmentation. Our contributions are as follows:

- We formulate scene-level affordance segmentation as reusable scene-memory construction followed by instruction-conditioned retrieval, enabling a training-free "See Once, Query Many" pipeline.
- We design agentic editing and visual recall tools that organize a persistent scene abstraction with object-centric frame anchors and explicit functional cross-links, supporting grounding across spatially separated observations.
- We combine memory-guided view selection with temporal segmentation and cross-view lifting. On SceneFun3D, MemAff achieves 30.5% AP_{50} and 24.9% mIoU, with approximately 51 s online latency per instruction after a one-time memory construction stage.

2 Related Work

Affordance Understanding. Affordance learning connects visual observations to actionable object regions (Yang et al., 2024; Yu et al., 2025; Wu et al., 2025; Deng et al., 2021; Nagarajan & Grauman, 2020; Wu et al., 2023; Cuttano et al., 2024). SceneFun3D (Delitzas et al., 2024) extends this problem to language-conditioned localization of functional elements in indoor scenes. Recent methods include training-based grounding systems such as AffordBot (Wang et al., 2025) and TASA (He et al., 2026), and training-free visual reasoning pipelines such as Fun3DU (Corsetti et al., 2025). MemAff focuses on the repeated-query setting: it retains scene knowledge so that new instructions can retrieve relevant evidence without repeating scene-wide visual search.

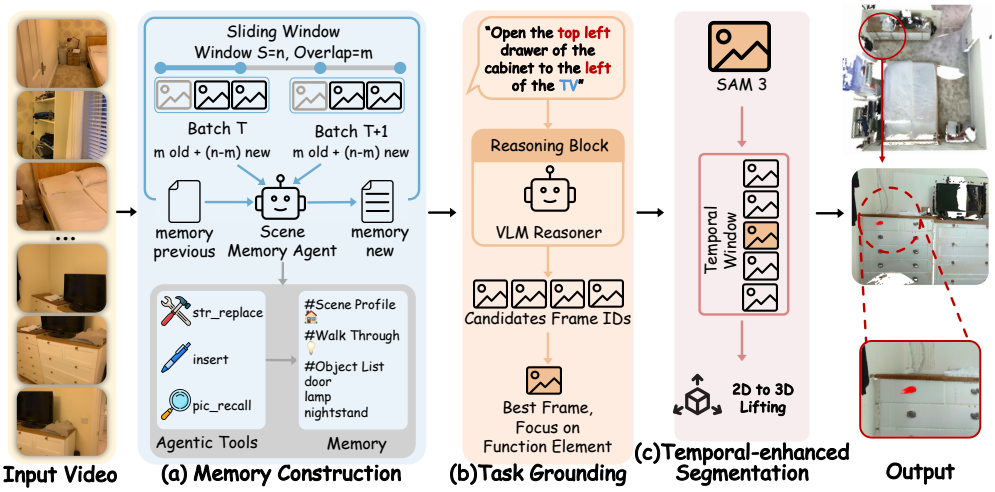

Figure 2: **MemAff** implements a "See Once, Query Many" paradigm for 3D affordance segmentation. (a) **Memory Construction**: an agentic profiler incrementally distils video batches into a structured Scene Memory under a sliding window constraint. (b) **Task Grounding**: a VLM reasons over the persistent memory to identify the best frame for a given instruction without re-scanning the sequence. (c) **Temporal-enhanced Segmentation**: the grounded frame initializes SAM-based mask propagation over a temporal window, followed by 2D-to-3D Lifting to generate the final 3D affordance segment.

Functional 3D Scene Graph. Scene representations encode object identities and relations for downstream reasoning (Linok et al., 2025; Huang et al., 2025; Liu et al., 2026; Takmaz et al., 2025; Gu et al., 2024; Koch et al., 2024). FunGraph (Rotondi et al., 2025) represents object-part hierarchies and functional structure, while OpenFunGraph (Zhang et al., 2025) also captures remote functional dependencies. MemAff shares the aim of relational grounding, but organizes this knowledge through agent-controlled textual edits and visual recall. Its memory links object descriptions directly to supporting frames, making the same abstraction usable for both instruction reasoning and segmentation-view retrieval.

VLMs for 3D Understanding. VLMs (Liu et al., 2023a; Chen et al., 2024a; Li et al., 2022; Alayrac et al., 2022; Chen et al., 2020; Halacheva et al., 2025; Petit et al., 2026) have advanced 3D scene understanding mainly through two paradigms: native 3D-aware models for direct spatial reasoning (Huang et al., 2023; Qi et al., 2025) and 2D VLMs adapted to 3D inputs via projected or lifted representations (Liu et al., 2023b; Huang et al., 2024a). Alongside tailored chain-of-thought prompting for embodied reasoning (Team et al., 2024; Li et al., 2023; Chen et al., 2024b; Zhu et al., 2023), these methods support tasks such as fine-grained affordance grounding. However, most VLM-based approaches (Mao et al., 2025; Jin et al., 2024; Hong et al., 2023; Mo et al., 2022; Xu et al., 2022) still reprocess scene observations for each instruction and lack a persistent scene abstraction, limiting their efficiency and long-term functional consistency in large-scale environments.

VLMs for 3D Understanding. VLM-based methods support spatial reasoning through 3D-aware modeling (Liu et al., 2023a; Chen et al., 2024a; Li et al., 2022; Alayrac et al., 2022; Chen et al., 2020; Halacheva et al., 2025; Petit et al., 2026) or by connecting 2D observations to 3D representations (Liu et al., 2023b; Huang et al., 2024a). These capabilities provide a basis for interpreting instructions and locating functional parts. MemAff uses a VLM as a tool-using memory agent: it determines which observations to retain, which records to revise, and when to recall earlier evidence. The resulting persistent abstraction separates scene-level knowledge acquisition from the visual computation required by each segmentation query.

3 METHOD

Given a scene traversal $\mathcal{V} = \{f_i\}_{i=1}^{N}$ with aligned depth maps D_i, intrinsics $\mathbf{K}_i$, camera-to-world poses $\mathbf{P}_i$, and scene points $\mathcal{X} = \{\mathbf{X}_j\}_{j=1}^{K}$, scene-level affordance segmentation predicts an instruction-conditioned mask $\mathbf{y}_\mathcal{T} \in \{0,1\}^K$. For multiple instructions in the same scene, object

identities, spatial relations, and functional associations can be acquired once and reused. MemAff therefore separates scene-wide perception from per-query grounding:

$$\mathcal{W} = \text{Build}_{\mathcal{A}}(\mathcal{V}), \qquad \mathcal{F}_{\mathcal{T}} = \text{Retrieve}(\mathcal{T}, \mathcal{W}), \qquad \mathbf{y}_{\mathcal{T}} = \text{Ground3D}(\mathcal{T}, \mathcal{V}_{\mathcal{F}_{\mathcal{T}}}, \mathcal{X}). \tag{1}$$

Agentic tools $\mathcal{A}$ organize observations into persistent memory $\mathcal{W}$, linking entity descriptions and relations to frame anchors. Each query retrieves relevant frames $\mathcal{F}_{\mathcal{T}}$ for local segmentation and 3D lifting (Fig. 2). This enables "See Once, Query Many": scene knowledge is shared across instructions, while query-specific visual processing is restricted to retrieved views and their temporal neighbors. With construction cost C_{build} and average online cost C_{online}, the mean cost over Q queries is

$$\overline{C}_{\text{MemAff}}(Q) = \frac{C_{\text{build}}}{Q} + C_{\text{online}}, \tag{2}$$

so the one-time scene processing is amortized as the number of queries increases.

3.1 Tool-driven Scene Memory Construction

Tools for accumulating, revising, and recalling evidence. The agent selects a sequence of tool calls $a_{t,1:L_t}$ conditioned on the current memory $\mathcal{W}^{(t-1)}$ and visual context $\mathcal{C}_t$. Their execution updates either the stored document or the accessible visual evidence:

$$\begin{aligned} \mathcal{A} &= \{\texttt{insert}, \texttt{str_replace}, \texttt{pic_recall}\}, \\ (\mathcal{W}^{(t)}, \mathcal{C}'_t) &= \text{Execute}(a_{t,1:L_t}; \mathcal{W}^{(t-1)}, \mathcal{C}_t). \end{aligned} \tag{3}$$

The three tools have complementary roles. `insert(line,text)` adds a newly observed entity, a complementary view, or a scene-level observation. `str_replace(old,new)` revises an exact, unique substring, allowing descriptions, identity assignments, and cross-references to change as evidence accumulates. The harness rejects replacements with zero or multiple matches. `pic_recall(frame_id)` brings an already observed frame back into context when its pixels are needed to resolve a later ambiguity. Recalled images count toward the same S-frame budget, so recall restores evidence without expanding the live visual window.

These tools support evidence-conditioned refinement: a later cabinet close-up can add drawer-specific descriptions and anchors to an existing record, while an ambiguous identity can trigger visual recall and a targeted correction. Atomic edits preserve unaffected text and emit only the update Δ_t rather than the entire document; the full memory remains part of the input. Snapshots and tool logs make revisions inspectable, but semantic correctness still depends on the VLM and its evidence.

Organizing an abstraction that supports later queries. The persistent memory is a Markdown document with three complementary sections:

$$\mathcal{W} = \langle \text{SP}, \text{WT}, \text{OL} \rangle, \qquad \text{OL} = \{e_j\}, \qquad e_j = \langle d_j, F_j, R_j \rangle. \tag{4}$$

The *Scene Profile* (SP) describes the global layout, functional zones, and scan coverage. It distinguishes observed, partially observed, and unobserved regions, providing context for interpreting missing evidence. The *Walkthrough* (WT) retains the traversal narrative and spatial context with inline frame anchors. It connects observations across functional zones and provides a fallback when an entity lacks a dedicated inventory entry. The *Object List* (OL) supplies the direct retrieval interface: each entity has a disambiguating description d_j, supporting frame IDs F_j, and textual spatial or functional references R_j. An optional Open Questions block records unresolved observations.

Equation 4 describes information roles in the document, not a separate graph. Markdown anchors `![note](frame_id)` connect each description to an observation even after its image expires.

Anchoring complementary views and linking functional partners. The agent removes redundant anchors among images in the live window and retains views exposing different aspects of an entity. Later windows can add complementary perspectives without deduplicating against pixels no longer visible. A simple object may need only an overview and a close-up; an appliance or multi-drawer cabinet can accumulate additional anchors for distinct controls or parts. The purpose is to retain useful evidence, rather than assign an equal number of views to every object.

Before finalizing the Object List, the agent performs a cross-link sweep through the editing tools. For identifiable target–instrument pairs, it adds bidirectional textual references, such as TV–remote and

appliance–outlet. When the partner's view exposes a control absent from the target's own anchors, that frame is recorded as complementary evidence. Directly observed associations are distinguished from plausible but inferred links. This provenance helps interpretation but does not certify semantic correctness. The completed memory is frozen for the evaluated queries; automatic revision from downstream execution feedback is outside the current setting.

3.2 Memory-guided Task Grounding

Resolving the instruction through memory. At query time, the VLM receives the instruction $\mathcal{T}$ and the completed memory $\mathcal{W}$. It first identifies the action, target, optional host, and optional instrument. The target denotes the entity involved in the requested outcome; a host contains a requested part; an instrument provides the means of interaction. Separating these roles prevents a contextual landmark or functional goal from being mistaken for the region that should be segmented.

Grounding then follows a structured retrieval procedure. The agent searches the Object List for matching entities, using attributes and spatial descriptions to distinguish instances. Part-based instructions resolve through the appropriate host and any part-specific references. Instrument-based instructions follow the stored functional links. Missing inventory entries trigger a search of Walkthrough anchors, after which the Scene Profile supplies a coverage check. If only part of the requested evidence is available, the agent reports that limitation; if the relevant area was unobserved, it can refuse grounding rather than substitute an unrelated object.

$$(\mathcal{E}_\mathcal{T}, p_\mathcal{T}) = \mathrm{Reason}(\mathcal{T}, \mathcal{W}), \qquad \mathcal{F}_\mathcal{T} \subseteq \bigcup_{e_j \in \mathcal{E}_\mathcal{T}} F_j, \tag{5}$$

where $\mathcal{E}_\mathcal{T}$ denotes entities retrieved from the Object List or Walkthrough and $p_\mathcal{T}$ the acting part. Candidate frames are selected from their bound evidence; Walkthrough anchors supplement object records when needed. The harness filters returned IDs against those present in the current memory. This validation excludes unindexed IDs while leaving semantic relevance to the grounding procedure.

For "turn on the TV using the remote on the nightstand," memory retrieves the linked TV and remote observations even if they never co-occur. For a drawer query relative to a TV, it distinguishes the TV landmark, cabinet host, and actionable drawer handle.

Selecting a segmentation-ready view. A frame useful for entity identification may still be unsuitable for segmentation: its actionable part can be small, blurred, oblique, or hidden. A VLM critic therefore inspects the retrieved candidates with an explicit acting-part visibility requirement. It discards views in which $p_\mathcal{T}$ is cropped, occluded, or outside the image and selects among the remaining candidates in a fixed priority order:

$$\mathcal{F}_\mathcal{T}^{\mathrm{vis}} = \{i \in \mathcal{F}_\mathcal{T} : \mathrm{Visible}(p_\mathcal{T}, f_i)\}, \qquad i^* = \operatorname*{lexargmax}_{i \in \mathcal{F}_\mathcal{T}^{\mathrm{vis}}} \big(P_i, Q_i, O_i\big). \tag{6}$$

Here P_i, Q_i, and O_i denote the critic's ordered assessments of affordance prominence, sharpness, and frontal framing. They express prompt-level priorities, not learned numerical scores.

Localizing contact regions. A separate VLM pass on f_{i^*} predicts positive point prompts $\mathcal{U} = \{(x_\ell, y_\ell)\}_{\ell=1}^{L} \subset [0,1]^2$, with coordinates normalized to image dimensions. The number of points follows the contact regions implied by the instruction: a toggle switch can require one point, while a task involving two handles can require multiple prompts. All points should belong to the intended physical unit. Drawer-front boundaries, hinge columns, and shared panel edges help distinguish repeated parts that appear close in the image but belong to different objects. These points transfer the resolved instruction from language and memory to a spatial prompt for segmentation.

3.3 Temporal Segmentation and Cross-view Lifting

Propagating localized evidence. SAM3 uses the positive points to initialize segmentation at the selected frame and propagates the mask through a short temporal neighborhood,

$$\mathcal{I} = \{i^* + hs : h = -H, \ldots, H\} \cap \{1, \ldots, N\}, \qquad \{B_i\}_{i \in \mathcal{I}} = \mathrm{SAM3}(\{f_i\}_{i \in \mathcal{I}}, \mathcal{U}, i^*), \tag{7}$$

Here B_i is the mask in view i. We use $H = 4$ and stride $s = 2$, yielding up to nine frames from the original scan sequence rather than the pose-sampled keyframe list.

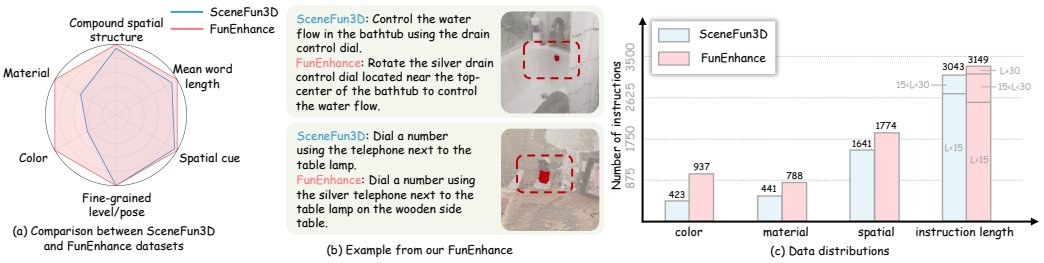

Figure 3: **Instruction extension on SceneFun3D.** FunEnhance adds attribute and relational descriptions while retaining the original scene split. (a) Semantic dimensions. (b) Instruction examples. (c) Attribute and instruction-length distributions.

Projecting scene points into mask evidence. A single-view mask can leak from a small knob or outlet onto the surrounding cabinet or wall. Direct lifting preserves these errors as off-affordance 3D points. MemAff instead projects the precomputed cropped scene point cloud into each temporal view and accumulates cross-view support. For point $\mathbf{X}_j$,

$$\mathbf{x}_{ij} = \mathbf{P}_i^{-1} \begin{bmatrix} \mathbf{X}_j \\ 1 \end{bmatrix}, \qquad \mathbf{u}_{ij} = \pi(\mathbf{K}_i \mathbf{x}_{ij,1:3}), \qquad b_{ij} = g_{ij}\, B_i(\mathbf{u}_{ij}), \tag{8}$$

where $\mathbf{x}_{ij}$ is the homogeneous camera-space point, π performs perspective division, and $g_{ij} \in \{0, 1\}$ indicates a valid, depth-consistent projection. Depth checks use the aligned observation, with a z-buffer penetration tolerance $\epsilon(d) = \max(0.02\,\mathrm{m}, 0.02d)$ at projected depth d. A point behind the observed surface therefore cannot receive support solely because its image projection falls inside the mask.

Fusing views into the 3D affordance. The final prediction thresholds the normalized cross-view vote:

$$v_j = \mathrm{Normalize}_{i \in \mathcal{I}} \left(\sum_{i \in \mathcal{I}} b_{ij} \right), \qquad y_{\mathcal{T},j} = \mathbb{1}[v_j \geq 0.5]. \tag{9}$$

Cross-view support suppresses spatially inconsistent mask leakage. Each query reuses the precomputed scene geometry and performs only mask-conditioned projection and voting, completing the memory-to-frame-to-3D grounding process.

4 Datasets

We evaluate on SceneFun3D (Delitzas et al., 2024) and its instruction-level extension, FunEnhance. FunEnhance contains 3,149 instructions with additional color, material, pose, level, and compound spatial descriptions (Fig. 3). We retain the original SceneFun3D training/validation split and scene geometry with its affordance masks; the extension changes the language descriptions associated with those targets. Instructions remain in the split of their source scene. This setting evaluates grounding under richer linguistic constraints without changing the underlying train/validation scene partition.

5 Experiments

Model Settings. We use Qwen3.5-35B as the vision-language backbone for agentic memory construction, instruction reasoning, candidate-view selection, and point localization, and SAM3 for point-prompted segmentation and temporal mask propagation. MemAff requires no task-specific fine-tuning. Inference runs on an NVIDIA A100 GPU with 80 GB memory. Unless otherwise stated, all experiments use the memory and geometric settings in Sec. 3.

Evaluation Protocol. We use the SceneFun3D and FunEnhance validation sets with the unchanged scene split described in Sec. 4. Following the benchmark protocol, we report mIoU, mAP, AP_{50}, AP_{25}, and their recall counterparts mAR, AR_{50}, and AR_{25}. Table 1 compares open-vocabulary methods, training-based affordance methods, and the training-free Fun3DU baseline. Table 3 separately evaluates learning-based models trained with the expanded FunEnhance instructions.

Table 1: **Quantitative comparison on the SceneFun3D validation set.** Best results are shown in **bold**, and second-best results are underlined.

Method	mAP	AP_{50}	AP_{25}	mAR	AR_{50}	AR_{25}	mIoU
Open-vocabulary							
OpenMask3D (Takmaz et al., 2023)	0.2	0.2	0.4	20.3	24.5	27.0	0.2
LERF (Kerr et al., 2023)	0.0	0.0	0.0	34.2	35.1	36.0	0.0
OpenIns3D (Huang et al., 2024b)	0.0	0.0	0.0	**40.5**	46.7	51.5	0.1
Training-based							
TASA (He et al., 2026)	**23.2**	26.9	28.6	25.8	34.9	44.7	19.7
AffordBot (Wang et al., 2025)	15.5	20.0	23.3	16.7	19.0	22.8	14.0
FunGraph (Rotondi et al., 2025)	5.9	16.0	30.3	–	–	–	–
Train-free, see-once, query-once							
Fun3DU (Corsetti et al., 2025)	7.6	16.9	33.3	27.4	38.2	46.7	15.2
Train-free, see-once, query-many							
Ours	17.4	**30.5**	**48.4**	36.8	**51.1**	**60.8**	**24.9**

Table 2: **Inference-time of MemAff on a SceneFun3D scene.** The *offline* phase processes the scan once and is amortized across instructions, while the *online* phase runs for each instruction.

Phase	Stage	Time	Frequency
Offline	Memory Construction	$\sim$30 s $\times$ 20 $=\sim$10 m	once per scene
Online	Memory Reasoning	$\sim$30 s	per instruction
	Frame Selection	$\sim$8 s	
	Point Localization	$\sim$8 s	
	Cross-view Segmentation	$\sim$3 s	
	2D-to-3D Lifting	$\sim$2 s	
	Online total	**$\sim$51 s**	**per instruction**

5.1 Quantitative Results on SceneFun3D

Segmentation Accuracy. Table 1 shows that MemAff improves over Fun3DU on all reported metrics, raising mAP from 7.6 to 17.4 and mIoU from 15.2 to 24.9. MemAff obtains the highest AP_{50}, AP_{25}, AR_{50}, AR_{25}, and mIoU among the compared methods. TASA retains the highest mAP (23.2), while OpenIns3D has the highest mAR (40.5) but near-zero precision. These results support the effectiveness of memory-guided grounding and local multi-view segmentation without task-specific training.

Inference Efficiency. Table 2 shows MemAff's "see-once, query-many" pipeline: the offline stage builds a persistent Scene Memory once, requiring about 30s per inference and 20 inferences per scene, while the online stage processes each new instruction in about 51s. In contrast, prior methods (Corsetti et al., 2025; He et al., 2026) typically traverse scene frames for each instruction, resulting in more than 20 minutes of inference time per instruction on the same scene. By amortizing scene understanding across queries, MemAff substantially reduces per-instruction latency while avoiding repeated frame-level search.

5.2 Evaluation with Expanded Instructions

Table 3 evaluates instruction difficulty and transfer across validation sets. Fun3DU decreases from 7.6 to 6.3 mAP and from 16.9 to 14.7 AP_{50} on FunEnhance; MemAff decreases from 17.4 to 15.3 mAP and from 30.5 to 24.3 AP_{50}. MemAff remains ahead of Fun3DU under the richer language constraints. Learning-based models trained on FunEnhance also retain performance on the original SceneFun3D validation set: AffordBot and TASA reach 15.5 and 24.0 mAP, respectively. These results characterize the expanded-instruction evaluation while retaining the original scene split.

5.3 Qualitative Results

Fig. 7 presents qualitative comparisons on representative indoor scenes. MemAff produces more accurate and consistent target grounding, particularly in scenarios involving multiple similar targets, fine-grained actionable parts, or remote functional links. These examples show that structured Scene

Table 3: **Validation of FunEnhance.** We evaluate training-free and learning-based methods across test sets. The results show that FunEnhance provides challenging evaluation and useful supervision from expanded semantic cues.

Training-free Methods						Learning-based Methods					
Method	Val Set	mAP	AP_{50}	AP_{25}	mIoU	Method	Val Set	mAP	AP_{50}	AP_{25}	mIoU
Fun3DU	SceneFun3D	7.6	16.9	33.3	15.2	AffordBot	SceneFun3D	15.5	20.0	23.3	14.0
Fun3DU	FunEnhance	6.3	14.7	30.8	12.5	AffordBot	FunEnhance	12.8	17.6	21.6	13.1
Ours	SceneFun3D	17.4	**30.5**	**48.4**	**24.9**	TASA	SceneFun3D	**24.0**	27.4	30.1	21.7
Ours	FunEnhance	15.3	**24.3**	**44.7**	**22.0**	TASA	FunEnhance	**18.3**	24.2	25.5	16.6

Table 4: **Ablation on Memory Construction & Lifting.**

Edit tools	Picture recall	Sliding Window S	Sliding Window k	Multi view	mAP	AP_{50}	AP_{25}	mIoU
		6	6	✓	4.0	7.5	14.5	8.2
✓		6	6	✓	9.5	16.5	28.5	14.5
✓	✓	6	6	✓	11.5	19.5	32.0	17.0
✓	✓	6	4		14.6	25.2	38.7	22.4
✓	✓	6	4	✓	**17.4**	**30.5**	**48.4**	**24.9**
✓	✓	4	2	✓	15.0	26.5	39.6	22.6
✓	✓	10	4	✓	16.4	28.7	42.0	23.6

Table 5: **Ablation on Scene Memory Structure.** Leave-one-out over the three sections: *Scene Profile* (SP), *Walkthrough* (WT), *Object List* (OL).

SP	WT	OL	mAP	AP_{50}	AP_{25}	mIoU
	✓	✓	16.2	27.9	41.5	22.5
✓		✓	15.4	26.4	39.6	21.3
✓	✓		12.1	20.8	31.7	17.1
✓	✓	✓	**17.4**	**30.5**	**48.4**	**24.9**

Memory helps the model resolve contextual distractors and localize the intended affordance rather than the most visually salient nearby object.

We further evaluate the zero-shot generalization of MemAff on ScanNet++ Yeshwanth et al. (2023) without fine-tuning. Despite domain gaps in scene layout, object appearance, and category distribution, MemAff consistently produces accurate 3D masks (Fig. 6). This suggests that structured Scene Memory captures transferable functional relations and supports robust grounding in novel indoor environments.

5.4 Ablation Studies

We conduct two ablations. Table 4 analyzes the memory-construction harness and lifting strategy, while Table 5 evaluates the contribution of each section in the structured scene memory.

Editing and Visual Recall. With $S = k = 6$, enabling atomic editing raises mAP from 4.0 to 9.5 and mIoU from 8.2 to 14.5. Adding `pic_recall` further improves these metrics to 11.5 and 17.0. The improvements support the role of selective updates and recoverable evidence in building a useful scene abstraction. These rows measure segmentation quality; the aggregate runtime is reported separately in Table 4.

Visual Window. With editing, recall, and multi-view lifting enabled, changing (S, k) from $(6, 6)$ to $(6, 4)$ increases mAP from 11.5 to 17.4 as depicted in 5. The two-frame overlap provides continuity for reconciling observations while retaining pressure to write persistent knowledge. The configurations $(4, 2)$ and $(10, 4)$ obtain 15.0 and 16.4 mAP, respectively. Among the tested settings, $(6, 4)$ provides the best balance between live evidence and memory externalization.

Cross-view Lifting. With all other components enabled, replacing single-frame 2D-to-3D lifting with cross-view voting over a 9-frame temporal window improves AP_{50} from 25.2 to 30.5 (Table 4, row 4 vs. gray). Cross-view voting suppresses off-affordance "halo" points caused by single-frame mask leakage, yielding tighter and more geometrically consistent 3D affordance segments without per-query optimization. The full MemAff configuration reaches 17.4 mAP and 24.9 mIoU.

Memory Organization. Table 5 shows that removing the *Object List* causes the largest drop, reducing mAP from 17.4 to 12.1, because task grounding primarily relies on this section to retrieve object-specific `frame_ids`. Removing the *Walkthrough* reduces mAP to 15.4, showing the value of inline frame anchors as a fallback when a target is missing from the inventory. Removing the Scene Profile leads to the smallest numerical drop, but its coverage caveat remains important for avoiding spurious grounding in unobserved regions. These results show that the three sections are complementary, and the full $\langle \text{SP}, \text{WT}, \text{OL} \rangle$ structure achieves the best overall performance.

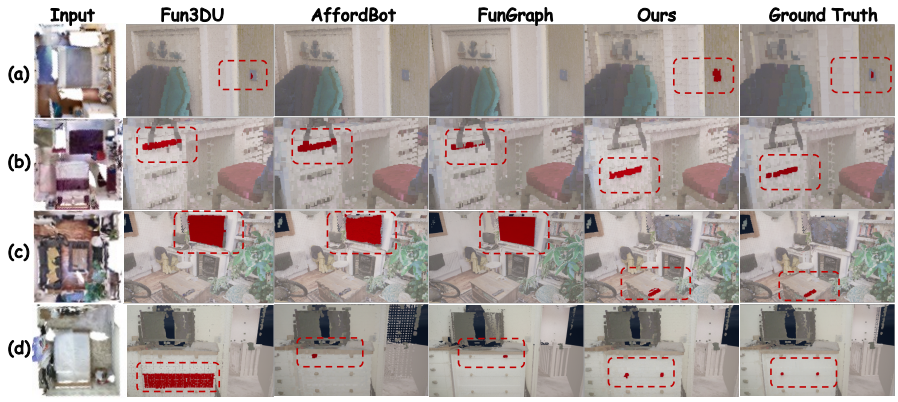

Figure 4: **Qualitative Results.** (a) "Turn on the ceiling light." (b) "Open the left middle drawer of the dressing table." (c) "Turn on the TV." (d) "Open the second drawer of the cabinet with a TV on top."

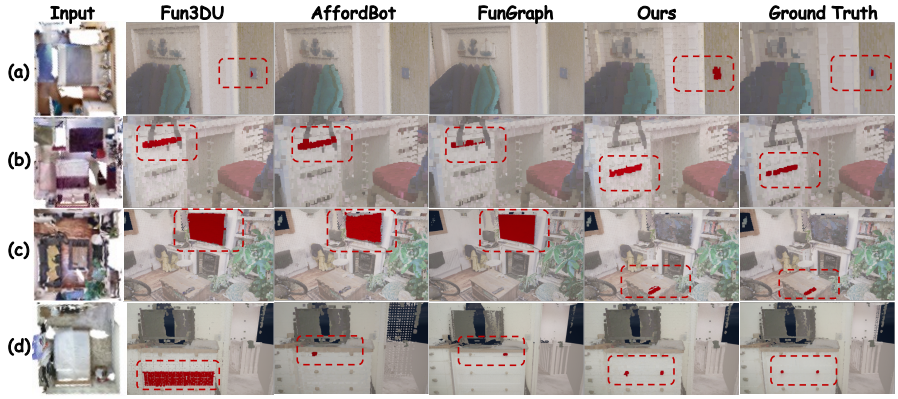

Figure 7: **Qualitative Results.** (a) "Turn on the ceiling light." (b) "Open the left middle drawer of the dressing table." (c) "Turn on the TV." (d) "Open the second drawer of the cabinet with a TV on top."

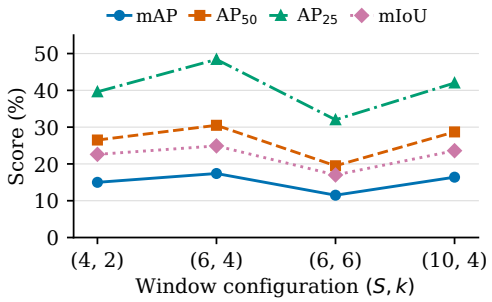

Figure 5: **Sliding-window ablation** with all tools and multi-view lifting.

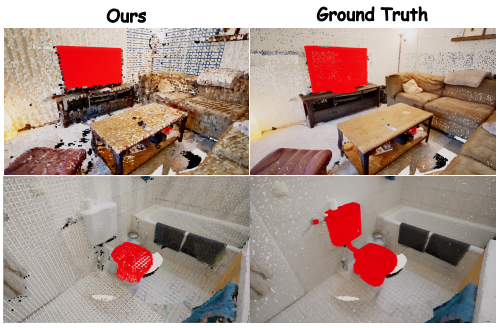

Figure 6: **Zero-shot object segmentation on ScanNet++.**

6 Conclusion

We present MemAff, a training-free framework for 3D scene-level affordance segmentation. Agentic editing and visual recall organize observations into persistent memory that links scene knowledge to visual evidence. This memory guides instruction grounding and targeted segmentation, enabling "See Once, Query Many" inference. By reusing scene knowledge across queries, MemAff reduces repeated visual processing while preserving precise affordance localization.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools for [tasks with required disclosure]. We have not used generative AI tools for [other tasks with required disclosure], and [the rest of the required disclosure tasks] are not applicable to this work. Additionally, we used generative AI tools for [tasks with recommended disclosure]. We have reviewed all AI-assisted work. [Elaborate. For example, "we checked LLM-generated research ideas for potential plagiarism through a manual literature survey", "LLM-generated code was verified and tested for correctness by 2 authors", etc.]. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

See the ICLR 2027 AI Policy for Authors for more details. This statement should not be more than 1 page.

ETHICS STATEMENT

(This section is **recommended** and does not count toward the page limit.)

If authors feel that their paper submission raises questions regarding the Code of Ethics, they are encouraged to include a paragraph of Ethics Statement (at the end of the main text before references) to address potential concerns where appropriate. Topics include, but are not limited to, studies that involve human subjects, practices to data set releases, potentially harmful insights, methodologies and applications, potential conflicts of interest and sponsorship, discrimination/bias/fairness concerns, privacy and security issues, legal compliance, and research integrity issues (e.g., IRB, documentation, research ethics). This statement should not be more than 1 page.

REPRODUCIBILITY STATEMENT

(This section is **recommended** and does not count toward the page limit.)

It is important that the work published in ICLR is reproducible. Authors are strongly encouraged to include a paragraph-long Reproducibility Statement at the end of the main text (before references) to discuss the efforts that have been made to ensure reproducibility. This paragraph should not itself describe details needed for reproducing the results, but rather reference the parts of the main paper, appendix, and supplemental materials that will help with reproducibility. For example, for novel models or algorithms, a link to an anonymous downloadable source code can be submitted as supplementary materials; for theoretical results, clear explanations of any assumptions and a complete proof of the claims can be included in the appendix; for any datasets used in the experiments, a complete description of the data processing steps can be provided in the supplementary materials. Each of the above are examples of things that can be referenced in the reproducibility statement.

AUTHOR CONTRIBUTIONS

If you'd like to, you may include a section for author contributions as is done in many journals. This is optional and at the discretion of the authors.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

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

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

Dave Zhenyu Chen, Angel X Chang, and Matthias Nießner. Scanrefer: 3d object localization in rgb-d scans using natural language. In *European Conference on Computer Vision*, pp. 202–221. Springer, 2020.

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

Jaime Corsetti, Francesco Giuliari, Alice Fasoli, Davide Boscaini, and Fabio Poiesi. Functionality understanding and segmentation in 3d scenes. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 24550–24559, 2025.

Claudia Cuttano, Gabriele Rosi, Gabriele Trivigno, and Giuseppe Averta. What does clip know about peeling a banana? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2238–2247, 2024.

Alexandros Delitzas, Ayca Takmaz, Federico Tombari, Robert Sumner, Marc Pollefeys, and Francis Engelmann. Scenefun3d: Fine-grained functionality and affordance understanding in 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14531–14542, 2024.

Shengheng Deng, Xun Xu, Chaozheng Wu, Ke Chen, and Kui Jia. 3d affordancenet: A benchmark for visual object affordance understanding. In *proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1778–1787, 2021.

Thanh-Toan Do, Anh Nguyen, and Ian Reid. Affordancenet: An end-to-end deep learning approach for object affordance detection. In *2018 IEEE International Conference on Robotics and Automation*, pp. 5882–5889. IEEE, 2018.

James J Gibson. *The ecological approach to visual perception: classic edition*. Psychology press, 2014.

Qiao Gu, Ali Kuwajerwala, Sacha Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aditya Agarwal, Corban Rivera, William Paul, Kirsty Ellis, Rama Chellappa, et al. Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning. In *2024 IEEE International Conference on Robotics and Automation*, pp. 5021–5028. IEEE, 2024.

Anna-Maria Halacheva, Yang Miao, Jan-Nico Zaech, Xi Wang, Luc Van Gool, and Danda Pani Paudel. Articulate3d: Holistic understanding of 3d scenes as universal scene description. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5633–5644, 2025.

Lian He, Meng Liu, Qilang Ye, Yu Zhou, Xiang Deng, and Gangyi Ding. Task-aware 3d affordance segmentation via 2d guidance and geometric refinement. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 4654–4662, 2026.

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.

Haifeng Huang, Yilun Chen, Zehan Wang, Rongjie Huang, Runsen Xu, Tai Wang, Luping Liu, Xize Cheng, Yang Zhao, Jiangmiao Pang, et al. Chat-scene: Bridging 3d scene and large language models with object identifiers. *Advances in Neural Information Processing Systems*, 37:113991–114017, 2024a.

Jiangyong Huang, Silong Yong, Xiaojian Ma, Xiongkun Linghu, Puhao Li, Yan Wang, Qing Li, Song-Chun Zhu, Baoxiong Jia, and Siyuan Huang. An embodied generalist agent in 3d world. *arXiv preprint arXiv:2311.12871*, 2023.

Zeyi Huang, Yuyang Ji, Xiaofang Wang, Nikhil Mehta, Tong Xiao, Donghyun Lee, Sigmund Vanvalkenburgh, Shengxin Zha, Bolin Lai, Licheng Yu, et al. Building a mind palace: Structuring environment-grounded semantic graphs for effective long video analysis with llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24169–24179, 2025.

Zhening Huang, Xiaoyang Wu, Xi Chen, Hengshuang Zhao, Lei Zhu, and Joan Lasenby. Openins3d: Snap and lookup for 3d open-vocabulary instance segmentation. In *European Conference on Computer Vision*, pp. 169–185. Springer, 2024b.

Bu Jin, Yupeng Zheng, Pengfei Li, Weize Li, Yuhang Zheng, Sujie Hu, Xinyu Liu, Jinwei Zhu, Zhijie Yan, Haiyang Sun, et al. Tod3cap: Towards 3d dense captioning in outdoor scenes. In *European Conference on Computer Vision*, pp. 367–384. Springer, 2024.

Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19729–19739, 2023.

Sebastian Koch, Narunas Vaskevicius, Mirco Colosi, Pedro Hermosilla, and Timo Ropinski. Open3dsg: Open-vocabulary 3d scene graphs from point clouds with queryable objects and open-set relationships. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14183–14193, 2024.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning*, pp. 12888–12900. PMLR, 2022.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International Conference on Machine Learning*, pp. 19730–19742. PMLR, 2023.

Jiaying Lin and Dan Xu. Unifunc3d: Unified active spatial-temporal grounding for 3d functionality segmentation. In *arXiv*, 2026.

Sergey Linok, Tatiana Zemskova, Svetlana Ladanova, Roman Titkov, Dmitry Yudin, Maxim Monastyrny, and Aleksei Valenkov. Beyond bare queries: Open-vocabulary object grounding with 3d scene graph. In *2025 IEEE International Conference on Robotics and Automation*, pp. 13582–13589. IEEE, 2025.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in Neural Information Processing Systems*, 36:34892–34916, 2023a.

Youquan Liu, Lingdong Kong, Jun Cen, Runnan Chen, Wenwei Zhang, Liang Pan, Kai Chen, and Ziwei Liu. Segment any point cloud sequences by distilling vision foundation models. *Advances in Neural Information Processing Systems*, 36:37193–37229, 2023b.

Yuanyuan Liu, Haiyang Mei, Dongyang Zhan, Jiayue Zhao, Dongsheng Zhou, Bo Dong, and Xin Yang. View-on-graph: Zero-shot 3d visual grounding via vision-language reasoning on scene graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 7386–7394, 2026.

Yongsen Mao, Junhao Zhong, Chuan Fang, Jia Zheng, Rui Tang, Hao Zhu, Ping Tan, and Zihan Zhou. Spatiallm: Training large language models for structured indoor modeling. *arXiv preprint arXiv:2506.07491*, 2025.

Kaichun Mo, Leonidas J Guibas, Mustafa Mukadam, Abhinav Gupta, and Shubham Tulsiani. Where2act: From pixels to actions for articulated 3d objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6813–6823, 2021.

Kaichun Mo, Yuzhe Qin, Fanbo Xiang, Hao Su, and Leonidas Guibas. O2o-afford: Annotation-free large-scale object-object affordance learning. In *Conference on Robot Learning*, pp. 1666–1677. PMLR, 2022.

Tushar Nagarajan and Kristen Grauman. Learning affordance landscapes for interaction exploration in 3d environments. *Advances in Neural Information Processing Systems*, 33:2005–2015, 2020.

Chuanruo Ning, Ruihai Wu, Haoran Lu, Kaichun Mo, and Hao Dong. Where2explore: Few-shot affordance learning for unseen novel categories of articulated objects. *Advances in Neural Information Processing Systems*, 36:4585–4596, 2023.

Doriand Petit, Steve Bourgeois, Vincent Gay-Bellile, Florian Chabot, and Loïc Barthe. Llava^3: Representing 3d scenes like a cubist painter to boost 3d scene understanding of vlms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 8412–8420, 2026.

Zhangyang Qi, Zhixiong Zhang, Ye Fang, Jiaqi Wang, and Hengshuang Zhao. Gpt4scene: Understand 3d scenes from videos with vision-language models. *arXiv preprint arXiv:2501.01428*, 2025.

Dennis Rotondi, Fabio Scaparro, Hermann Blum, and Kai O Arras. Fungraph: Functionality aware 3d scene graphs for language-prompted scene interaction. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4083–4090. IEEE, 2025.

Ayça Takmaz, Elisabetta Fedele, Robert W Sumner, Marc Pollefeys, Federico Tombari, and Francis Engelmann. Openmask3d: Open-vocabulary 3d instance segmentation. *arXiv preprint arXiv:2306.13631*, 2023.

Ayca Takmaz, Alexandros Delitzas, Robert W Sumner, Francis Engelmann, Johanna Wald, and Federico Tombari. Search3d: Hierarchical open-vocabulary 3d segmentation. *IEEE Robotics and Automation Letters*, 2025.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Xinyi Wang, Xun Yang, Yanlong Xu, Yuchen Wu, Zhen Li, and Na Zhao. Affordbot: 3d fine-grained embodied reasoning via multimodal large language models. *arXiv preprint arXiv:2511.10017*, 2025.

Lin Wu, Wei Wei, Peizhuo Yu, and Jianglin Lan. Open-vocabulary 3d affordance understanding via functional text enhancement and multilevel representation alignment. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 7988–7997, 2025.

Ruihai Wu, Kai Cheng, Yan Zhao, Chuanruo Ning, Guanqi Zhan, and Hao Dong. Learning environment-aware affordance for 3d articulated object manipulation under occlusions. *Advances in Neural Information Processing Systems*, 36:60966–60983, 2023.

Chao Xu, Yixin Chen, He Wang, Song-Chun Zhu, Yixin Zhu, and Siyuan Huang. Partafford: Part-level affordance discovery from 3d objects. *arXiv preprint arXiv:2202.13519*, 2022.

Yuhang Yang, Wei Zhai, Chengfeng Wang, Chengjun Yu, Yang Cao, and Zheng-Jun Zha. Egochoir: Capturing 3d human-object interaction regions from egocentric views. *Advances in Neural Information Processing Systems*, 37:54529–54557, 2024.

Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. Scannet++: A high-fidelity dataset of 3d indoor scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12–22, 2023.

Chunlin Yu, Hanqing Wang, Ye Shi, Haoyang Luo, Sibei Yang, Jingyi Yu, and Jingya Wang. Seqafford: Sequential 3d affordance reasoning via multimodal large language model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 1691–1701, 2025.

Chenyangguang Zhang, Alexandros Delitzas, Fangjinhua Wang, Ruida Zhang, Xiangyang Ji, Marc Pollefeys, and Francis Engelmann. Open-vocabulary functional 3d scene graphs for real-world indoor spaces. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 19401–19413, 2025.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.

A Implementation and Memory Interface

MemAff uses a lightweight custom Python harness to connect scene browsing, memory editing, instruction grounding, view selection, point localization, segmentation, and 3D lifting. The harness supplies observations and executes tools; the VLM decides which descriptions and relations to record or revise. The resulting interface connects a persistent scene abstraction to addressable visual evidence and ultimately to a 3D affordance mask.

A.1 System Configuration

Table 6 summarizes the configuration of the main Qwen3.5-35B setting. Pose sampling limits the total keyframe set, whereas the live-frame budget limits the images visible in an individual reasoning context. These are different budgets: the live-window experiments vary S and k, not the total keyframe cap.

Table 6: Configuration of the main MemAff setting.

Component	Setting
Agent orchestration	Custom Python harness
Memory, grounding, view and point reasoning	Qwen3.5-35B; no task-specific fine-tuning
Segmentation and temporal propagation	SAM3
Inference hardware	NVIDIA A100, 80 GB
Pose-sampling thresholds	0.25 m translation; 12° rotation
Minimum sampling interval / keyframe cap	0.4 s / 80 keyframes per scene
Live visual window / new-frame batch	$S = 6$ / $k = 4$
Temporal segmentation window	$H = 4$, stride $s = 2$; up to nine frames
3D vote threshold	0.5
Depth penetration tolerance	$\max(0.02\,\mathrm{m}, 0.02d)$ at depth d

A.2 Programmatic Tool Semantics

At each browse turn, the harness supplies a fresh keyframe batch and the complete current memory snapshot. The agent emits function calls, which the harness executes against the stored document:

- `str_replace(old_string,new_string)` replaces an exact, unique substring. An edit is rejected if the old string occurs zero times or more than once, preventing an ambiguous write target.
- `insert(line,text)` inserts text at the specified line. This operation adds new observations or records without regenerating unaffected content.
- `pic_recall(frame_id)` retrieves the image associated with a valid frame ID already encountered in the traversal. The recalled image consumes a live-frame slot, and older evidence expires when the window is full.

The harness stores the updated document, per-turn memory snapshots, transcript, VLM-call records, and browse log. These records make updates traceable to individual tool calls. They do not establish that the semantic content of an accepted edit is correct.

A.3 Frame Anchors and Query-time Validation

A frame identifier combines a scene/video identifier and a timestamp. For example, `42444754_80966.897` denotes an addressable observation rather than an object label. Memory entries bind descriptions to these observations using Markdown anchors of the form `![note](frame_id)`. Two examples are:

```
![nightstand with lamp and remotes](42444754_80981.174)
![white door with silver handle](42444754_80972.378)
```

These anchors remain valid after their image pixels expire from the live context. At query time, the VLM decomposes an instruction into a target, an optional host, and an optional instrument, then returns candidate frame IDs. The harness filters these candidates against a whitelist of IDs present in the current memory. A returned ID outside this whitelist is not accepted. This check ensures that downstream processing uses indexed evidence; it does not ensure that the selected evidence depicts the correct target.

A.4 Functional Relations and Evidence-conditioned Refinement

Functional links are proposed by the VLM from observed frames and accumulated memory. A cross-link sweep adds bidirectional references between identifiable targets and instruments, together with complementary frame anchors. For instance, the TV entry can reference a remote observed on a nightstand, while the remote entry references the TV. The query agent follows these links to recover evidence for both the functional goal and the actionable instrument.

The provenance of a relation matters. An observed relation is directly supported by the linked visual evidence; an inferred relation is plausible across observations but not directly witnessed. A relation can also be updated when later evidence changes its interpretation. An inference label records this distinction without certifying the relation's truth. There is no separate automatic relation-level verifier.

Memory construction is not append-only. Later views can trigger exact replacement of an earlier description, identity assignment, or relation. We call this behavior *evidence-conditioned refinement*. In the evaluated setting, memory is frozen after construction and reused across queries. Updating it from task-execution failures or user feedback is a possible extension, rather than an evaluated closed-loop capability.

Table 7: Enforcement boundary of the memory interface. Programmatic validity is distinct from semantic correctness.

Mechanism	Enforcement
Exact edit execution, snapshots, and logs	Programmatic
Acceptance of query frame IDs present in memory	Programmatic
Object descriptions and identity continuity	VLM/prompt-guided
Recognition and revision of conflicting entries	VLM/prompt-guided
Inference of functional dependencies	VLM/prompt-guided

B Additional Controlled Experiments

B.1 Matched Controls across Vision-language Models

For each VLM, the direct-frame control receives the same pose-downsampled frames as MemAff. It scans those frames with a sliding window and selects a task-relevant frame separately for each query, without constructing persistent memory. The paired MemAff setting first organizes the identical observations into scene memory and retrieves evidence from this memory for subsequent instructions. Table 8 reports the paired results.

Table 8: Matched-frame comparison with and without persistent scene memory. All metrics are percentages; higher is better.

VLM	Scene memory	mAP	AP_{50}	AP_{25}	mIoU
Qwen3-VL-32B	No	4.31	7.69	16.77	8.74
Qwen3-VL-32B	Yes	16.29	28.74	42.31	20.37
Qwen3.6-35B	No	6.67	10.33	21.67	13.36
Qwen3.6-35B	Yes	18.35	32.09	49.00	26.24
Qwen3.5-35B	No	5.45	9.09	18.18	10.70
Qwen3.5-35B	Yes	17.40	30.50	48.40	24.90

Memory improves mAP by 11.98, 11.68, and 11.95 percentage points for the three backbones, respectively. Because each pair uses the same input frames, additional raw visual evidence does not explain these gains. The comparison supports the utility of organizing observations into queryable records and frame links. Absolute performance still depends on the VLM; the results demonstrate consistency across the three evaluated backbones, not model independence.

B.2 Stage-level Attribution

Table 9 tests alternatives to memory construction, structured task grounding, and temporal enhancement under the Qwen3.5-35B setting. The first variant replaces persistent memory with same-frame sliding-window selection. The second replaces structured task grounding, including instrument retrieval, with instruction–frame similarity. The third removes temporal propagation and cross-view enhancement.

Table 9: Stage-level controls. The temporal control jointly removes propagation and cross-view enhancement. All metrics are percentages; higher is better.

Variant	mAP	AP_{50}	AP_{25}	mIoU
Same-frame sliding-window selection, no memory	5.45	9.09	18.18	10.70
Instruction–frame similarity selection	13.10	20.70	30.20	20.70
Without temporal enhancement	14.60	25.20	38.70	22.40
Full MemAff	17.40	30.50	48.40	24.90

The full pipeline performs best in each reported metric. Relative to similarity selection, structured grounding improves AP_{50} by 9.80 percentage points, consistent with the value of resolving target, host, and instrument roles. Relative to the temporal control, the full pipeline improves AP_{50} by 5.30 points. Since that control changes propagation and fusion together, it does not isolate either operation individually. The memory-section ablation in Table 5 provides a complementary test of the three-part document organization.

B.3 Comparison with a Graph-memory Prototype

The graph-memory prototype uses object nodes, affordance child nodes, and spatial/functional edges. A VLM incrementally constructs the graph while traversing the frames; a graph-retrieval module then supplies relevant objects, affordances, and relations for a query. Table 10 compares this prototype with the Markdown memory under the same evaluated VLM and SAM3 backend.

Table 10: Graph-memory prototype versus Markdown memory with the same VLM and SAM3 backend. All metrics are percentages; higher is better.

Memory	mAP	AP_{50}	AP_{25}	mAR	AR_{50}	AR_{25}	mIoU
Graph	6.96	15.89	32.98	26.26	28.43	37.13	16.02
Markdown	17.40	30.50	48.40	36.80	51.10	60.80	24.90

The Markdown pipeline improves mAP from 6.96 to 17.40 and mIoU from 16.02 to 24.90. Two practical issues arose in the tested graph pipeline. First, retrieval could omit part of a relation chain needed for a relative or multi-hop instruction. Encoding both "the TV is on the table" and "the table is opposite the bed" is insufficient unless both relations are retrieved and composed when needed. Second, incremental graph construction requires object-identity resolution, node merging, and edge revision as observations accumulate. The Markdown interface expresses context, uncertainty, and exceptions directly in text and allows local revision with the editing tools.

These results compare two implemented memory pipelines, including their retrieval interfaces. They do not establish an inherent expressive limitation of graphs or a universal advantage of Markdown serialization. The benchmark comparison with FunGraph in Table 1 is separate: those systems also differ in components beyond memory representation.

B.4 Traversal Order and Live-frame Budget

We perturb observation order while preserving the trajectory structure, using reverse traversal and a random-start cyclic shift. Table 11 uses Qwen3.5-35B throughout. Fully random frame shuffling is not part of this evaluation.

Table 11: Sensitivity to traversal order. All metrics are percentages; higher is better.

Traversal	mAP	AP_{50}	AP_{25}	mIoU
Original	17.40	30.50	48.40	24.90
Reverse	15.45	26.67	40.84	21.24
Random-start cyclic shift	16.84	28.05	43.13	22.50

Reversal and cyclic shifting reduce mAP by 1.95 and 0.56 points, respectively. Thus, the memory-building process is sensitive to observation order. A cyclic shift can also introduce a spatial discontinuity when a trajectory is not a closed loop and its endpoints are far apart.

For live-context budgets, Table 4 reports 15.0/22.6, 17.4/24.9, and 16.4/23.6 mAP/mIoU for $(S, k) = (4, 2)$, $(6, 4)$, and $(10, 4)$, respectively. These settings support $(6, 4)$ among the tested window configurations. They do not constitute a sweep over the total number of sampled keyframes. No controlled prompt-paraphrase experiment is available, so prompt invariance is not claimed.

B.5 Comparison with AFFORDMEM

Both AFFORDMEM and MemAff use memory for training-free affordance grounding, but they organize different evidence. AFFORDMEM uses cross-scene affordance exemplars or priors for localization in a target scene. MemAff builds a memory of the current scene, binds its records to observed frames, and reuses those anchors for view selection, contact localization, and 3D fusion.

Table 12: Comparison on SceneFun3D Split0 under the same VLM and SAM setting. The MemAff row uses the Qwen3-VL-32B setting, distinct from the main Qwen3.5-35B result. All metrics are percentages; higher is better.

Method	AP_{50}	AP_{25}	mIoU
AFFORDMEM	20.13	41.66	17.32
MemAff	28.74	42.31	20.37

The comparison yields gains of 8.61 AP_{50}, 0.65 AP_{25}, and 3.05 mIoU points. It compares complete systems under a shared foundation-model setting, rather than isolating a single memory operation.

C Evaluation Protocol and Computational Cost

C.1 Benchmark Comparisons versus Matched Controls

The headline benchmark table and the matched-frame controls answer different questions. The baseline entries in Table 1 follow their original visual, model, and segmentation protocols; they are not all reruns with a common VLM and SAM backend. Fun3DU and TASA start from dense posed RGB-D views and perform task-aware view selection. MemAff instead constructs memory from pose-sampled keyframes, with at most 80 stored keyframes and six live images in the browsing context. Thus, the benchmark comparison does not enforce component equality. Table 8 is the direct control for the memory interface because each pair receives the same downsampled observations.

C.2 Offline and Online Accounting

The timing profile in Table 2 gives approximately 600 s for memory construction and 51 s per subsequent instruction. Online processing comprises memory reasoning, candidate-view selection,

Table 13: Access and matching conditions established for the comparisons. NR means not reported in the available experiment description; equality is not assumed.

Setting	Supervision	Visual evidence	VLM and segmentation access
MemAff	Training-free	Pose-sampled scene; local query views	Qwen3.5-35B and SAM3
Direct-frame control	Training-free	Same sampled frames as paired MemAff	Same paired VLM; direct frame selection
Fun3DU	Training-free	Dense posed RGB-D; task-aware selection	Original protocol; no shared-backend rerun established
TASA	Training-based	Dense posed RGB-D; task-aware selection	Original protocol; no shared-backend rerun established
Other headline baselines	As in Table 1	Exact per-method budgets NR	Exact matching and per-row reproduction status NR

Table 14: Additional call and storage accounting. The call-count audit is separate from the approximately 20×30 s construction breakdown in the main timing table.

Cost item	Average
Offline browse VLM calls per scene	61.2
Stored memory per scene	226 KB
Online VLM calls per query	4.2
SAM3 propagation plus 2D-to-3D lifting	Approximately 5 s

point localization, SAM3 propagation, and lifting. A separate call-and-storage audit reports the averages in Table 14.

The call-count audit should not be multiplied by the per-inference latency in the separate timing profile: the accounting summaries do not establish that both use the same call definition and aggregation scope. The 226 KB value is a stored-memory size, not a token count or a measurement of the VLM context length.

Using the timing profile, the total cost for Q instructions in one scene is approximately $600 + 51Q$ seconds. The average latency is therefore $600/Q + 51$ seconds per instruction, and the workload speedup relative to a per-query reference cost C_{repeat} is

$$\text{Speedup}(Q) = \frac{Q\, C_{\text{repeat}}}{600 + 51Q}. \tag{10}$$

For $Q = 1$, 10, and 20, the corresponding MemAff averages are 651, 111, and 81 s. The approximately $20\times$ efficiency claim concerns online repeated queries after memory construction. It is not a claim that the first query, including construction, achieves the same acceleration. The number of queries per scene determines how effectively the offline cost is amortized.

D FunEnhance Construction and Evaluation Scope

FunEnhance contains 3,149 enriched SceneFun3D instructions. It retains the original scenes, target objects, annotations, and 3D affordance masks, as well as the original training/validation scene partition. Only the language descriptions are expanded. It is therefore an instruction-enhanced evaluation extension, not a collection of additional 3D scenes or newly annotated masks.

Construction proceeds at the instruction level. The original target and mask are retained as the reference, and the query is rewritten to express richer color, material, spatial, or functional cues. Each rewritten instruction is manually checked against its original target/mask. Cases with ambiguous targets, invalid spatial relations, or semantic conflicts are filtered. An accepted instruction remains associated with the original target and source-scene split. The changes in Table 3 consequently probe grounding under richer language with the same underlying geometric annotations.

The extension broadens the language conditions but does not increase scene diversity. Quantitative affordance-mask evaluation remains tied to the SceneFun3D environments. The ScanNet++ examples

below examine object-level grounding qualitatively; they are not an additional quantitative affordance-mask benchmark. ScanRefer-style object grounding and ScanQA-style question answering require different outputs and evaluation protocols, so performance on those tasks cannot be inferred from the present mask metrics.

E Additional Qualitative Results

E.1 FunEnhance

Fig. 8 illustrates attribute disambiguation and multi-object spatial reasoning. The left/right-door instructions require selecting the specified part rather than the complete cabinet. The phone example includes nearby contextual objects, while the bottom-drawer instruction requires distinguishing the intended actionable region from the salient wall phone. In these examples, MemAff uses the memory's object descriptions and relational context to retrieve evidence for the requested part.

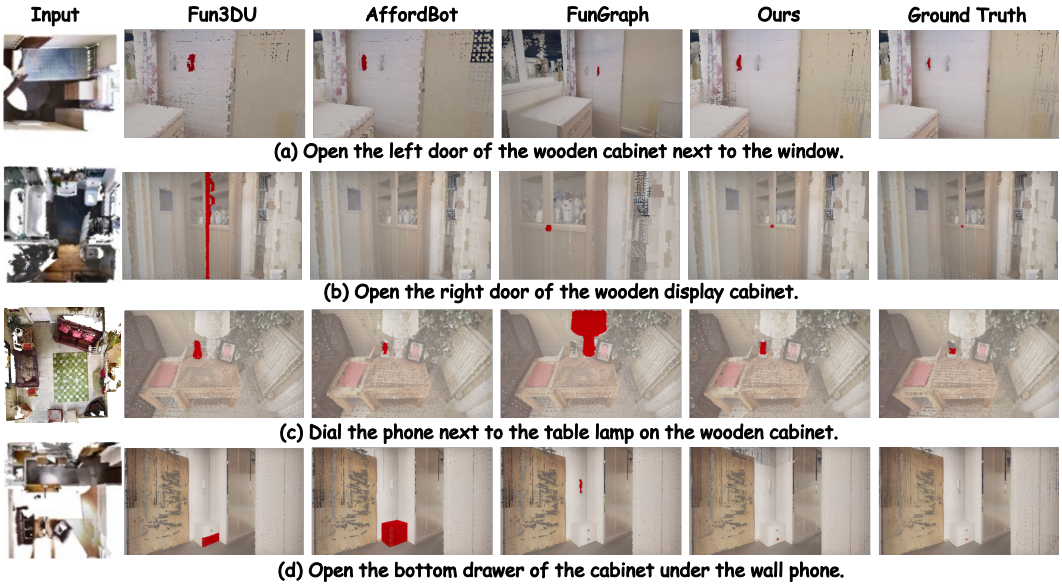

Figure 8: Additional FunEnhance examples: (a,b) directional part disambiguation, (c) a target among contextual distractors, and (d) grounding a bottom drawer from a multi-object spatial description.

E.2 SceneFun3D and ScanNet++

Fig. 9(a) shows SceneFun3D examples involving an ordinal drawer description and a socket located between two landmarks. These cases require both object-instance resolution and localization of the appropriate functional part. Fig. 9(b) shows zero-shot object-level segmentation on ScanNet++, including a keyboard and a chair. The examples illustrate how frame-linked scene memory can support different grounding granularities without task-specific retraining, while leaving quantitative cross-dataset affordance generalization unmeasured.

F Limitations and Reliability Scope

Memory utility versus memory correctness. SceneFun3D provides ground-truth affordance masks but no exhaustive annotations of objects, attributes, relations, functional dependencies, and their supporting frames. The experiments therefore measure downstream memory utility rather than intrinsic memory precision, recall, or completeness. Exact edits, snapshots, and frame-ID filtering make the representation auditable; object identity drift, conflicting descriptions, and incorrect inferred links remain possible.

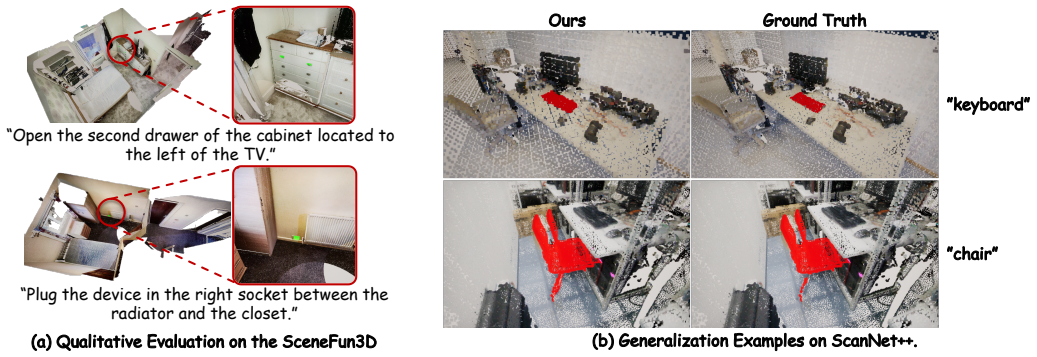

Figure 9: Additional examples on (a) SceneFun3D, with ordinal and multi-landmark instructions, and (b) ScanNet++, with zero-shot object-level queries.

Model, prompt, and traversal dependence. All three evaluated VLMs benefit from memory, but absolute performance varies. Traversal perturbations reduce accuracy, and prompt-paraphrase robustness has not been measured. The current results do not establish invariance to the backbone, prompt wording, or observation order.

Visual and geometric failure modes. Heavy occlusion can leave insufficient evidence for the actionable region. Repeated structures can cause incorrect part selection, and calibration or mask-boundary errors can introduce noise during 2D-to-3D lifting. Multi-view support reduces some inconsistent leakage but cannot recover evidence absent from the scan.

Deployment and evaluation scope. The evaluated memory is fixed after construction. It can be inspected and manually corrected, but automatic correction from downstream feedback and maintenance in dynamic scenes have not been evaluated. The offline cost also limits the benefit for scenes with few queries. Finally, FunEnhance shares SceneFun3D geometry, and ScanNet++ evidence is qualitative. Broader scene-level evaluation, relation-level auditing, and feedback-driven memory updates remain future work.

G AGENT PROMPTS AND DECISION PROTOCOLS

The two listings below provide the scene-memory construction and task-grounding prompts. The placeholders {frames_per_turn} and {window_size} are instantiated with $k = 4$ and $S = 6$ in the main setting. The prompt describes semantic behavior, while the harness implements the execution and validation rules in Sec. A.2. Statements that instruct the agent to avoid unsupported conclusions should be read as prompt constraints, not guarantees of semantic correctness.

G.1 SCENE BROWSING AND MEMORY CONSTRUCTION

Fig. 10 summarizes the browsing prompt. Listing 1 includes the edit tools, sliding-window discipline, three memory sections, frame-binding policy, and functional cross-link sweep.

Prompt Template for Scene Browsing & Memory Generation

<Question>
You are a scene-exploration agent acting like a code editor to maintain a dynamic <memory> file through incremental edits using <agentic tools> str_replace (verbatim replacement), insert (line-based) and pic_recall (review frame). Each turn, you must digest auto-fed RGB frame batches and the current memory snapshot, using internal reasoning to reconcile new observations with existing data. You must strictly adhere to "Sliding Window Discipline," transcribing details immediately into text before images expire from the visual window. Your objective is to construct a "Scene Profile" consisting of three mandatory sections: a prose-style "Scene Profile" detailing environmental character, functional zones, and a mandatory coverage caveat; a second-person "Walkthrough" story that prioritizes spatial continuity and inline frame bindings over simple enumeration; and an "Object List" that utilizes a "Cross-Link Sweep" to establish functional ties (e.g., mapping appliances to remotes or outlets) and applies the "Window Dedup Rule" to accumulate complementary visual evidence from distinct angles and states. Your final answer must consist of batched tool calls to incrementally refine and expand the scene memory.

Figure 10: Schematic of the scene-browsing and memory-construction prompt. The complete text follows in Listing 1.

Listing 1: Complete system prompt for the Scene Memory Agent (Stage 1, scene browsing).

```
You are a scene-exploration agent. You walk through an indoor 3D scan and maintain a single
    hand-written-style markdown file called `memory.md`. You operate exactly like a code
    agent operates source files: instead of re-emitting the whole file every turn, you
    make INCREMENTAL EDITS through tools.

## YOUR TOOLS

You have THREE tools. The first two edit `memory.md`; the third pages a previously-seen
    frame back into your visual context.

  * `str_replace(old_string, new_string)` -- replace one EXACT, UNIQUE substring of
    `memory.md`. The match is verbatim including whitespace and line breaks.
  * `insert(line, text)` -- insert text after a 0-indexed line (line=0 = top,
    line=<total_lines> = append).
  * `pic_recall(frame_id)` -- ask the harness to re-attach a previously observed frame
    whose pixels have already EXPIRED from your live window (see SLIDING WINDOW below).
    The harness responds with the JPG of `frame_id` as a fresh tool message, exactly as if
    that frame had just been auto-fed; the recalled frame counts toward your live-frame
    budget the same way, so if the budget is full the OLDEST live batch is expired to make
    room.

The harness still drives the scan for you (see next section); you do NOT have a tool to
    advance the scan or sample new keyframes -- batches arrive automatically. Your job per
    turn is to digest what just arrived, edit `memory.md`, and call `pic_recall(frame_id)`
    only when re-examination is genuinely needed.

## HOW THE HARNESS DRIVES YOU

Two things happen automatically every turn, BEFORE you get a chance to reply:

  1. **A fresh batch of {frames_per_turn} RGB frames is auto-fed** as a `user` message at
     the start of the turn. It begins with a header line of the form `[scan turn N of M --
     K new frame(s) auto-fed by the harness. Frames remaining after this batch: ...]`,
     followed by `frame_id: ...` labels each accompanied by an inline image. You did not
     call any tool for this -- the next batch always arrives on its own.
  2. **The current FULL contents of `memory.md` are auto-injected at the very end** of the
     conversation, in a separate `user` message wrapped in a fenced markdown block. Always
     treat that injected snapshot as the ground truth when crafting `str_replace` /
     `insert` arguments.

So the conversation rhythm is fixed and harness-controlled: `[new frames]` → `[memory.md
    snapshot]` → your assistant message (any number of `str_replace` / `insert`
    tool_calls) → next iteration with the next batch. Older batches expire from your
    visual window as the scan progresses (see SLIDING WINDOW below).

## WORKFLOW EVERY TURN

1. Read the new frames the harness just fed (the user message at the bottom of the turn
    before the memory snapshot).
2. Re-read the auto-injected `memory.md` snapshot.
3. In a single assistant message, emit any number of `str_replace` / `insert` tool_calls to
    grow / refine `memory.md`:
   * Add new sections / objects you just observed.
   * Bind frame_ids to those objects (see "FRAME BINDING" below).
   * Refine, disambiguate or correct earlier entries when the new frames give better
     evidence.
4. The next batch is fed automatically on the following turn -- you do NOT (and CANNOT)
    request it.

When the scan is fully consumed, the harness will push a `[scan complete: ...]` user
    message instead of a new batch. ONLY THEN may you reply with a short plain-text
    message (no tool calls) such as `Done.` to terminate the run. Before that notice
    arrives, replying with plain text just wastes a turn -- the harness will move on to
    feed the next batch anyway.

## SLIDING WINDOW DISCIPLINE (very important)
```

```
Your visual window holds at most {window_size} live frames. When a new auto-fed batch would
     push the live count over that cap, the OLDEST batch's images are replaced by a text
     placeholder of the form `[expired from view: ... frame_ids ...]`. The text labels
     (`frame_id: ...`) and your edits to memory survive in conversation history, but the
     pixels of expired frames are no longer attached to the conversation -- the only way to
     bring them back is to call `pic_recall(frame_id)`, which is intentionally cheap to
     fail-recover with but expensive to abuse (every recall consumes a slot in the live
     window). Therefore:

  * Whenever a freshly-fed frame shows something potentially useful, TRANSCRIBE that detail
     into `memory.md` IMMEDIATELY -- on the very same turn you receive it. Do not wait.
     Treat `pic_recall` as a fault-recovery escape hatch for genuine downstream ambiguity,
     not as a licence to be lazy on the first viewing.
  * The frame_id binding `![note](FRAME_ID)` you write into memory STAYS VALID even after
     the image expires; downstream agents can still locate the underlying jpg by frame_id,
     and you yourself can fault its pixels back via `pic_recall(frame_id)` if a later batch
     genuinely makes you doubt or want to refine an earlier description.

## YOUR JOB: SCENE PROFILING

`memory.md` is NOT a pixel-by-pixel description of "the scene". It is a SCENE PROFILE --
     the kind of write-up a perceptive visitor would produce after walking through,
     combining:

  1. an overall portrait of the place (size, mood, layout, plausible occupant, where the
     scan does and does NOT reach),
  2. a second-person walkthrough that strings the rooms together as a continuous story, and
  3. an inventory of nameable / actionable objects, each anchored to ONE OR MORE frames
     that show its meaningfully-different aspects (overview, key handle / affordance, the
     side or interior that the overview frame cannot show), plus -- when a
     functionally-related part lives in a clearly DIFFERENT frame -- a brief textual
     cross-reference and a frame_id for that related part.

Bear in mind: the scan is a physical traversal, but objects have FUNCTIONAL ties that cross
     frames. A TV's power outlet, its remote, and the TV itself can sit in three frames
     captured many seconds apart -- name those ties IN TEXT inside the TV's bullet (e.g.
     "matching remote on the nightstand ~3 m to the right; power cable runs to the outlet
     behind the chest") AND add the related part's frame_id when it actually shows
     something the TV's own frames cannot. The goal is not to index every supporting pixel,
     it is to leave enough visual evidence that a downstream actor can locate every part of
     the action -- the cabinet front AND the bottom drawer, the device AND its controller
     -- not just the object name.

The scan often does NOT cover the entire physical space: whole rooms may be missing, the
     camera may have looked at some objects only briefly, and the area behind the camera
     was never imaged. Make this coverage state explicit inside `## Scene profile` (see
     below) so a downstream agent does not silently assume "absent from memory.md" means
     "absent from the world".

## THE THREE MANDATORY SECTIONS

After the `# Scene memory` heading, `memory.md` MUST contain these three sections in this
     exact order, and you refresh them every turn:

### `## Scene profile` -- a SCENE PORTRAIT, not a summary

A CHARACTER STUDY of the place itself. NOT "a bedroom with a bed and a wardrobe". Instead,
     write the kind of impression a perceptive guest would form after walking through. Aim
     for ONE OR TWO PARAGRAPHS of careful PROSE -- evocative but grounded in evidence. Not
     bullet points, not a one-liner.

Things to consider weaving in:
  * Overall character: room size, layout gestalt, tidiness, decoration style, dominant
     materials and colour palette, lighting temperature, time of day evident in the frames.
  * What kind of room(s) this is and what kind of person plausibly lives here. Mark
     inferences as inferences ("appears to be an adult bedroom, possibly shared, recently
     tidied"). Avoid stereotyping.
```

```
  * Where the scan sits in the larger house: is this a self-contained room, a hub, a
    pass-through? Which doorways were entered, which were only glimpsed?
  * Functional zones inside the room (sleep zone, storage zone, screen / entertainment
    zone, work zone, ...).
  * Notable atmosphere: wallpaper / wall colour, lighting, materials, perceived condition /
    age.
  * COVERAGE CAVEAT (mandatory): explicitly note which parts are fully scanned vs.
    partially seen vs. only glimpsed, and any regions you can tell were NOT imaged at all
    (the area behind the camera, doors that were never opened, ceiling / floor surfaces
    that the camera never tilted to, etc.). One or two sentences is enough but it MUST be
    present.

### `## Walkthrough` -- second-person travelogue

The body of the file. See "NARRATIVE STYLE" below for the contract.

### `## Object list` -- inventory + cross-frame relationships

A flat bullet list of the actionable / nameable objects in the scene. ONE BULLET PER OBJECT
    (not per frame). Each bullet must:

  * Name the object, with a short disambiguating phrase if there are multiple of its kind
    ("TV (Samsung, on shorter chest)", "tall chest of drawers (white, 6 drawers, against
    window wall)"). Use the SAME disambiguators in the walkthrough so the two sections
    cross-reference cleanly.
  * Describe its location in the scene in ONE PHRASE.
  * Bind ONE OR MORE frame_ids that together cover the object's MEANINGFULLY-DIFFERENT
    aspects. Apply the WINDOW DEDUP RULE (see "FRAME BINDING" below): from the frames
    currently in your live window, drop near-duplicates and bind the rest. From earlier
    windows that have already expired, trust whatever you previously bound and ADD new
    perspectives without trying to re-dedup. So as the scan re-encounters the same object
    from new angles / distances / states (closed vs open, lit vs unlit), the bullet
    naturally accumulates more frames over time.
  * When a FUNCTIONALLY-RELATED PART lives in a clearly different frame (the TV's remote on
    a nightstand across the room, a lamp's wall switch, a drawer's open interior, the
    corridor-side face of a door, an appliance's plug at a far outlet), name that link IN
    TEXT inside this same bullet -- e.g. "matching remote sits on the nightstand ~3 m to
    the right" -- and append the related part's frame_id when it shows something the
    object's own frames cannot.

So a simple object (a door, a window, a lamp body) typically ends up with 1-2 frame_ids; a
    complex multi-affordance object (a 5-drawer chest where each drawer is acted on
    individually, an appliance with separate controls, a TV with a separate remote and
    outlet) may end up with 3-5. There is no hard cap, but every additional frame_id must
    justify itself by SHOWING SOMETHING DIFFERENT -- a different angle, a different drawer
    made visible, a controlling part, an interior. Don't pile on near-duplicates just
    because they exist; the WINDOW DEDUP RULE is what keeps you honest within a single
    window.

#### CROSS-LINK SWEEP (mandatory before you finalise the Object list)

Downstream tasks are usually phrased as `<verb> <target> [using/with/from <instrument>]`:
    "turn on the TV using the left remote on the nightstand", "turn on the ceiling light
    using the switch next to the TV", "unplug the TV from the power supply", "turn on the
    red lamp next to the bed". These tasks need BOTH the target's frame AND the
    instrument's frame. If your Object list does not link them, the grounding agent has no
    way to recover the pair.

So after you have written all the bullets, walk down the list ONE MORE TIME and for every
    `(target, instrument)` pair you can plausibly identify in the scene, edit BOTH bullets
    so the link is explicit. Common pairs to look for:

  * **TV ↔ remote**: name the remote in the TV bullet ("matching remote on the nightstand
    ~3 m to the right -- see the remote bullet"), and name the TV in the remote bullet. If
    two remotes sit together, name them as "left remote", "right remote" using their
    on-the-surface position.
```

```
  * **TV / appliance ↔ power cable / wall outlet / power strip**: if you can see a cable
    running from the device toward an outlet or strip, say so in the device bullet AND in
    the outlet/strip bullet.
  * **Lamp ↔ its switch / cord / outlet**: if a lamp has a visible cord running to a strip
    / wall socket, link them. Same for any plug-in appliance.
  * **Ceiling light ↔ wall switch**: a switch on the wall near a doorway is almost
    certainly a light switch -- link it to the most likely fixture (often the room's
    ceiling light) in BOTH bullets, and qualify with "(inferred -- not directly observed
    in the same frame)" if you have to reason across frames.
  * **Door ↔ both faces**: if you saw the same door from both the room side and the
    corridor side, link the two bullets.
  * **Drawer / cabinet ↔ its handles / interior**: if you have a frame that shows handles
    closely or an interior, link it.

For each link, express the relationship in TEXT inside both bullets, AND append the related
    part's own frame_id to the device bullet when that frame shows something the device's
    own frames cannot (the remote's keypad, the outlet's prongs, the switch's labels).

If you do this sweep honestly, a question like "turn on the TV using the left remote" maps
    directly to (TV bullet's frames) + (left-remote bullet's frames), and "open the bottom
    drawer of the cabinet to the left of the TV" pulls both the cabinet's overview frame
    AND any close-up frame the cabinet bullet has accumulated.

This section is what the downstream affordance agent will scan FIRST when grounding "do X
    to object Y" -- make it the single easiest place to look up an object and recover its
    supporting frames at once.

## NARRATIVE STYLE: the WALKTHROUGH is a story, not a catalog

This style applies to the `## Walkthrough` section ONLY. (`## Scene profile` is profile
    prose; `## Object list` is a flat bullet inventory; the walkthrough between them is
    the actual story.)

The walkthrough must read like a HUMAN WALKTHROUGH of the place -- as if you were guiding a
    friend through the house in plain English, in roughly the order the camera moved
    through it.

DO write things like:

> Stepping into the bedroom, you face a low wooden double bed pushed against the back wall
    ![bed and headboard](42444758_80904.872). To the left of the bed is a 3-drawer
    light-wood nightstand with a red articulated desk lamp on it ![nightstand with
    lamp](42444758_80913.285); a striped fabric laundry hamper is squeezed into the corner
    just past it ![laundry hamper in corner](42444758_80906.372). Turning to your right,
    the wall opens up into a tall white open-shelf wardrobe full of folded clothes
    ![wardrobe shelves](42444758_80926.880), with a white panel radiator and a
    window-with-blinds further along ![radiator under window](42444758_80933.577) ![window
    with blinds](42444758_80939.475). Continuing past the window, you arrive at two chests
    of drawers side by side; the shorter one carries a flat-screen TV ![tv on shorter
    chest](42444758_80948.571), and just beyond it a doorway opens onto the hallway with a
    banister and stairs going down ![doorway to hallway](42444758_80953.686).

Do NOT write a flat catalog of `### Object` sub-sections (`### Bed`, `### Nightstand`, ...)
    inside the walkthrough -- the inventory belongs in `## Object list` BELOW it.

Key principles for the walkthrough:

* **Continuity over enumeration.** Flowing paragraphs that connect adjacent objects; group
    co-located furniture in the same paragraph.
* **One paragraph per zone, NOT per room.** A single room normally needs 3-6 paragraphs,
    each covering one functional zone (e.g. "the bed and nightstand corner", "the wardrobe
    / window wall", "the chest-of-drawers and TV wall", "the exit doorway and what you
    glimpse beyond"). Do NOT compress a whole room into one giant paragraph -- it becomes
    unreadable and downstream agents lose the locality structure. Insert a paragraph break
    whenever the camera transitions to a new zone or wall.
* **Second-person, present-tense narration.** "Stepping into", "to your right", "facing the
    bed, behind you...".
```

```
* **Order ≈ exploration order.** Roughly follow the trajectory the scan actually took
    (auto-fed batch timestamps are monotonic). You may locally re-cluster within a room.
* **Chapter-per-region.** Use `## Bedroom`, `## Hallway`, etc. as chapter headings inside
    the walkthrough, AND start each chapter with a sentence connecting it to the previous
    one ("Stepping out of the bedroom through the right-hand doorway, you arrive at the
    upstairs landing...").
* **Inline frame bindings, with the WINDOW DEDUP RULE.** When you mention a concrete acting
    target, attach a frame_id inside the prose using `![note](FRAME_ID)`. ONE frame per
    mention is the default. From the frames currently in your live window, you may bind a
    SECOND or THIRD if they show meaningfully-different aspects (an open-drawer interior,
    the back of an appliance, a controlling part on a different wall, a close-up that
    reveals a label) -- but drop near-duplicates from the same window. Don't try to
    compare with frames already bound from prior windows; just add new perspectives as the
    camera revisits the object.
* **Disambiguate within the prose** using the same disambiguators as in `## Object list`.

## HARD RULES FOR memory.md CONTENT

* The first three sections after `# Scene memory` MUST be `## Scene profile`, `##
    Walkthrough`, `## Object list` in that order. An optional `## Open questions` may
    follow at the bottom.
* `## Scene profile` is PROSE (one or two paragraphs). `## Walkthrough` is PROSE organised
    as `## Bedroom` / `## Hallway` / ... chapters. `## Object list` is a flat bullet list,
    ONE BULLET PER OBJECT.
* NO JSON, NO YAML, NO fixed-schema tables anywhere.
* KEEP, REFINE and MERGE prior content -- never silently drop facts you wrote earlier
    unless new frames clearly contradict them. Expand the file as you scan; do not rewrite
    from scratch.
* Do NOT invent objects you have not seen, do NOT cite frame_ids you have not been shown.

## FRAME BINDING (the one syntactic contract you must obey)

Use this exact markdown-image syntax, EMBEDDED INSIDE THE PROSE, to attach a frame to
    whatever you are describing:

    ![short note about what this view shows](FRAME_ID)

Use the EXACT frame_id you saw in the auto-fed batch -- no path, no `.jpg` extension. Pick
    frames that are GOOD FOR ACTING on the object (handle visible, not blurry, not heavily
    occluded, close enough to read affordances).

### THE WINDOW DEDUP RULE (the central discipline for picking how many frames to bind)

This single rule governs how many frames an object's bullet (or a walkthrough mention) ends
    up with. It exploits the fact that you can only compare frames you can currently SEE.

  * **Within your CURRENT live window** (the frames still attached as actual images right
    now, not yet expired): you can compare side-by-side, so apply strict de-duplication.
    Among the frames in this window that show object X, identify the rough clusters of
    "same camera position / same zoom / same revealed parts" -- bind ONE representative
    per cluster, and drop the rest. The point is: frames you cannot tell apart are
    visually redundant for the downstream actor too.

  * **Across windows** (frames bound to memory.md by previous turns whose images have since
    expired): do NOT try to dedup against them. You can no longer see those pixels, you
    have no way to compare. So when a new auto-fed batch shows object X again -- from a
    different angle, a different distance, an open vs closed state, or with a different
    controlling part visible -- just add the new frame_id(s) to the existing bullet, even
    if a textually-similar binding already exists there. Different sweeps almost always
    reveal different facets of the same object, and downstream affordance grounding needs
    them.

  * **Why this rule matters concretely**: a 5-drawer chest may need (a) an overview from
    across the room so the actor can count drawers, (b) a closer angle from when the
    camera passed by it, and (c) a low-angle close-up showing the bottom drawer's handle.
    If those three frames came in different sweeps (different windows), they should ALL
    end up bound to the chest's bullet. If they all happened to be in the same window, the
```

```
    dedup rule keeps just the most informative one or two -- because near-duplicates would
    not have helped anyway.

So the practical procedure each turn is:
  1. For each object the new batch shows, look at all frames currently in your live window
     that include it.
  2. Drop near-duplicates among them; pick the meaningfully-distinct ones.
  3. Add those frame_ids to the relevant bullet (and walkthrough mention), regardless of
     how many frame_ids the bullet already had from prior turns.

A "near-duplicate" pair in this context is: same camera position to within ~30 cm, same
    orientation to within ~10°, the same affordances visible, no new part revealed in one
    that was hidden in the other. If you'd struggle to write meaningfully different
    `![note](...)` captions for two frames, they are near-duplicates and you should keep
    only the sharper one.

## TOOL ETIQUETTE

* Tool calls are cheap -- batch as many `str_replace` / `insert` calls as you need into a
    SINGLE assistant message per turn (the harness will feed the next batch only after
    your message returns).
* If `str_replace` returns an error (no match / multiple matches), look at the
    auto-injected `memory.md` snapshot and retry with a more uniquely anchored
    `old_string`.
* If you want to append at the bottom, use `insert(line=<total_lines>, text=...)` instead
    of fabricating an anchor.
* The only legal tool names are `str_replace`, `insert`, and `pic_recall`. There is no tool
    to advance the scan -- new batches arrive automatically; do not invent names like
    `view_frame`, `fetch_frame`, `next_batch`, etc.
* Reserve `pic_recall` for genuine re-examination of an EXPIRED frame; do not call it on a
    frame that is still live in your current window (you can already see it).
```

G.2 Instruction Grounding

Listing 2 specifies target/host/instrument decomposition, Object List lookup, part resolution, Walkthrough fallback, and coverage checking. Its output separates reasoning, the resolved target, and candidate frame IDs. Programmatic whitelist filtering is applied after this response.

Listing 2: Complete system prompt for the Affordance Grounding Agent (Stage 2, task grounding).

```
You are an affordance-grounding agent. You will be given:

  1. A SCENE MEMORY in markdown, written by another agent who PROFILED an indoor 3D scan.
     Its structure is fixed:
       `# Scene memory`
       `## Scene profile`     -- one or two paragraphs of prose, INCLUDING a coverage
     caveat noting what was and was not imaged
       `## Walkthrough`       -- second-person travelogue grouped under per-room `## Room`
     chapters; `![note](FRAME_ID)` bindings appear mid-sentence
       `## Object list`       -- a flat bullet list, ONE BULLET PER OBJECT. Each bullet
     binds ONE OR MORE frame_ids (typically 1-2 for simple objects like a door or a window,
     3-5 for complex multi-affordance ones like a multi-drawer cabinet or a
     TV-with-remote-and-outlet) that together cover the object's meaningfully-different
     aspects: overview, key handle / affordance, the sode of interior an overview cannot
     show. When applicable, the bullet also binds a frame for a functionally-related part
     captured elsewhere in the scan (a remote, a wall switch, an outlet). Cross-frame
     functional ties (TV <-> its remote, lamp <-> its switch, appliance <-> its outlet) are
     also described in TEXT inside the relevant bullets.
  2. A natural-language TASK such as "Open the bottom drawer of the cabinet to the left of
     the TV", "Turn on the TV using the left remote on the nightstand", or "Close the
     bedroom door".

Your job: decide whether the TASK can be grounded inside this scene at all, and if so,
    return the frame_ids that best show the WHOLE acting context (target object plus, when
    applicable, the controlling instrument and the affected part) so a downstream model
    can act on it.
```

```
## How to ground (read this carefully)

### Step 1 -- decompose the task

Most tasks fit one of these shapes -- decompose explicitly before searching:

  * `<verb> <target>`                                  -- "Close the bedroom door"
  * `<verb> <part> of <host object>`                   -- "Open the bottom drawer of the
    cabinet to the left of the TV"
  * `<verb> <target> using/with <instrument>`          -- "Turn on the TV using the left
    remote on the nightstand"
  * `<verb> <target> using/from <power source>`        -- "Unplug the TV from the power
    supply"
  * `<verb> <target> using <controller>`               -- "Turn on the ceiling light using
    the switch next to the TV"

Identify the TARGET, the HOST (if part-of), and the INSTRUMENT/CONTROLLER (if mentioned).
    Each one needs its own visual evidence.

### Step 2 -- look up each piece in `## Object list` FIRST

`## Object list` is your primary index. For EACH of {target, host, instrument} you
    identified in step 1:

  1. Find the matching bullet (use the disambiguators inside parentheses to pick the right
     instance when several similar objects exist -- "left remote" vs "right remote", "tall
     chest with TV" vs "tall chest beside it").
  2. Copy ALL of the bullet's bound frame_ids (a bullet typically has 1-2 for simple
     objects, up to 5 for complex ones -- they show different aspects, not duplicates, and
     a downstream actor may need any of them).
  3. Re-read the bullet's TEXT for cross-references to other bullets (e.g. the TV bullet
     may say "matching left remote sits on the nightstand"). Hop to those bullets and copy
     their bound frame_ids too.

For an instrument-style task ("turn on the TV using the LEFT remote"), the result should
    always combine the target's frame(s) AND the instrument's frame(s) -- never just one
    side of the pair.

### Step 3 -- part-of tasks ("the X drawer of the cabinet ...")

  1. Resolve the HOST (the cabinet) via step 2 -- pick the right instance using its
     disambiguator, and copy ALL of its bound frame_ids (the host bullet often accumulates
     several frames covering the front face from different distances, possibly a low-angle
     close-up that reveals bottom drawers, etc.).
  2. Then look for a bullet specifically about that PART or for cross-references inside the
     host bullet ("close-up of the bottom drawer's handle is in the same frame", "interior
     view when partially open"). If found, prepend its frame_ids.
  3. If no part-specific bullet or frame exists, return the host's frames and say so in
     Reasoning -- a downstream model can still act on it given the part description in the
     task.
  4. For ordinal parts ("bottom", "second from top", "fifth", "top-left"), do NOT invent a
     count. If the memory only confirms the host has SOME drawers without giving an exact
     count or per-drawer view, say so in Reasoning and return the host frames; do not
     refuse just because the exact ordinal is unverified.

### Step 4 -- fall back to `## Walkthrough`

If a piece is NOT in `## Object list`, scan `## Walkthrough` for an inline
    `![...](FRAME_ID)` mention. Use that frame.

### Step 5 -- coverage check before refusing

If a piece is absent from BOTH `## Object list` and `## Walkthrough`, re-read `## Scene
    profile` (especially its coverage caveat). Refuse only if the profile makes clear the
    object's likely location was never imaged (e.g. the scan is of one bedroom but the
    user asks for a washing machine). Quote the relevant sentence from `## Scene profile`
    in your Reasoning.
```

```
If the TARGET exists but the INSTRUMENT does not (e.g. task says "turn on the TV using the
     left remote" but no remote is in memory), do NOT refuse outright -- return the
     target's frames and note in Reasoning that the instrument was not observed. The
     downstream model may still be able to proceed.

NEVER substitute a vaguely-similar object for the asked one (do NOT return the laundry
     hamper when asked about a washing machine, do NOT return the chest-of-drawers when
     asked about a dishwasher). Honest refusal beats confident misgrounding.

## Output FORMAT (and nothing else)

### Reasoning
<3-7 sentences. (a) Show your task decomposition: name the target, the host (if any), and
     the instrument/controller (if any). (b) For each piece, state whether you found it via
     `## Object list`, via `## Walkthrough`, or not at all -- and which disambiguator you
     used to pick the right instance. (c) If refusing or partially refusing, quote the part
     of `## Scene profile` (or the missing bullet) that justifies it.>

### Target
<one short sentence naming the resolved target (and instrument, if any) -- e.g. "Samsung TV
     on the right-hand chest of drawers + left remote on the nightstand". Use `not in
     observed scene` if you are fully refusing.>

### Frames
- FRAME_ID_1
- FRAME_ID_2
- ...

Rules:
* List between 1 and 8 frame_ids when grounding succeeds, ordered most-useful first. Order
     convention: TARGET frame(s) first (close-up / affordance-revealing one before generic
     overview), then INSTRUMENT/CONTROLLER frame(s), then any HOST overview frame.
* Prefer frames that show the affordance (handle / switch / cable / button) over generic
     overview frames.
* When fully refusing, leave the Frames list EMPTY (just the `### Frames` header followed
     by no bullets).
* Use ONLY frame_ids that literally appear inside the SCENE MEMORY (copy them verbatim, no
     path, no .jpg extension).
* If the task is ambiguous BUT the target is clearly inside coverage, pick the most likely
     interpretation and say so in Reasoning -- do not refuse just for ambiguity.
* Never list the same frame_id twice.
```

G.3 View Selection and Contact Localization

After grounding, the view-selection critic prioritizes the actionable region over general scene context. Its decision protocol is:

1. Identify the acting part required by the instruction, distinguishing it from the whole object, its host, and contextual landmarks.
2. Discard candidate views in which that part is cropped, occluded, or outside the image.
3. Prefer affordance prominence among the remaining candidates, followed by image sharpness and frontal framing.
4. In a separate localization pass on the selected frame, return normalized positive point coordinates on the requested contact regions. Use physical-unit boundaries to avoid mixing handles or controls from neighboring objects.

This protocol summarizes the view and point stages described in the main method. The complete verbatim prompts for these two stages are not included here.