PORTER: EDGE–CLOUD RESIDENCY FOR PERSISTENT 3D SCENE GRAPH MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent task-driven and just-in-time 3D Scene Graph (3DSG) methods reduce per-task representations by constructing or activating only task-relevant information. Yet sparse per-task working sets do not bound onboard memory usage over a robot's lifetime: as tasks change, payloads accumulated for earlier tasks may become irrelevant to the current task but can be useful again in future tasks. Over repeated task switches and expanding environments, retaining such reusable payloads causes local memory to grow, whereas discarding them entirely can lead to costly repeated construction of the same payloads later. We introduce **PORTER**, which decouples persistence from residency: lightweight anchors remain in the limited memory of the edge robot while heavy object payloads migrate between the edge and the cloud. Relevance alone is insufficient for deciding residency because multiple relevant payloads may provide redundant information. We therefore decompose each task into functional requirements and introduce *Irreplaceable Support Erasure (ISE)*, which measures the loss in requirement coverage caused by offloading. ISE discounts replaceable support and penalizes losses more strongly when the remaining coverage of a requirement is weak. PORTER constructs a budget-aware local working set by repeatedly offloading the payload with the smallest marginal ISE per byte. Experiments on JITOMA-Bench evaluate PORTER across four 3DSG builders. Under progressive compression, pooled relative mR@3 remains at **100%** through **91%** payload-byte offloading.

1 INTRODUCTION

Long-lived embodied agents need memory that outlasts any individual task: objects perceived today may become useful again minutes, days, or tasks later. Yet retaining raw observations indefinitely is neither an efficient nor a useful way to reason about a growing environment. 3D Scene Graphs (3DSGs) (Gay et al., 2018; Armeni et al., 2019) offer an appealing alternative by abstracting accumulated observations into persistent, geometrically grounded objects and relations, yielding a compact and queryable representation for downstream navigation, manipulation, and scene reasoning (Honerkamp et al., 2024; Yin et al., 2024; Werby et al., 2024; Yan et al., 2025; Chen et al., 2026; Xia et al., 2026). This abstraction makes 3DSGs a natural substrate for long-term robot memory. But making the representation persistent raises a different systems question: *can the robot retain what it has learned without keeping all of it locally resident?*

Recent 3DSGs have become increasingly capable of serving reusable scene memory. Open-vocabulary representations broaden what can be queried (Jatavallabhula et al., 2023; Gu et al., 2024; Koch et al., 2024; Werby et al., 2024; Yamazaki et al., 2024; Linok et al., 2025; Zhang et al., 2025; Chang et al., 2026b), while task-driven and just-in-time methods avoid processing the full scene for every goal by constructing, abstracting, or activating only task-relevant information (Agia et al., 2022; Maggio et al., 2024; Maggio & Carlone, 2025; Maggio et al., 2026; Chang et al., 2026a). These advances improve what the memory can express and how efficiently it serves the *current* task, but they do not bound what accumulates over the robot's *lifetime*. As tasks change, payloads useful to earlier tasks may no longer deserve scarce local memory while remaining valuable for future reuse. Over repeated task switches and expanding environments, retaining such reusable payloads causes onboard memory to grow, whereas deleting them turns efficiency into forgetting. This exposes a missing distinction between *persistence* and *residency*. An object need not disappear from the robot's memory merely because its representation is no longer worth occupying the edge.

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.

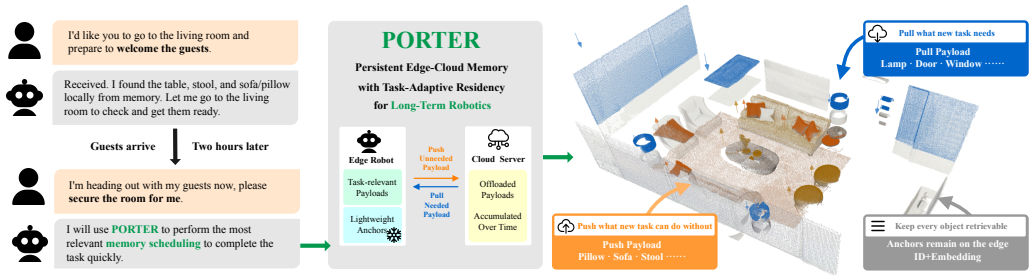

Figure 1: **Overview of PORTER.** PORTER keeps lightweight anchors on the robot while task-dependent payloads move between the edge and cloud, allowing scene knowledge to persist across tasks without requiring all payloads to remain locally resident.

We therefore formulate *task-switch-aware edge–cloud residency*, where scene knowledge persists across tasks while its physical placement adapts to the task. **PORTER** realizes this idea by factorizing each 3DSG object into a persistent lightweight anchor and a migratable heavy payload. The anchor preserves identity and retrieval data locally, whereas the payload contains the storage-intensive state supplied by the upstream 3DSG builder—for example, dense object point clouds, historical image crops, or other builder-specific observations. Because only the payload moves, PORTER can release local memory without deleting the object, while remaining agnostic to how that object was constructed. Task switching then becomes a placement problem rather than a forgetting problem: which edge-resident payloads should be pushed to the cloud, and which cloud-resident payloads should be pulled back to form the working set for the new task?

Making this decision requires more than task relevance: what matters is how much task support would be lost if a payload left *now*. A chair and a sofa may both satisfy the need for seating; either may incur little modeled support loss while the other remains, but the survivor becomes increasingly valuable once its substitute is gone. PORTER models this state-dependent redundancy by decomposing the task into functional requirements and aggregating alternatives with a Noisy-OR model. We formalize the resulting counterfactual loss as *Irreplaceable Support Erasure (ISE)*, which accounts for both substitute coverage and residual fragility. PORTER then constructs a budget-aware working set by repeatedly offloading the payload with the smallest *marginal ISE per byte*, recomputing its value as the resident set changes. Across four heterogeneous 3DSG builders on JITOMA-Bench (Chang et al., 2026a), PORTER reduces local residency while preserving task capability, with no mR@3 degradation through 91% payload-byte offloading. On a real robot, it further reduces average local residency by 59.1% and mapping TPF by 70.3% without reducing mR@3.

2 Related Work

2.1 3D Scene Graphs for Embodied Intelligence

Scene graphs abstract visual environments into entities and relations, and their extension to 3D yields geometrically grounded, object-centric scene representations that are more structured and compact than dense sensory observations (Johnson et al., 2015; Lu et al., 2016; Krishna et al., 2017; Armeni et al., 2019). Early 3DSG systems further enabled online and hierarchical mapping for robotic operation (Rosinol et al., 2021; Wu et al., 2021; Hughes et al., 2022), while recent vision–language models have extended 3DSGs beyond predefined semantic categories to open-vocabulary representations (Jatavallabhula et al., 2023; Gu et al., 2024; Koch et al., 2024; Werby et al., 2024; Yamazaki et al., 2024; Linok et al., 2025; Zhang et al., 2025; Chang et al., 2026b). These developments have made 3DSGs a versatile interface for embodied tasks including navigation, object search, manipulation, and high-level planning (Gadre et al., 2022; Agia et al., 2022; Shah et al., 2023; Rana et al., 2023; Yin et al., 2024; Yan et al., 2025; Chen et al., 2026; Xia et al., 2026). More broadly, their persistent object identities and relational structure allow accumulated scene knowledge to be queried and reused across interactions, making 3DSGs a natural substrate for embodied memory.

2.2 Toward Persistent 3D Scene Graph Memory

As open-vocabulary 3DSGs grow richer, maintaining all available scene information conflicts with the resource constraints of embodied platforms. Task-driven and just-in-time methods address this tension by selectively allocating representation and computation to the current objective. Taskography (Agia et al., 2022) introduced task-relevant reasoning over large 3DSGs, while Clio (Maggio et al., 2024) adapts open-set abstraction to downstream tasks. Subsequent methods further explore task-conditioned representation and granularity (Maggio & Carlone, 2025), including FOUND-IT (Maggio et al., 2026) and JITOMA (Chang et al., 2026a), which adapt granularity or activate expensive semantics as task demands evolve. Related systems treat 3DSGs as embodied memory: GraphEQA (Saxena et al., 2024) combines 3DSGs with task-relevant observations as multimodal memory, while KARMA (Wang et al., 2025) integrates them into a long- and short-term memory architecture. Yet selective abstraction and activation alone do not bound long-term memory growth: as environments and tasks evolve, reusable information continues to accumulate beyond what must remain locally active. Recent analyses (Rotondi et al., 2026) accordingly identify compact representation and memory management over months or years as open challenges for long-term 3DSGs.

3 Method

PORTER decouples logical *persistence* from physical *residency*. Each 3DSG object retains a lightweight local anchor, while its storage-intensive payload may migrate between the edge and the cloud. The key question is not how relevant a payload is, but what task support would be lost if it left the edge. We formalize this counterfactual loss as *Irreplaceable Support Erasure (ISE)*, which accounts for both substitute support and the fragility of what remains. PORTER then constructs a budget-aware working set by repeatedly offloading the payload with the smallest *marginal ISE per byte*, recomputing its value as the resident set changes.

3.1 Persistent Anchors and Migratable Payloads

Let $\mathcal{V} = \{v_1, \dots, v_N\}$ denote the object nodes produced by an upstream 3DSG. PORTER factorizes each object as $\mathcal{O}_v = (A_v, P_v)$, where A_v is a lightweight anchor that preserves identity and compact retrieval data, including its vision–language embedding for task-conditioned retrieval, and P_v contains the storage-intensive builder-specific state, such as a dense 3D point cloud, historical object crops, or other per-object observations. Let b_v denote the local memory footprint of P_v.

All anchors remain on the edge, while only payloads are subject to residency decisions. Let $S \subseteq \mathcal{V}$ denote the objects whose payloads are currently local, with memory footprint $B(S) = \sum_{v \in S} b_v$. Offloading v therefore moves only P_v to the cloud while retaining A_v locally, so the object remains identifiable and retrievable in the persistent 3DSG. Residency changes are thus reversible: a later task can restore P_v without reconstructing the object from scratch or revisiting the scene.

3.2 From Relevance to Irreplaceable Task Support

Residency should depend not only on object relevance in isolation, but on what task support would become unavailable without it. This replaceability is *requirement-dependent*: the same object may be indispensable for one functional need yet redundant for another. We represent task q as a compact set of functional requirements $\mathcal{R}_q = \{r_1, \dots, r_M\}$, with equal weights $\omega_r = 1/M$. The requirements may be supplied by a task parser or language model; PORTER itself operates only on the resulting requirement set. This lets PORTER ask not only whether an object is related to the task, but *which requirement it supports and whether that support is already provided elsewhere*.

We first estimate the potential support of object v for requirement r. Let $\hat{\mathbf{f}}_v$ and $\hat{\mathbf{g}}_r$ denote their normalized embeddings in a shared vision–language space, with $s_{vr} = \hat{\mathbf{f}}_v^{\mathsf{T}} \hat{\mathbf{g}}_r$. We define

$$a^F_{vr} = \mathbf{1}[v \in \mathcal{K}_r]\ \mathrm{clip}\left(\frac{s_{vr} - \alpha}{1 - \alpha}, 0,\ 1 - \varepsilon\right), \qquad a^A_{vr} = \eta a^F_{vr}, \tag{1}$$

where $\mathcal{K}_r$ contains the top-K candidate objects for requirement r, and $\eta \in [0, 1]$ controls how much task support remains available from the lightweight anchor after its payload is offloaded.

Thus, a_{vr}^F and a_{vr}^A represent the soft support associated with the full payload and anchor-only states, respectively. These scores are not treated as calibrated probabilities; their role is to expose how object support changes with residency. For a candidate local set S, the effective support becomes

$$e_{vr}(S) = \begin{cases} a_{vr}^F, & v \in S, \\ a_{vr}^A, & v \notin S. \end{cases} \tag{2}$$

What matters, however, is not this support alone but how much of it remains uncovered by alternatives. Under a Noisy-OR aggregation, $1 - e_{vr}(S)$ is the portion of requirement r left unsupported by object v, and their product therefore represents the support left uncovered by all objects. We define

$$c_r(S) = 1 - \prod_{v \in \mathcal{V}} \big(1 - e_{vr}(S)\big), \tag{3}$$

where $c_r(S)$ is the soft coverage of requirement r under residency set S.

This construction makes substitute redundancy explicit. When several objects support the same requirement, its uncovered residual is already small and another substitute contributes little additional coverage; as those alternatives disappear, the remaining objects become progressively more valuable. PORTER therefore does not equate relevance with value: an object's value is the support that the rest of the current working set cannot replace. The next section measures the erasure of precisely this irreplaceable support under payload offloading, turning replaceability into a residency criterion.

3.3 Irreplaceable Support Erasure

The coverage $c_r(S)$ above tells us what requirement support remains after accounting for substitutes. Residency, however, requires the complementary quantity: how much support is erased as payloads leave the edge. Importantly, this loss should be measured against what the upstream 3DSG could provide under full residency, so that PORTER is charged only for information lost through offloading rather than support that was never available in the first place.

For requirement r, we therefore define the *support-survival ratio*

$$\rho_r(S) = \frac{c_r(S) + \epsilon_{\mathrm{ISE}}}{c_r(\mathcal{V}) + \epsilon_{\mathrm{ISE}}}, \qquad 0 < \rho_r(S) \le 1, \tag{4}$$

where $\epsilon_{\mathrm{ISE}} > 0$ is a small numerical stabilizer. Since $a_{vr}^A \le a_{vr}^F$, full residency provides maximal support and $\rho_r(\mathcal{V}) = 1$; smaller values indicate that more of the originally available support for requirement r has been erased locally.

A linear support drop, however, does not distinguish whether the remaining requirement is still well covered or close to losing its support entirely. We therefore calibrate relative erasure through a negative logarithm and define *Irreplaceable Support Erasure (ISE)* as

$$\mathrm{ISE}_q(S) = -\sum_{r \in \mathcal{R}_q} \omega_r \log_2 \rho_r(S) = \sum_{r \in \mathcal{R}_q} \omega_r \log_2 \frac{c_r(\mathcal{V}) + \epsilon_{\mathrm{ISE}}}{c_r(S) + \epsilon_{\mathrm{ISE}}}. \tag{5}$$

Thus, $\mathrm{ISE}_q(\mathcal{V}) = 0$, while the same absolute support drop incurs a larger penalty when the remaining coverage is already weak. For example, ignoring ϵ_{ISE} for illustration, reducing a requirement from 0.9 to 0.8 contributes only $\log_2(0.9/0.8) \approx 0.17$, whereas reducing it from 0.2 to 0.1 contributes 1 bit before weighting. The logarithm therefore protects requirements that are close to losing their remaining support, rather than treating all coverage drops equally. Because $c_r(S)$ is soft vision–language evidence rather than a calibrated probability, we use the logarithm as an information-style calibration of relative erasure rather than a Shannon entropy estimate.

The target residency set can then be defined by minimizing ISE under the local memory budget,

$$S_q^\star \in \arg\min_{S \subseteq \mathcal{V}} \mathrm{ISE}_q(S) \qquad \text{s.t.} \quad B(S) \le B. \tag{6}$$

ISE is the complement of a monotone submodular retained-support utility, making Eq. 6 a budgeted submodular maximization problem (Appendix A.1).

To turn this set-level objective into a payload-level decision, consider offloading P_v for $v \in S$. The Noisy-OR construction gives the resulting drop in requirement r exactly as

$$\delta_{vr}(S) := c_r(S) - c_r(S \setminus \{v\}) = \big(a^F_{vr} - a^A_{vr}\big) \prod_{u \neq v} \big(1 - e_{ur}(S)\big). \tag{7}$$

This factorization exposes replaceability: the first term is the support removed with P_v, while the product measures how much of that support is not already covered by the remaining objects.

Substituting this counterfactual drop into Eq. 5 yields the *marginal ISE* of offloading P_v,

$$\boxed{\Delta^{\mathrm{ISE}}_q(v; S) := \mathrm{ISE}_q(S \setminus \{v\}) - \mathrm{ISE}_q(S) = \sum_{r \in \mathcal{R}_q} \omega_r \log_2 \left(\frac{c_r(S) + \epsilon_{\mathrm{ISE}}}{c_r(S) - \delta_{vr}(S) + \epsilon_{\mathrm{ISE}}} \right)}. \tag{8}$$

Notably, the full-residency reference cancels in the marginal, leaving a purely state-dependent ratio between the requirement coverage before and after offloading P_v. Marginal ISE therefore captures two complementary effects: the counterfactual drop $\delta_{vr}(S)$ discounts support that can be replaced by substitutes, while the logarithmic ratio amplifies losses to requirements whose remaining coverage is already fragile. It asks not "how relevant is v?", but *what would the task lose if P_v were offloaded?*

Crucially, this value depends on the current residency set S. A payload that is redundant now may become irreplaceable after its substitutes are offloaded: both its uncovered support and the fragility of the corresponding requirement increase as the working set shrinks. Marginal ISE must therefore be recomputed as residency changes, which motivates the state-dependent scheduler introduced next.

3.4 ISE-Aware Residency Selection

We operationalize this objective with a lightweight reverse-greedy removal rule. At each step, PORTER offloads the resident payload whose marginal ISE per freed byte is smallest,

$$v^\star = \arg\min_{v \in S} \frac{\Delta^{\mathrm{ISE}}_q(v; S)}{b_v}. \tag{9}$$

Starting from full residency $S_0 = \mathcal{V}$, reverse greedy repeats this decision until the local budget $B(S) \leq B$ is satisfied, yielding the target residency set $\widehat{S}_q$. Crucially, $\Delta^{\mathrm{ISE}}_q(v; S)$ is recomputed after every offload: a payload that is redundant early can become costly to remove once its substitutes disappear. PORTER therefore allocates scarce local memory to support that is hardest to erase safely, rather than simply retaining the objects most relevant to the current task.

3.5 Residency Across Task Switches

The target $\widehat{S}_q$ depends on the current task, but not on the residency left by previous tasks; the latter determines only which transfers are needed. Given the current local set $\mathcal{L}_t$, PORTER realizes the new target by reconciling the two residency sets,

$$\mathrm{Push}_t = \mathcal{L}_t \setminus \widehat{S}_q, \qquad \mathrm{Pull}_t = \widehat{S}_q \setminus \mathcal{L}_t. \tag{10}$$

Push sends payloads that no longer belong in the target working set to the cloud, while Pull restores newly required payloads to the edge. Because their anchors persist locally, both operations preserve object identity without modifying the upstream 3DSG or reconstructing the scene representation. In the current implementation, a pushed payload is frozen while cloud-resident; Pull restores its latest persisted state, after which the upstream builder resumes native updates.

4 Experiments

We evaluate whether PORTER can reduce local 3DSG residency while preserving task capability under changing task demands. Our experiments address two questions: **(Q1)** Can PORTER act as a construction-agnostic residency layer across heterogeneous 3DSG builders without materially compromising their task capability? **(Q2)** Does PORTER preserve task capability more effectively than simpler offloading policies as local scene memory is progressively compressed?

4.1 EXPERIMENTAL SETUP

Benchmark and protocol. JITOMA-Bench (Chang et al., 2026a), built upon the real-world data of Clio-Bench (Maggio et al., 2024), contains three scenes with 35 Tier 2 and 55 Tier 3 ground-truth target cases. We use two complementary evaluation tracks. Tier 2 executes a sequence of tasks over the same persistent scene memory: after one task is evaluated, the next begins without resetting the accumulated representation. Tier 3 instead contains complex instructions involving multiple primary and latent targets, providing a complementary setting for testing how well a residency policy preserves multi-requirement task capability as local scene memory is progressively compressed.

Evaluation metrics and control. For Tier 2, we follow JITOMA-Bench and report top-1 **IoU**, **mR@1**, **mR@3**, **Objs**, **Peak**, and **TPF**. mRecall averages recall over IoU thresholds $\{0.1, 0.2, 0.3\}$; Objs and Peak are the average and maximum numbers of edge-resident objects, and TPF is the average time per frame. For Tier 3, we report **relative mR@3**, normalized to the Keep-All reference, against cumulative payload bytes offloaded. All methods share the same RGB-D trajectories, task annotations, and ground truth; PORTER changes only payload residency. Tier 3 policies additionally begin each task from the identical Keep-All ConceptGraphs state and differ only in payload-removal order. Implementation details and hyperparameters are provided in Appendix A.3.

4.2 CONSTRUCTION-AGNOSTIC RESIDENCY

To answer **Q1**, we deliberately evaluate PORTER on four 3DSG builders with substantially different construction philosophies: ConceptGraphs (Gu et al., 2024), a bottom-up open-vocabulary mapping pipeline; ReasoningGraph (Puigjaner et al., 2026), which emphasizes hierarchical object–relational reasoning; Clio (Maggio et al., 2024), a real-time task-driven scene graph; and DAAAM (Gorlo et al., 2026), which constructs hierarchical 4D spatio-temporal memory. Together, these methods span distinct design choices in when, how, and at what structure scene information is instantiated, providing a diverse testbed for PORTER's construction-agnostic design.

For each builder, we compare its native execution with the same method augmented by PORTER, while keeping its original construction procedure and parameters unchanged. The paired variants differ only in payload residency across task switches: PORTER changes which payload-bearing objects remain locally available for subsequent updates, while offloaded objects persist through their lightweight anchors and can be restored when needed. This paired design tests whether the same residency principle can reduce the local working representation across heterogeneous 3DSG paradigms without materially compromising the task capability inherited from each builder. Table 1 reports the resulting task-performance–efficiency trade-off.

Table 1: **Construction-agnostic evaluation on JITOMA-Bench (Chang et al., 2026a) Tier 2.** We pair four heterogeneous 3DSG builders with PORTER while keeping their native construction pipelines unchanged. For each scene, we report grounding accuracy (**IoU**, **mR@1**, and **mR@3**) and system efficiency (**Objs**, **Peak**, and **TPF**). The paired comparison tests whether PORTER can reduce the local payloads across different 3DSG paradigms while largely preserving the task capability.

3DSG Builder	Method	Apartment						Office						Cubicle					
		Accuracy ↑			*Efficiency*			*Accuracy* ↑			*Efficiency*			*Accuracy* ↑			*Efficiency*		
		IoU	mR@1	mR@3	Objs ↓	Peak ↓	TPF ↓	IoU	mR@1	mR@3	Objs ↓	Peak ↓	TPF ↓	IoU	mR@1	mR@3	Objs ↓	Peak ↓	TPF ↓
ConceptGraphs	Vanilla	17.8	33.3	52.8	452	1476	6.03	19.0	40.5	45.2	924	5796	12.58	11.3	22.2	63.0	224	745	5.37
	+ PORTER	17.6	33.3	55.6	83	341	3.96	19.2	40.5	45.2	128	1726	5.02	12.4	22.2	63.0	75	270	3.98
ReasoningGraph	Vanilla	12.6	33.3	33.3	276	488	2.91	17.3	42.9	45.2	261	605	3.05	12.9	22.2	29.6	149	260	2.92
	+ PORTER	12.6	33.3	33.3	5	92	2.91	17.3	42.9	45.2	4	83	3.05	12.9	22.2	29.6	4	59	2.92
Clio	Vanilla	10.9	25.0	27.8	4	637	0.35	17.7	38.1	47.6	5	678	0.42	23.8	66.7	74.1	5	408	0.46
	+ PORTER	10.9	25.0	27.8	3	74	0.38	17.9	40.5	47.6	3	65	0.40	24.5	66.7	74.1	5	75	0.41
DAAAM	Vanilla	11.6	22.2	33.3	739	1327	0.16	14.2	33.3	50.0	713	1738	0.14	12.6	29.6	48.2	417	710	0.15
	+ PORTER	13.9	27.8	33.3	61	745	0.16	14.9	35.7	46.7	134	602	0.14	14.2	33.3	51.9	196	442	0.15

Results. Table 1 shows that PORTER consistently reduces local residency across all four 3DSG builders while preserving task capability. Across ConceptGraphs, ReasoningGraph, Clio, and DAAAM, PORTER reduces the average resident object count by 82.1%, 98.1%, 21.4%, and 79.1%, respectively. These results support PORTER's use as a construction-agnostic residency layer over heterogeneous 3DSGs under a unified task-conditioned residency objective. Accuracy remains

broadly comparable across IoU, mR@1, and mR@3, with several entries even improving. In some cases, reducing the resident candidate set can suppress distracting evidence while preserving useful retrievals, as illustrated by the Clio case studies in Appendix A.7. This suggests that ISE preserves irreplaceable task support while removing redundant payloads.

Beyond memory reduction, PORTER improves online efficiency when computation scales with resident objects. For ConceptGraphs, PORTER reduces average TPF from 6.03s to 3.96s in Apartment, 12.58s to 5.02s in Office, and 5.37s to 3.98s in Cubicle. For builders with weaker runtime dependence on residency, such as Clio and DAAAM, PORTER primarily provides memory reduction while maintaining similar execution time without modifying their native construction pipelines.

4.3 Residency under Progressive Compression

To answer **Q2**, we fix the upstream representation to the complete ConceptGraphs (Gu et al., 2024) reconstruction and use the Tier 3 tasks of JITOMA-Bench (Chang et al., 2026a). Their multiple primary and latent targets provide a discriminative setting for testing whether an offloading policy preserves the different functional requirements of a task rather than only its most salient object. Using a fixed, task-agnostic 3DSG also isolates the residency decision from any upstream construction.

For every Tier 3 task, all policies start from the same Keep-All state. Each policy then offloads one object payload at a time according to its own ranking rule. After every offload, the task is re-evaluated using only the payload-bearing objects that remain locally resident; offloaded payloads do not participate in that evaluation. At each cumulative payload-byte checkpoint, we pool mR@3 across all Tier 3 tasks and normalize it by the pooled Keep-All mR@3 over the same task set. The resulting trajectory measures how gracefully each policy preserves task capability as increasingly more of the local 3DSG is moved to the cloud.

Compared policies. We compare five offloading strategies. **Random** removes payloads in random order and provides a task-agnostic reference. **Largest-First** greedily removes the largest remaining payload, capturing a purely memory-driven strategy. **CLIP/Byte** combines direct task–object semantic relevance with payload size. **Requirement/Byte** decomposes the task into the same functional requirements used by PORTER and scores objects from their requirement support, but does not model substitute coverage. **PORTER** instead ranks each candidate by its current marginal ISE per byte and recomputes this quantity after every removal. Additional comparisons with redundancy-aware baselines are provided in Appendix A.6.

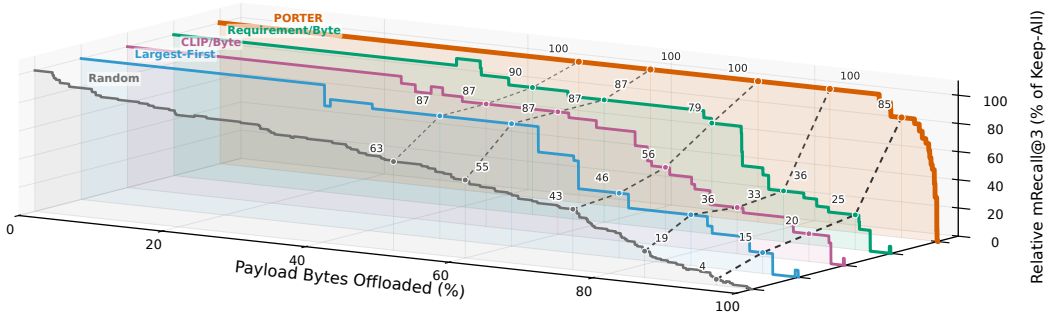

Figure 2: **Task retention under progressive 3DSG compression.** All policies start each Tier 3 task from the same ConceptGraphs representation and sequentially offload object payloads. The horizontal axis reports cumulative payload bytes offloaded, while the vertical axis reports mR@3 relative to Keep-All. Methods are separated along the depth axis only for visualization; dashed lines compare policies at matched offloading ratios of 50%, 60%, 75%, 85%, and 95%.

Results. Figure 2 reveals a separation between relevance-based selection and state-dependent residency. PORTER retains 100% of Keep-All mR@3 through the 85% plotted checkpoint (91% on the full trajectory), whereas the strongest competing policy in Figure 2 retains only 90%, 87%, 79%, and 36% at the matched 50%, 60%, 75%, and 85% checkpoints, respectively. At 95% offloading, PORTER still preserves 85% of the Keep-All capability, while the baseline retains only 25%.

The baseline ordering clarifies where the gain comes from. Random and Largest-First ignore task value; CLIP/Byte adds semantic relevance, and Requirement/Byte further resolves functional requirements, yet both degrade once compression becomes aggressive. Figure 2 therefore focuses on this diagnostic progression, while stronger redundancy-aware set-selection baselines are evaluated in Appendix A.6. These baselines remain competitive under moderate compression but fall behind PORTER in the high-compression regime, consistent with the view that redundancy-aware coverage explains much of the early robustness, while ISE's fragility-aware protection becomes important as the resident set approaches the task-conditioned working set.

4.4 Real-Robot Deployment

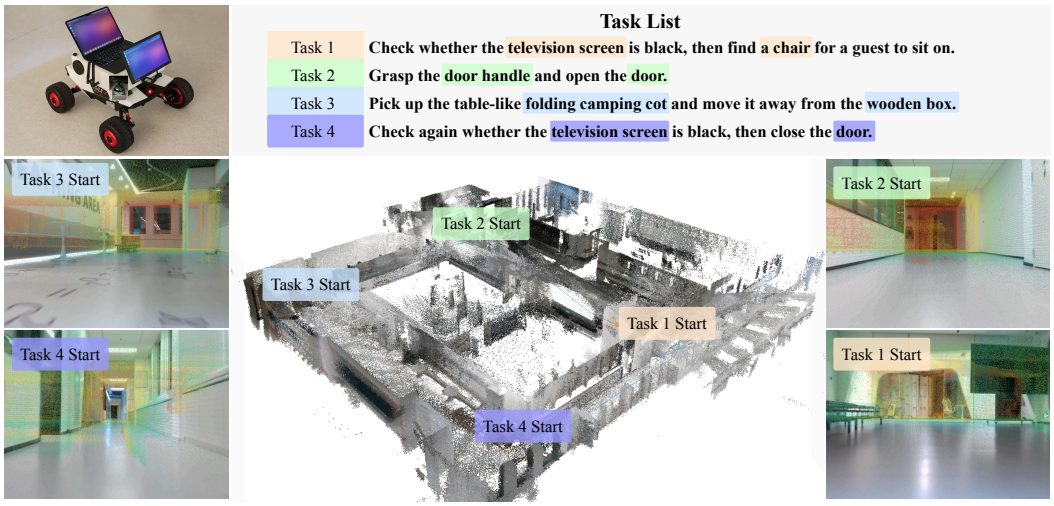

Figure 3: **Real-robot deployment.** The robot continuously maps an indoor environment while executing four consecutive tasks. The central reconstruction shows the task start locations; surrounding views show the corresponding observations. Repeated requirements, such as the television and door, test whether offloaded scene knowledge can later be restored when it becomes useful again.

We further evaluate PORTER on a real mobile robot under continuous scene construction and task switching. The platform uses an Intel RealSense D435 RGB-D camera and two Livox Mid-360 LiDARs, with FAST-LIO2 (Xu et al., 2022) providing odometry. Registered LiDAR geometry is reprojected into the camera views to refine depth, and the resulting RGB-D sequence is used to construct ConceptGraphs and annotate task-relevant 3D ground truth. We execute four consecutive tasks over a 1,183-frame indoor trajectory, including repeated requirements that test whether previously offloaded knowledge can be restored (Figure 3). We report mR@3 using the benchmark IoU thresholds; additional sensing, annotation, and communication details are provided in Appendix A.5.

Compared with vanilla ConceptGraphs, PORTER reduces the average number of locally resident objects from 278 to 114 (59.1%) and the peak from 540 to 224 (58.5%), while preserving the same mR@3 of 75.0%. The smaller local working set also reduces mapping time per frame from 6.276 s to 1.863 s (70.3%). Figure 4 shows the mechanism behind this gain: each task switch sharply removes payloads that are no longer needed, while continued mapping subsequently grows the local set again. Crucially, the later switches also restore previously offloaded objects—T3 and T4 trigger 4 and 9 Pulls, respectively—showing that PORTER changes *residency* rather than irreversibly forgetting scene knowledge.

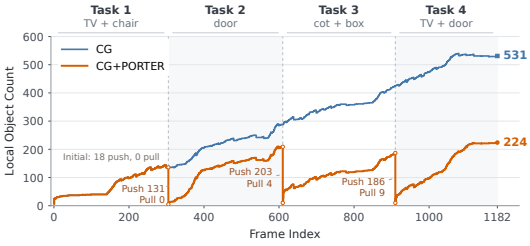

Figure 4: **Real-robot residency.** Vanilla ConceptGraphs continually accumulates objects, whereas PORTER contracts the working set at each task switch and resumes mapping from the new task-conditioned residency state.

4.5 COMPONENT ABLATIONS

Having established PORTER's effectiveness in both controlled benchmarks and real-robot deployment, we next isolate the design choices responsible for its advantage. We follow the same Tier 3 progressive-compression setting as Sec. 4.3, changing one component at a time. **w/o Noisy-OR** replaces Noisy-OR coverage with maximum support; **w/o Log-Information Loss** replaces negative-log survival with linear coverage loss; **w/o Byte Normalization** ranks payloads by marginal ISE alone; and **w/o Dynamic Re-ranking** freezes priorities computed from the Keep-All state.

Table 2: **Component ablation of PORTER on JITOMA-Bench Tier 3.** Overall nAUC covers the full payload-byte offloading curve and Tail nAUC the high-compression interval $[95\%, 100\%]$; remaining entries report pooled mR@3 relative to the pooled Keep-All reference.

	Curve nAUC (%) ↑		Relative $\mathrm{mRecall@3}$ at ρ_B (%) ↑				
Variant	Overall	Tail	95%	97%	99%	99.5%	99.9%
PORTER w/o Noisy-OR	61.20	35.93	39.34	39.34	36.07	26.23	0.00
PORTER w/o Byte Normalization	97.89	62.42	80.33	72.13	45.90	16.39	0.00
PORTER w/o Log-Information Loss	97.95	66.55	81.97	72.13	50.82	40.98	19.67
PORTER w/o Dynamic Re-ranking	98.27	69.51	85.25	77.05	54.10	40.98	19.67
PORTER	**98.30**	**72.03**	**85.25**	**81.97**	**59.02**	**50.82**	**31.15**

Results. Table 2 shows that full PORTER achieves the best Overall and Tail nAUC (98.30% and 72.03%), with the largest drop caused by removing Noisy-OR (61.20% and 35.93%). Most other differences emerge only under high compression, when irrelevant and redundant payloads have largely been removed and the resident set approaches the task-relevant working set.

The tail results expose complementary roles of PORTER's components: Noisy-OR captures replaceability, dynamic re-ranking updates it as substitutes disappear, log-survival protects fragile requirements, and byte normalization balances task loss against memory released. At 99.9% offloading, PORTER retains 31.15% of Keep-All mR@3, versus 19.67% without dynamic re-ranking or log-information loss, while the no-byte and no-Noisy-OR variants reach zero. We treat this tail as a stress test rather than a nominal operating point. Parameter sensitivity is reported in Appendix A.4.

4.6 RUNTIME AND ONLINE POTENTIAL

Our current implementation recomputes PORTER only at task switches. Across 25 Tier 3 cases, the full residency core takes only 2.034 ms median (P95: 3.938 ms) on one pinned AMD EPYC 7542 logical CPU. Together with the linear complexity derived in Appendix A.2.1, this suggests that ISE can support more frequent or asynchronous residency updates as task context, memory pressure, or scene state evolves. Detailed timings and scene-size scaling are reported in Appendix A.2.2. These timings exclude physical payload transfer. In our real-robot deployment, bulk Push transfers run asynchronously with mapping, while Pull is synchronous because newly required payloads must be restored before use. As illustrated in Figure 4, the task-critical Pull path is sparse: T3 and T4 restore 3.72 MB and 19.23 MB in 0.76 s and 1.24 s, respectively, while the substantially larger Push transfers run concurrently with continued mapping. Detailed transfer measurements and connectivity considerations are provided in Appendix A.5.4.

5 CONCLUSION

We introduced PORTER, a construction-agnostic edge–cloud residency layer for persistent 3DSG memory that decouples persistence from local residency. Its Irreplaceable Support Erasure (ISE) objective measures task-support loss by accounting for substitute support and residual fragility. Across heterogeneous 3DSG builders, progressive compression, and real-robot deployment, PORTER reduces local residency while preserving task capability, with additional mapping savings when computation scales with the resident set. Its millisecond-scale decision cost further supports ISE as a lightweight criterion for memory adaptation. Overall, our results suggest that persistent scene knowledge can remain recoverable without remaining continuously resident on the edge.

AI USE STATEMENT

In this work, we did not use generative AI tools for research ideation or generating synthetic data. We used generative AI tools to assist with writing, presentation, and proofreading mathematical derivations for possible omissions, inconsistencies, or algebraic errors. All mathematical statements and derivations in the paper were independently checked and verified by the authors. Generative AI was not used to generate figures or experimental results. We take responsibility for the final content of this work, including text or claims produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce PORTER and our experiments throughout the main paper and appendix. The complete method, including requirement support, ISE, marginal residency selection, and Push/Pull reconciliation, is specified in Secs. 3.2–3.5, with derivations and theoretical properties in Appendix A.1. Experimental protocols, metrics, and comparison settings are described in Sec. 4.1 and the corresponding experiment sections. Implementation details, default hyperparameters, upstream 3DSG configurations, and residency-policy definitions are provided in Appendix A.3; hyperparameter sensitivity is reported in Appendix A.4. Real-robot sensing, ground-truth annotation, task sequencing, and edge–cloud transfer details are given in Appendix A.5. Computational complexity and runtime measurement protocols are documented in Appendices A.2.1 and A.2.2. We plan to release the implementation and experiment configurations upon publication.

REFERENCES

Christopher Agia, Krishna Murthy Jatavallabhula, Mohamed Khodeir, Ondrej Miksik, Vibhav Vineet, Mustafa Mukadam, Liam Paull, and Florian Shkurti. Taskography: Evaluating robot task planning over large 3d scene graphs. In *Conference on Robot Learning*, pp. 46–58. PMLR, 2022.

Iro Armeni, Zhi-Yang He, JunYoung Gwak, Amir R Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. 3d scene graph: A structure for unified semantics, 3d space, and camera. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5664–5673, 2019.

Yue Chang, Rufeng Chen, Yifan Tian, Dazhi Huang, Zhaofan Zhang, Yi Chen, Wenze Zhang, Li Chen, Hui Xiong, and Sihong Xie. Just-in-time scene graph growth: Combating perceptual saturation in long-horizon robotics. *arXiv preprint arXiv:2607.13245*, 2026a.

Yue Chang, Rufeng Chen, Zhaofan Zhang, Yi Chen, Yifan Tian, and Sihong Xie. Rag-3dsg: Enhancing 3d scene graphs with re-shot guided retrieval-augmented generation. *arXiv preprint arXiv:2601.10168*, 2026b.

Rufeng Chen, Yue Chang, Xiaqiang Tang, Hechang Chen, and Sihong Xie. Psg-nav: Probabilistic scene graph navigation via multiverse decision making. *arXiv preprint arXiv:2606.01313*, 2026.

Samir Yitzhak Gadre, Mitchell Wortsman, Gabriel Ilharco, Ludwig Schmidt, and Shuran Song. Clip on wheels: Zero-shot object navigation as object localization and exploration. *arXiv preprint arXiv:2203.10421*, 3(4):7, 2022.

Paul Gay, James Stuart, and Alessio Del Bue. Visual graphs from motion (vgfm): Scene understanding with object geometry reasoning. In *Asian Conference on Computer Vision*, pp. 330–346. Springer, 2018.

Nicolas Gorlo, Lukas Schmid, and Luca Carlone. Describe anything anywhere at any moment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 35002–35013, 2026.

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 (ICRA)*, pp. 5021–5028. IEEE, 2024.

Daniel Honerkamp, Martin Büchner, Fabien Despinoy, Tim Welschehold, and Abhinav Valada. Language-grounded dynamic scene graphs for interactive object search with mobile manipulation. *IEEE Robotics and Automation Letters*, 2024.

Nathan Hughes, Yun Chang, and Luca Carlone. Hydra: A real-time spatial perception system for 3d scene graph construction and optimization. *arXiv preprint arXiv:2201.13360*, 2022.

Krishna Murthy Jatavallabhula, Alihusein Kuwajerwala, Qiao Gu, Mohd Omama, Tao Chen, Alaa Maalouf, Shuang Li, Ganesh Iyer, Soroush Saryazdi, Nikhil Keetha, et al. Conceptfusion: Open-set multimodal 3d mapping. *arXiv preprint arXiv:2302.07241*, 2023.

Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David Shamma, Michael Bernstein, and Li Fei-Fei. Image retrieval using scene graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3668–3678, 2015.

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.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International journal of computer vision*, 123(1):32–73, 2017.

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 (ICRA)*, pp. 13582–13589. IEEE, 2025.

Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *European conference on computer vision*, pp. 852–869. Springer, 2016.

Dominic Maggio and Luca Carlone. Bayesian fields: Task-driven open-set semantic gaussian splatting. *arXiv preprint arXiv:2503.05949*, 2025.

Dominic Maggio, Yun Chang, Nathan Hughes, Matthew Trang, Dan Griffith, Carlyn Dougherty, Eric Cristofalo, Lukas Schmid, and Luca Carlone. Clio: Real-time task-driven open-set 3d scene graphs. *IEEE Robotics and Automation Letters*, 2024.

Dominic Maggio, Nicolas Gorlo, and Luca Carlone. Found-it: Foundation-model-first task-driven 3d scene graphs with granularity on demand. *arXiv preprint arXiv:2605.25371*, 2026.

Albert Gassol Puigjaner, Angelos Zacharia, and Kostas Alexis. Relationship-aware hierarchical 3d scene graph for task reasoning. *arXiv preprint arXiv:2602.02456*, 2026.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. *arXiv preprint arXiv:2307.06135*, 2023.

Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. Kimera: From slam to spatial perception with 3d dynamic scene graphs. *The International Journal of Robotics Research*, 40(12-14):1510–1546, 2021.

Dennis Rotondi, Francesco Argenziano, Sebastian Koch, Nathan Hughes, Martin Buechner, Johanna Wald, Lukas Rosenberger Schmid, Daniele Nardi, Abhinav Valada, Liam Paull, et al. 3d scene graphs: Open challenges and future directions. *arXiv preprint arXiv:2606.19383*, 2026.

Saumya Saxena, Blake Buchanan, Chris Paxton, Peiqi Liu, Bingqing Chen, Narunas Vaskevicius, Luigi Palmieri, Jonathan Francis, and Oliver Kroemer. Grapheqa: Using 3d semantic scene graphs for real-time embodied question answering. *arXiv preprint arXiv:2412.14480*, 2024.

Dhruv Shah, Błażej Osiński, Sergey Levine, et al. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on robot learning*, pp. 492–504. PMLR, 2023.

Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004. ISSN 0167-6377. doi: https://doi.org/10.1016/S0167-6377(03)00062-2. URL https://www.sciencedirect.com/science/article/pii/S0167637703000622.

Zixuan Wang, Bo Yu, Junzhe Zhao, Wenhao Sun, Sai Hou, Shuai Liang, Xing Hu, Yinhe Han, and Yiming Gan. Karma: Augmenting embodied ai agents with long-and-short term memory systems. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2025.

Abdelrhman Werby, Chenguang Huang, Martin Büchner, Abhinav Valada, and Wolfram Burgard. Hierarchical open-vocabulary 3d scene graphs for language-grounded robot navigation. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024.

Shun-Cheng Wu, Johanna Wald, Keisuke Tateno, Nassir Navab, and Federico Tombari. Scenegraphfusion: Incremental 3d scene graph prediction from rgb-d sequences. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7515–7525, 2021.

Ziyi Xia, Chaoran Xiong, Litao Wei, Xinhao Hu, and Ling Pei. Exploring bottlenecks in vlm-llm navigation: How 3d scene understanding capability impacts zero-shot vln. *arXiv preprint arXiv:2605.14801*, 2026.

Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics*, 38(4):2053–2073, 2022.

Kashu Yamazaki, Taisei Hanyu, Khoa Vo, Thang Pham, Minh Tran, Gianfranco Doretto, Anh Nguyen, and Ngan Le. Open-fusion: Real-time open-vocabulary 3d mapping and queryable scene representation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9411–9417. IEEE, 2024.

Zhijie Yan, Shufei Li, Zuoxu Wang, Lixiu Wu, Han Wang, Jun Zhu, Lijiang Chen, and Jihong Liu. Dynamic open-vocabulary 3d scene graphs for long-term language-guided mobile manipulation. *IEEE Robotics and Automation Letters*, 2025.

Hang Yin, Xiuwei Xu, Zhenyu Wu, Jie Zhou, and Jiwen Lu. Sg-nav: Online 3d scene graph prompting for llm-based zero-shot object navigation. *Advances in neural information processing systems*, 37:5285–5307, 2024.

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.

A Appendix

A.1 Derivation and Properties of Irreplaceable Support Erasure

We derive the marginal Irreplaceable Support Erasure (ISE) used in Sec. 3.3 and summarize several properties that clarify its behavior.

Requirement-wise support drop. Recall that the Noisy-OR coverage of requirement r under residency set S is

$$c_r(S) = 1 - \prod_{u \in \mathcal{V}} \big(1 - e_{ur}(S)\big). \tag{11}$$

For $v \in S$, offloading P_v changes only its effective support from a^F_{vr} to a^A_{vr}. Let

$$Q_{vr}(S) = \prod_{u \neq v} \big(1 - e_{ur}(S)\big).$$

Then

$$c_r(S) = 1 - \big(1 - a^F_{vr}\big) Q_{vr}(S), \tag{12}$$

$$c_r(S \setminus \{v\}) = 1 - \big(1 - a^A_{vr}\big) Q_{vr}(S), \tag{13}$$

and hence

$$\delta_{vr}(S) = c_r(S) - c_r(S \setminus \{v\}) \tag{14}$$

$$= \big(a^F_{vr} - a^A_{vr}\big) \prod_{u \neq v} \big(1 - e_{ur}(S)\big). \tag{15}$$

The first factor is the support removed with P_v, while the second is the portion of requirement r not already covered by alternative objects. Thus, the same payload incurs little loss when substitutes remain and becomes more valuable as those substitutes disappear.

Marginal ISE. From Eq. 5,

$$\mathrm{ISE}_q(S) = \sum_{r \in \mathcal{R}_q} \omega_r \log_2 \frac{c_r(\mathcal{V}) + \epsilon_{\mathrm{ISE}}}{c_r(S) + \epsilon_{\mathrm{ISE}}}. \tag{16}$$

The marginal erasure caused by offloading P_v is

$$\Delta^{\mathrm{ISE}}_q(v; S) = \mathrm{ISE}_q(S \setminus \{v\}) - \mathrm{ISE}_q(S) \tag{17}$$

$$= \sum_{r \in \mathcal{R}_q} \omega_r \log_2 \frac{c_r(S) + \epsilon_{\mathrm{ISE}}}{c_r(S \setminus \{v\}) + \epsilon_{\mathrm{ISE}}}. \tag{18}$$

The full-residency reference $c_r(\mathcal{V})$ therefore cancels in the marginal: it defines the set-level erasure baseline, but the next offloading decision depends only on the current residency state.

Using $c_r(S \setminus \{v\}) = c_r(S) - \delta_{vr}(S)$ gives

$$\Delta^{\mathrm{ISE}}_q(v; S) = \sum_{r \in \mathcal{R}_q} \omega_r \log_2 \left(\frac{c_r(S) + \epsilon_{\mathrm{ISE}}}{c_r(S) - \delta_{vr}(S) + \epsilon_{\mathrm{ISE}}} \right). \tag{19}$$

Set-function structure. **Proposition.** Assume $0 \leq a^A_{vr} \leq a^F_{vr} < 1$ and $\omega_r \geq 0$. Then, for every requirement r, the Noisy-OR coverage $c_r(S)$ is a monotone submodular function of the resident set S. Moreover, the normalized retained log-support utility

$$F_q(S) := \sum_{r \in \mathcal{R}_q} \omega_r \log_2 \frac{c_r(S) + \epsilon_{\mathrm{ISE}}}{c_r(\emptyset) + \epsilon_{\mathrm{ISE}}} \tag{20}$$

is nonnegative, normalized, monotone, and submodular. Consequently,

$$\mathrm{ISE}_q(S) = F_q(\mathcal{V}) - F_q(S) \tag{21}$$

is monotone non-increasing and supermodular.

Proof. For a fixed requirement r, define

$$\lambda_{vr} := \frac{1 - a^F_{vr}}{1 - a^A_{vr}} \in (0, 1], \qquad \kappa_r := \prod_{u \in \mathcal{V}} \big(1 - a^A_{ur}\big).$$

Because an object contributes a^F_{vr} when resident and a^A_{vr} otherwise, the uncovered support can be written as

$$1 - c_r(S) = \kappa_r \prod_{u \in S} \lambda_{ur}. \tag{22}$$

Hence, for $v \notin S$,

$$c_r(S \cup \{v\}) - c_r(S) = \kappa_r (1 - \lambda_{vr}) \prod_{u \in S} \lambda_{ur}. \tag{23}$$

Since every $\lambda_{ur} \leq 1$, for any $A \subseteq B$ and $v \notin B$,

$$c_r(A \cup \{v\}) - c_r(A) \geq c_r(B \cup \{v\}) - c_r(B), \tag{24}$$

which establishes diminishing returns; monotonicity follows because the marginal gain is nonnegative. Thus, c_r is monotone submodular.

Now let

$$\Delta c_r(v \mid S) := c_r(S \cup \{v\}) - c_r(S).$$

The marginal gain of the corresponding log-support term is

$$\Delta F_r(v \mid S) = \log_2 \frac{c_r(S \cup \{v\}) + \epsilon_{\mathrm{ISE}}}{c_r(S) + \epsilon_{\mathrm{ISE}}} \tag{25}$$

$$= \log_2 \left(1 + \frac{\Delta c_r(v \mid S)}{c_r(S) + \epsilon_{\mathrm{ISE}}} \right). \tag{26}$$

For $A \subseteq B$, submodularity of c_r gives $\Delta c_r(v \mid A) \geq \Delta c_r(v \mid B)$, while monotonicity gives $c_r(A) \leq c_r(B)$. Therefore,

$$\frac{\Delta c_r(v \mid A)}{c_r(A) + \epsilon_{\mathrm{ISE}}} \geq \frac{\Delta c_r(v \mid B)}{c_r(B) + \epsilon_{\mathrm{ISE}}},$$

and hence $\Delta F_r(v \mid A) \geq \Delta F_r(v \mid B)$. Each log-support term is therefore monotone submodular, and so is their nonnegative weighted sum F_q.

Finally, since $\mathrm{ISE}_q(S) = F_q(\mathcal{V}) - F_q(S)$, ISE is the complement of a monotone submodular retained-support utility and is therefore monotone non-increasing and supermodular. □

Equivalently, for $A \subseteq B$ and $v \in A$,

$$\Delta^{\mathrm{ISE}}_q(v; A) \geq \Delta^{\mathrm{ISE}}_q(v; B), \tag{27}$$

so removing a payload can only become more costly as the resident set shrinks. This formally captures PORTER's state-dependent repricing: substitutes reduce marginal erasure while they remain available, and the surviving payloads become progressively harder to remove as that redundancy disappears.

Relation to submodular knapsack optimization. The preceding proposition gives an equivalent retained-utility view of the PORTER objective. Since

$$\mathrm{ISE}_q(S) = F_q(\mathcal{V}) - F_q(S),$$

minimizing ISE under the local memory budget is equivalent to

$$S^\star_q \in \arg\max_{S \subseteq \mathcal{V}} F_q(S) \qquad \text{s.t.} \qquad \sum_{v \in S} b_v \leq B. \tag{28}$$

Because F_q is monotone submodular and the payload footprint is modular, the target-residency problem is an instance of monotone submodular maximization under a knapsack constraint. Classical forward-selection algorithms for this problem admit constant-factor approximation guarantees; in

particular, a $(1 - 1/e)$ approximation is achievable with the standard enumeration-assisted greedy construction (Sviridenko, 2004).

PORTER does not use this forward construction directly. Instead, it starts from the currently complete resident set and repeatedly removes the payload with the smallest marginal ISE per freed byte, as in Eq. 9. This reverse-greedy form directly produces an eviction ordering and naturally supports progressive memory release as the budget tightens. Although it exploits the same diminishing-returns structure, the classical $(1 - 1/e)$ guarantee for forward submodular-knapsack maximization does not directly apply to this removal rule, and we do not claim such a guarantee for the current scheduler.

Consistency with sequential offloading. Consider a removal sequence $S_0 = \mathcal{V} \supset S_1 \supset \cdots \supset S_T$, where $S_t = S_{t-1} \setminus \{v_t\}$. The marginal erasures telescope:

$$\sum_{t=1}^{T} \Delta_q^{\mathrm{ISE}}(v_t; S_{t-1}) = \mathrm{ISE}_q(S_T) - \mathrm{ISE}_q(\mathcal{V}) \tag{29}$$

$$= \mathrm{ISE}_q(S_T), \tag{30}$$

since $\mathrm{ISE}_q(\mathcal{V}) = 0$. Thus, sequential marginal erasure is consistent with set-level ISE.

Local interpretation. For a small requirement-wise drop $\delta_{vr}(S)$,

$$\log_2 \frac{c_r(S) + \epsilon_{\mathrm{ISE}}}{c_r(S) - \delta_{vr}(S) + \epsilon_{\mathrm{ISE}}} \approx \frac{\delta_{vr}(S)}{(c_r(S) + \epsilon_{\mathrm{ISE}}) \ln 2}. \tag{31}$$

Locally, ISE can therefore be interpreted as irreplaceable support loss weighted by requirement fragility: the same absolute loss receives a larger penalty when the available support is already weak. This distinguishes ISE from a linear coverage-loss objective.

A.2 Computational Complexity and Runtime Scalability

We complement the runtime results in Sec. 4.6 with the theoretical complexity of PORTER and detailed latency measurements. The former characterizes how residency reasoning scales with scene size, while the latter verifies this behavior on the Tier 3 scene graphs used in our experiments.

A.2.1 Theoretical Complexity

PORTER's computation follows the two task-conditioned stages of the method: constructing requirement support from persistent anchors and selecting the local residency set using marginal ISE. We analyze these stages after task decomposition and text embeddings are available. Let $N = |\mathcal{V}|$ be the number of objects in the persistent 3DSG, $M = |\mathcal{R}_q|$ the number of task requirements, d the embedding dimension, and K the number of candidate supporters retained per requirement.

Requirement support. Because every object retains its lightweight anchor locally, PORTER can score a new task without accessing its heavy payloads. Computing the similarity s_{vr} between all N object anchors and M requirement embeddings costs

$$T_{\mathrm{support}} = \mathcal{O}\big(NM(d + \log K)\big), \tag{32}$$

where the $\log K$ term accounts for retaining the top-K supporters of each requirement. For fixed K, this is dominated by the NMd similarity computation.

More importantly, top-K support bounds the number of objects that can have nonzero marginal ISE. Let

$$\mathcal{C}_q = \bigcup_{r \in \mathcal{R}_q} \mathcal{K}_r, \qquad C = |\mathcal{C}_q|. \tag{33}$$

Then

$$C \le \min(N, KM). \tag{34}$$

Objects outside $\mathcal{C}_q$ have zero support for every requirement under Eq. 1, and therefore zero marginal ISE. Thus, although task support requires scanning all N persistent anchors, only the support-active set $\mathcal{C}_q$ enters the nontrivial residency decisions.

Marginal ISE selection. For a current residency set S, evaluating $\Delta_q^{\mathrm{ISE}}(v; S)$ for one candidate requires aggregating over the M requirements and therefore costs $\mathcal{O}(M)$. If $L \leq C$ active payloads are removed before the memory budget is satisfied, reverse greedy evaluates at most $C-\ell$ candidates at removal step ℓ. Hence,

$$T_{\mathrm{ISE}} = \mathcal{O}\left(M \sum_{\ell=0}^{L-1} (C - \ell)\right) \tag{35}$$

$$= \mathcal{O}(LCM) \subseteq \mathcal{O}(C^2 M). \tag{36}$$

Combining requirement support and marginal ISE selection gives

$$T_{\mathrm{PORTER}} = \mathcal{O}\big(NM(d + \log K) + LCM\big). \tag{37}$$

Since $C \leq KM$ and $L \leq C$, the ISE-selection term is independent of scene size N. Under PORTER's fixed top-K construction, with bounded task requirements and a fixed embedding dimension, the overall complexity with respect to scene size reduces to

$$T_{\mathrm{PORTER}} = \mathcal{O}(N). \tag{38}$$

Thus, scene growth increases only the cost of searching persistent anchors, while state-dependent ISE reasoning remains bounded by the task-conditioned support set, preserving linear scaling as persistent scene memory grows.

A.2.2 Runtime Measurements

Protocol. We measure PORTER on the ConceptGraphs representations used in the Tier 3 experiments. Cubicle, Apartment, and Office contain 229, 429, and 922 objects and contribute 11, 8, and 6 tasks, respectively, yielding 25 task-switch cases. The tasks contain 55 requirement instances over 42 distinct labels, with $M \in \{2, 3\}$ and CLIP embedding dimension $d = 1024$. Tier 3 oracle object labels are used directly as requirements and their OpenCLIP text embeddings are precomputed, isolating PORTER's residency computation from external language-model inference. Timings are collected on one pinned logical CPU of an AMD EPYC 7542.

Table 3: Stage-wise PORTER latency over 25 Tier 3 task-switch cases on one pinned logical CPU. Values are median and 95th-percentile latency.

Stage	Median (ms)	P95 (ms)
Requirement support	0.716	1.412
Marginal ISE selection	0.963	1.612
Support + ISE	1.686	3.007
Push/Pull reconciliation	0.304	0.859
PORTER core	**2.034**	**3.938**

We separately measure requirement-support construction, marginal ISE selection, and Push/Pull reconciliation. Subtotal and total statistics are computed from the jointly measured per-invocation latency before aggregation; consequently, their medians need not equal the sum of independently reported stage medians. Table 3 reports the resulting stage-wise and PORTER-core latencies.

Scaling with scene size. Table 4 breaks down PORTER-core latency by scene. As the number of persistent objects increases from 229 to 429 and 922, median latency rises from 1.679 ms to 2.197 ms and 3.492 ms, respectively, while P95 remains below 4.4 ms in all three scenes. These measurements provide an empirical counterpart to Eq. 38: increasing persistent scene size primarily increases anchor scoring, while the task-conditioned ISE reasoning remains small.

A.3 Implementation Details

PORTER is implemented as a lightweight residency layer over an upstream 3DSG. It operates on an object-level interface consisting of a persistent identity, a vision–language embedding, the byte size of the migratable payload, and its current residency state. The upstream builder retains control of scene construction and object updates, while PORTER determines only which acquired payloads should remain locally resident.

Table 4: PORTER-core latency across Tier 3 scenes of different sizes.

Scene	Objects N	Median (ms)	P95 (ms)
Cubicle	229	1.679	2.214
Apartment	429	2.197	2.841
Office	922	3.492	4.324

A.3.1 PORTER Deployment and Configuration

Anchor–payload instantiation. For each upstream object v, PORTER separates a lightweight persistent anchor A_v from its builder-specific payload P_v. The anchor retains the object identity and normalized vision–language embedding required for retrieval and residency reasoning. The payload contains the complete object state required to restore the object, including its geometric and accumulated observation data. We define b_v as the serialized size of this native payload. Anchor storage is excluded from the migratable-payload budget because anchors remain local under every residency policy.

Anchor memory scaling. Anchors are lightweight but not free: because one anchor is retained for every persistent object, their total footprint grows linearly with catalog size. In our current implementation, the dominant fixed-size anchor field is a 1024-dimensional FP32 vision–language embedding, requiring $1024 \times 4 = 4096$ bytes (4 KiB) per object. The embedding buffer alone therefore requires approximately 3.91 MiB, 39.06 MiB, 390.63 MiB, and 3.81 GiB for 10^3, 10^4, 10^5, and 10^6 objects, respectively. Additional identities, indexing structures, and runtime metadata introduce further implementation-dependent overhead.

PORTER targets the substantially heavier builder-specific payload term rather than making persistent metadata independent of scene size. Consequently, the edge footprint remains linear in the number of persistent anchors even when payload residency is tightly bounded. Compressing, hierarchically indexing, or forgetting long-lived anchors is complementary to PORTER and outside the scope of the present residency layer.

Update semantics for cloud-resident payloads. PORTER treats Push as a residency transition rather than a distributed update protocol. Once a payload is pushed, its cloud copy remains frozen at the last persisted state and does not receive builder-specific updates from subsequent observations. If the payload is later pulled, PORTER restores that state to the edge, after which the upstream 3DSG builder resumes its native object-update procedure. Lightweight anchors remain locally available for identity and residency reasoning throughout this interval.

This choice preserves PORTER's construction-agnostic interface: propagating new observations into cloud-resident payloads would require builder-specific association, fusion, and conflict-resolution rules for heterogeneous payload representations. Such remote synchronization is complementary to PORTER but outside the scope of the present residency layer. Our current experiments validate restoration across task switches but do not isolate objects that change while their payloads are cloud-resident. Measuring staleness under long-horizon dynamic scenes, and combining PORTER with builder-specific delta logging or cloud-side update mechanisms, are therefore natural extensions.

Requirement interface and parser instantiation. PORTER operates on a supplied set of functional requirements and is agnostic to how this set is obtained. For the Tier 2 experiments, we instantiate this interface using GPT-5.2 with the fixed prompt below. The parsed requirement set is fixed for each task and shared across all paired comparisons. Although the parser output schema includes a `weight` field, we do not use model-generated weights; all requirements are assigned uniformly as $\omega_r = 1/|\mathcal{R}_q|$, following Sec. 3.2. For Tier 3, we instead use the benchmark-provided target labels directly as requirements, thereby isolating residency reasoning from requirement-generation errors. Ground-truth 3D boxes remain evaluation-only and are never exposed to PORTER or the competing policies. We do not separately evaluate robustness to parser errors or omitted requirements; if a necessary requirement is absent from the supplied set, PORTER may undervalue the corresponding

task support. Requirement generation is therefore treated as an external interface rather than part of the PORTER residency objective. For reproducibility, the fixed Tier 2 parser prompt is:

> **System:** Decompose a robot task into the smallest set of independently retrievable object requirements for a 3D scene graph. Return JSON only, with schema `{"requirements":[{"text":str,"weight":number}]}`. Use short concrete noun phrases. Include every object or destination whose presence is necessary to perform the task. Do not emit actions, synonyms, explanations, or duplicate phrases. Preserve multiple requirements when the task needs multiple objects. If the task needs exactly one object, return exactly one requirement; never invent a second requirement to increase the list length.
> **User:** `Task:` {task_text}

Support and ISE configuration. Objects and requirements are encoded in the same OpenCLIP ViT-H/14 vision–language space using 1024-dimensional normalized embeddings. Support is constructed according to Eq. 1, and residency is ranked by the marginal ISE per byte defined in Eq. 9. The default configuration used throughout the experiments is summarized in Table 5.

Table 5: Default PORTER configuration used in the main experiments.

Component	Setting
Vision–language encoder	OpenCLIP ViT-H/14
Embedding dimension	$d = 1024$
Requirement weights	$\omega_r = 1/\lvert\mathcal{R}_q\rvert$
Similarity threshold	$\alpha = 0.2$
Candidate supporters	$K = 5$ per requirement
Anchor support scale	$\eta = 0$
ISE stabilizer	$\epsilon_{\mathrm{ISE}} = 10^{-6}$

A.3.2 Upstream 3DSG Baselines

PORTER does not alter the construction procedure of the upstream 3DSG. We use the released implementations and recommended configurations of ConceptGraphs (Gu et al., 2024), ReasoningGraph (Puigjaner et al., 2026), Clio (Maggio et al., 2024), and DAAAM (Gorlo et al., 2026), adapting only their input interfaces where required by JITOMA-Bench. Within each paired comparison, the native and PORTER-augmented variants share the same perceptual input, task sequence, and builder configuration; they differ only in payload residency.

For the controlled Tier 3 experiments, we use a single frozen ConceptGraphs reconstruction for every residency policy, eliminating scene-construction variation from the comparison. The resulting maps contain 229, 429, and 922 objects for Cubicle, Apartment, and Office, respectively. Builder-specific adapters retain each method's native object representation rather than converting all methods into a common simplified graph. Complete builder configurations will be released with the code.

A.3.3 Residency Policy Configuration

Common protocol. The Tier 3 policy comparison fixes the upstream ConceptGraphs representation and varies only the order in which payloads are offloaded. Every method receives the same object catalog, payload sizes, task annotations, and vision–language embeddings, and each task begins from the identical Keep-All state. After every removal, evaluation uses only the payload-bearing objects that remain locally resident.

Random and Largest-First. Random removes objects according to a uniformly sampled permutation; we average results over five seeds. Largest-First removes objects in decreasing order of payload size b_v, representing a memory-driven policy without task semantics.

CLIP/Byte. The complete task instruction is embedded as $\hat{\mathbf{g}}_q$, and each object is scored by

$$\phi_v^{\mathrm{clip}} = \frac{\hat{\mathbf{f}}_v^{\mathsf{T}} \hat{\mathbf{g}}_q}{b_v}. \tag{39}$$

Objects with the lowest scores are removed first. CLIP/Byte therefore combines direct task relevance with payload cost but does not explicitly reason over functional requirements.

Requirement/Byte. We first form a weighted task embedding from the individual requirement embeddings,

$$\bar{\mathbf{g}}_q = \sum_{r \in \mathcal{R}_q} \omega_r \hat{\mathbf{g}}_r, \qquad \phi_v^{\mathrm{req}} = \frac{\hat{\mathbf{f}}_v^{\mathsf{T}} \bar{\mathbf{g}}_q}{b_v}. \tag{40}$$

Objects with the lowest scores are again removed first. This baseline adds functional task decomposition but does not model substitute coverage, fragility, or state-dependent repricing.

PORTER and controlled ablations. PORTER removes the payload with the smallest current $\Delta_q^{\mathrm{ISE}}(v; S)/b_v$ and recomputes the marginal after every removal. The component ablations in Table 2 each modify one element of this policy: Noisy-OR aggregation, log-survival calibration, dynamic re-ranking, or byte normalization. Unless explicitly varied, all variants use the default parameters in Table 5. The support-set sensitivity study varies only $K \in \{1, 3, 5, 10, 15, 20\}$.

For a target byte-offloading ratio ρ_B, all methods are evaluated at the first whole-object state whose cumulative removed payload reaches that target,

$$t_\pi(\rho_B) = \min \left\{ t : \frac{B_{\mathrm{off}}^\pi(t)}{B(\mathcal{V})} \geq \rho_B \right\}. \tag{41}$$

Payloads are never fractionally removed; consequently, the realized ratio may slightly exceed the target by one object, and the same rule is applied to every policy.

A.3.4 Hardware and Software Environment

Experiments are conducted on a server equipped with AMD EPYC 7542 CPUs, 1 TiB of host memory, and NVIDIA RTX A6000 GPUs. Upstream 3DSG construction uses GPU acceleration where required by the corresponding builder, whereas the PORTER residency core runs on CPU.

A.4 Hyperparameter Sensitivity

A.4.1 Support Set Size K

In Eq. 1, K controls the maximum number of candidate supporters retained for each task requirement. Small K yields a sparser object–requirement support graph but may omit useful substitutes, whereas larger K admits more alternative support at increased computation. We follow the same Tier 3 progressive-compression protocol as Sec. 4.3 and vary $K \in \{1, 3, 5, 10, 15, 20\}$ while keeping all other settings fixed. We report relative mR@3 together with Overall and Tail nAUC, using the same definitions as in Sec. 4.5. The main experiments use $K = 5$.

Table 6: Sensitivity of PORTER to the number of candidate supporters K on JITOMA-Bench Tier 3. Overall nAUC covers the complete compression and Tail nAUC covers $[95\%, 100\%]$. All remaining columns report mR@3 relative to Keep-All. † denotes the default setting used in the experiments.

	Curve nAUC (%) ↑		Relative mRecall@3 at ρ_B (%) ↑				
K	Overall	Tail	95%	97%	99%	99.5%	99.9%
1	61.19	35.85	39.34	39.34	36.07	26.23	0.00
3	98.41	72.84	85.25	85.25	54.10	50.82	31.15
5†	98.30	72.03	85.25	81.97	59.02	50.82	31.15
10	98.26	73.14	85.25	85.25	63.93	60.66	31.15
15	98.22	73.27	85.25	85.25	63.93	50.82	9.84
20	98.23	73.49	85.25	90.16	63.93	50.82	9.84

Results. Table 6 reports the sensitivity of PORTER to K. With $K = 1$, Overall and Tail nAUC fall to 61.19% and 35.85%, respectively. With only one candidate per requirement, Noisy-OR has essentially no alternative support from which to infer replaceability, consistent with the large degradation observed when Noisy-OR is removed in Table 2.

Once multiple alternatives are retained, PORTER is largely insensitive to the precise value of K. Across $K \in \{3, 5, 10, 15, 20\}$, Overall nAUC varies by only 0.19 points and Tail nAUC by 1.46 points. No single setting dominates every extreme checkpoint because payloads are removed as whole objects rather than fractional bytes, so small changes in object ordering can produce discrete jumps near full offloading. This stability indicates that the observed gains are not driven by a particular top-K truncation once several alternative supporters are retained. It does not, however, rule out settings with unusually diffuse long-tail support, for which adaptive or threshold-based support sets may be preferable. We use $K = 5$ as a moderate fixed setting that preserves multiple alternative support while keeping the task-conditioned candidate set compact.

A.4.2 SIMILARITY THRESHOLD

In Eq. 1, α filters weak object–requirement matches before support aggregation. We follow the same Tier 3 progressive-compression protocol as Sec. 4.3, fix $K = 5$, and vary $\alpha \in \{0.10, 0.15, 0.20, 0.25, 0.30\}$ while keeping all other settings fixed. As in the main experiments, mR@3 is pooled across Tier 3 tasks at each checkpoint and normalized by the pooled Keep-All reference. We report Overall and Tail nAUC using the definitions in Sec. 4.5. The main experiments use $\alpha = 0.20$.

Table 7: **Sensitivity of PORTER to the similarity threshold α on JITOMA-Bench Tier 3.** Overall nAUC covers the complete compression curve and Tail nAUC covers $[95\%, 100\%]$. Remaining columns report mR@3 relative to Keep-All. † denotes the default setting.

α	Overall	Tail	95%	97%	99%	99.5%	99.9%
0.10	98.55	71.86	90.16	81.97	59.02	45.90	31.15
0.15	98.04	71.48	85.25	81.97	59.02	50.82	31.15
0.20†	98.30	72.03	85.25	81.97	59.02	50.82	31.15
0.25	96.58	60.05	77.05	59.02	50.82	50.82	31.15
0.30	62.75	30.93	34.43	34.43	29.51	24.59	0.00

Results. PORTER is stable over moderate similarity thresholds. For $\alpha \in \{0.10, 0.15, 0.20\}$, Overall nAUC varies by only 0.51 points and Tail nAUC by 0.55 points. At the default $\alpha = 0.20$, each requirement retains 4.78 nonzero supporters on average, with no zero-support requirements. Increasing α to 0.25 reduces this average to 3.29 and leaves $2/55$ requirements without positive support, while Tail nAUC drops to 60.05%. At $\alpha = 0.30$, $31/55$ requirements have zero support and Overall and Tail nAUC fall to 62.75% and 30.93%. These results indicate that PORTER is insensitive within a broad moderate range, but degrades once thresholding removes too much substitute structure. We use $\alpha = 0.20$ as a moderate default that filters weak matches while retaining sufficient alternative support for replaceability reasoning.

A.5 REAL-ROBOT EXPERIMENTAL DETAILS

A.5.1 PLATFORM AND SENSING

The real-robot experiment uses an AgileX mobile platform equipped with an Intel RealSense D435 RGB-D camera and two Livox Mid-360 LiDARs. The two LiDAR streams are fused in our FAST-LIO2 (Xu et al., 2022) pipeline to provide a common odometric trajectory and registered LiDAR geometry over the full traversal. This dual-LiDAR configuration improves spatial coverage around the robot and provides the geometric reference used throughout the experiment. Figure 5 visualizes the resulting odometry, registered LiDAR point cloud, and colorized scene reconstruction.

The experiment contains 1,183 RGB-D frames collected during a continuous traversal of the indoor environment shown in Fig. 3. ConceptGraphs receives the same image sequence and camera

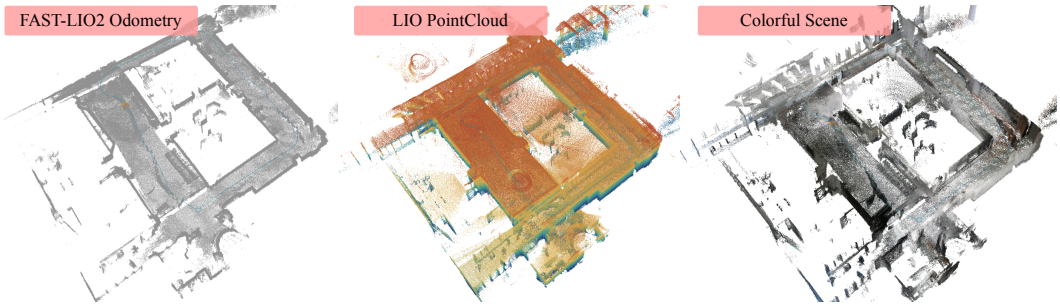

Figure 5: **Dual-LiDAR odometry and scene reconstruction.** The two Livox Mid-360 streams are fused through FAST-LIO2 (Xu et al., 2022) to estimate the robot trajectory and register LiDAR observations in a common frame. The resulting geometry provides a dense scene-level reference that is subsequently aligned with the RealSense observations.

trajectory in both the Vanilla and PORTER conditions. PORTER therefore changes only payload residency; the sensing trajectory and upstream perception input are held fixed.

A.5.2 Geometry and Ground-Truth Annotation

Although the LiDAR pipeline provides accurate global geometry and camera registration, ConceptGraphs operates on RGB-D observations. Raw RealSense depth is sufficiently noisy and incomplete to make precise 3D reconstruction and object-box annotation difficult, particularly for distant surfaces and object boundaries. We therefore use the registered LiDAR reconstruction as a higher-fidelity geometric reference for refining the D435 depth. Figure 6 compares representative RGB, raw depth, and LiDAR observations.

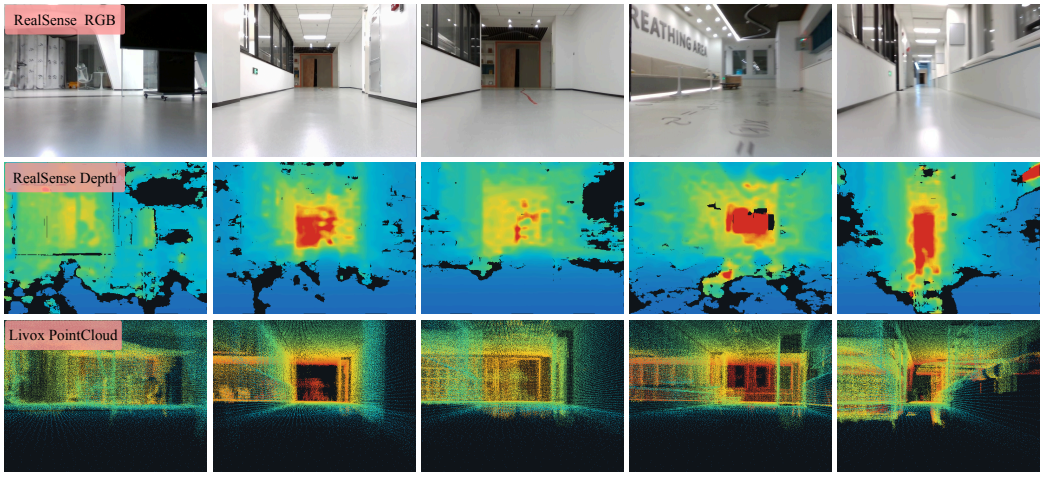

Figure 6: **RGB-D and LiDAR observations in the real-robot experiment.** Representative RealSense RGB images (top), raw D435 depth measurements (middle), and registered Livox point clouds (bottom). The raw depth contains substantial missing and noisy regions, motivating the use of LiDAR geometry as a higher-fidelity reference for depth refinement and 3D annotation.

Using the calibrated camera poses, the registered LiDAR reconstruction is reprojected into each camera view to obtain LiDAR-refined depth aligned with the RGB observations. The refined RGB-D sequence is then used to construct a geometrically consistent ConceptGraphs representation.

Task-relevant ground truth is annotated at the object level. For each target, we jointly inspect its reconstructed 3D object and the associated historical RGB observations accumulated by ConceptGraphs, as illustrated in Fig. 7. Multiple views provide additional evidence for object identity when

a single observation is ambiguous, while the reconstructed 3D geometry determines the corresponding evaluation box.



Figure 7: **Task-object ground-truth annotation.** Example reconstructed 3D object (left) and associated RGB observations from multiple frames. Annotators use the accumulated views to verify object identity and the reconstructed geometry to assign its 3D ground-truth box.

The resulting boxes serve only as evaluation ground truth and are never provided to ConceptGraphs, PORTER, or the residency policies during operation. Task performance is evaluated with mR@3 using the same IoU thresholds $\{0.1, 0.2, 0.3\}$ as in the main benchmark experiments.

A.5.3 Task-Switch Sequence

We evaluate four consecutive tasks over the same continuously growing scene memory. Their temporal boundaries and required objects are summarized in Table 8. The instructions are:

T1: Check whether the television screen is black, then find a chair for a guest to sit on.
T2: Grasp the door handle and open the door.
T3: Pick up the table-like folding camping cot and move it away from the wooden box.
T4: Check again whether the television screen is black, then close the door.

The repeated television and door requirements test whether scene knowledge useful to an earlier task can be recovered after its payload leaves local residency.

Table 8: Task sequence used in the real-robot experiment.

Task	Frames	Required objects
T1	0–300	television, chair
T2	305–600	door / door handle
T3	610–900	camping cot, wooden box
T4	910–1180	television, door

A.5.4 Edge–Cloud Residency and Transfer

The robot and remote server maintain separate local and cloud payload stores. The remote server establishes an SSH reverse tunnel to the robot, through which payloads are transferred using HTTP PUT and GET requests. Before transfer, payloads are gzip-compressed and contain the builder-specific object state, including the object point cloud and historical observation crops.

Transfers are executed concurrently with mapping. Consequently, the mapping TPF reported in Sec. 4.4 measures the online mapping path rather than serializing communication into the per-frame latency. PORTER accounts for payload volume through its marginal-ISE-per-byte criterion. Table 9 summarizes the observed Push/Pull activity and the resulting system-level gains.

Table 9: **Real-robot residency and system performance.** (a) Push/Pull activity and gzip-compressed payload volume across task switches. (b) Comparison of vanilla ConceptGraphs (CG) and ConceptGraphs+PORTER.

(a) Edge–cloud transfer

Switch	Push	Pull	Upload (MB)	Download (MB)
T1	18	0	1.42	0.00
T2	131	0	166.08	0.00
T3	203	4	189.14	3.72
T4	186	9	182.22	19.23
Total	538	13	538.86	22.95

(b) System summary

Metric	CG	+PORTER
Frames	1183	1183
Avg. local objs.	278.58	114.07
Peak local objs.	540	224
Final local objs.	531	224
TPF (s/frame)	6.276	1.863
mR@3	75.0%	75.0%

Transfer latency and connectivity. Push and Pull have asymmetric roles in the online system. Push transfers move payloads that are no longer required by the current task and therefore execute asynchronously off the mapping-critical path. Pull transfers are task-critical, since newly required cloud-resident payloads must be restored before they can be used. In our real-robot deployment, Push traffic is substantially larger: the four task switches upload 538.86 MB in total, whereas only 22.95 MB is downloaded across 13 Pulls.

Table 10 reports the measured end-to-end transfer times under the deployment network. The large Pushes at T2–T4 transfer 166.08–189.14 MB in 9.12–11.02 s while mapping continues concurrently. In contrast, the task-critical Pulls are much smaller: T3 restores 3.72 MB in 0.76 s and T4 restores 19.23 MB in 1.24 s. Thus, the large communication volume is dominated by asynchronous eviction, while the synchronous restoration path transfers only the subset required by the new task.

Table 10: **Measured edge–cloud transfer latency in the real-robot deployment.** Times are end-to-end measurements under the deployment network.

Switch	Upload (MB)	Push (s)	Download (MB)	Pull (s)
T1	1.42	0.54	0.00	–
T2	166.08	9.12	0.00	–
T3	189.14	11.02	3.72	0.76
T4	182.22	10.06	19.23	1.24

These measurements characterize our deployment network rather than a network-independent latency guarantee. Physical transfer time depends on payload size, available bandwidth, RTT, and contention. Under intermittent connectivity, locally resident payloads and all lightweight anchors remain available, but a cloud-resident payload cannot be restored until connectivity recovers. PORTER therefore preserves logical scene memory across a network outage, while immediate access to offloaded payloads remains subject to the availability of the backing store.

The nonzero Pull operations in T3 and T4 are particularly important: they show that cloud-resident payloads remain reusable across task switches and can be restored when their task value rises again. The experiment therefore exercises the full residency cycle—local retention, Push, persistent cloud storage, and subsequent Pull—rather than treating offloading as irreversible deletion.

Switching cost. PORTER deliberately optimizes task-conditioned information retention rather than jointly optimizing residency and network migration. The target residency set is therefore independent of the current residency; the latter determines only the Push/Pull needed to realize that target. This separation avoids coupling ISE to deployment-specific bandwidth, RTT, and asymmetric transfer costs. Although our experiments invoke PORTER at task switches, the method does not require a residency update at every transition: its invocation policy is external to ISE and can

be adapted to task duration, memory pressure, network conditions, or application-specific batching. Rapidly alternating tasks may therefore use a coarser residency schedule, while migration-aware hysteresis or switching penalties can be added at the execution layer without changing the ISE criterion.

A.5.5 Real-Robot Summary

As shown in Table 9, PORTER reduces average and peak local residency by 59.1% and 58.5%, respectively, while preserving the same 75.0% mR@3. The smaller working set also reduces mapping TPF from 6.276 s/frame to 1.863 s/frame, a 70.3% reduction, demonstrating a system-level benefit beyond memory footprint alone.

A.6 Comparison with Redundancy-Aware Residency Baselines

The main progressive-compression experiment focuses on a diagnostic progression from task-agnostic removal to relevance- and requirement-aware policies. We further compare PORTER with two stronger set-selection baselines that explicitly model redundancy: maximal marginal relevance (MMR) and task-weighted facility location. All methods use the same frozen ConceptGraphs representation, oracle task requirements, object embeddings, and payload sizes, and are evaluated under the same Tier 3 progressive-compression protocol. At each requested payload-byte offloading ratio, we evaluate the first whole-object state whose cumulative removed payload reaches that ratio, without fractional-object interpolation.

A requirement-aware submodular coverage objective is already evaluated as the *w/o Log-Information Loss* ablation in Table 2. Under the same dynamic removal rule, this ablation is equivalent to Submodular Coverage/Byte, so we do not duplicate it here. Instead, the following baselines provide complementary tests of semantic diversity and set-level representativeness.

MMR. We adapt maximal marginal relevance to object residency using the same object and requirement embeddings as the other Tier 3 policies. Let $\mathbf{v}_i$ and $\mathbf{q}_r$ denote individually ℓ_2-normalized object and requirement vision–language embeddings. We define dense task relevance and retained-set redundancy as

$$R(i) = \frac{1}{M}\sum_{r=1}^{M} \mathbf{v}_i^\top \mathbf{q}_r, \qquad D(i,S) = \begin{cases} 0, & S = \emptyset, \\ \max_{j\in S} \mathbf{v}_i^\top \mathbf{v}_j, & S \neq \emptyset. \end{cases} \tag{42}$$

Starting from an empty retained set, MMR repeatedly selects

$$i^\star = \arg\max_{i\notin S} \left[\lambda R(i) - (1-\lambda) D(i,S)\right], \tag{43}$$

with a fixed $\lambda = 0.5$ for all tasks. Both terms use raw signed cosine similarities, without clipping, support thresholding, Top-K pruning, or other PORTER-specific calibration. Ties are resolved by canonical object ID. We construct the complete forward retention ordering and reverse it to obtain the offloading sequence. MMR scores are therefore updated during forward selection, as in the standard formulation, rather than recomputed as deletion marginals during offloading. We do not byte-normalize the MMR objective; payload sizes enter only through the common byte-offloading checkpoints used for evaluation.

Task-weighted Facility-Location/Byte. We also evaluate a redundancy-aware representativeness objective that does not use PORTER's Noisy-OR aggregation. Each object u receives the task-conditioned weight

$$w_u = \sum_{r\in\mathcal{R}_q} \omega_r a_{ur}^F, \tag{44}$$

and object–object similarity is defined in the shared vision–language space as

$$k(u,v) = \max\left(0, \hat{\mathbf{f}}_u^\top \hat{\mathbf{f}}_v\right). \tag{45}$$

The retained-set utility is

$$F_{\mathrm{FL}}(S) = \sum_{u\in\mathcal{V}} w_u \max_{v\in S} k(u,v), \tag{46}$$

Table 11: **Comparison with redundancy-aware residency baselines on JITOMA-Bench Tier 3.** Entries report pooled relative mR@3 at matched cumulative payload-byte offloading ratios. Requirement-aware Submodular Coverage/Byte is identical to the w/o Log-Information Loss ablation in Table 2 and is not duplicated.

Method	85%	90%	95%	97%	99%	99.5%	99.9%
MMR	77.05	77.05	62.30	52.46	34.43	19.67	0.00
Task-weighted Facility-Location/Byte	95.08	90.16	72.13	72.13	**59.02**	36.07	0.00
PORTER	**100.00**	**100.00**	**85.25**	**81.97**	**59.02**	**50.82**	**31.15**

where the maximum is zero for $S = \emptyset$. At each removal step, the policy selects

$$v^\star = \arg\min_{v \in S} \frac{F_{\mathrm{FL}}(S) - F_{\mathrm{FL}}(S \setminus \{v\})}{b_v}, \tag{47}$$

and recomputes the marginal after every removal. Facility Location therefore maintains a compact set representative of task-relevant objects while discounting semantically redundant representatives, but it does not explicitly preserve coverage separately for each functional requirement.

Results. Table 11 shows that PORTER's advantage is not limited to comparisons with independent relevance heuristics. MMR explicitly trades task relevance against semantic redundancy, while Facility Location maintains a representative task-relevant subset. Both remain substantially more robust than simple relevance-based removal as compression increases, but the separation from PORTER grows once the resident set becomes scarce. At 85% offloading, PORTER retains 100% of Keep-All mR@3, compared with 95.08% for Facility Location and 77.05% for MMR; at 95%, the corresponding values are 85.25%, 72.13%, and 62.30%.

Together with the w/o Log-Information Loss ablation in Table 2, these comparisons separate several forms of redundancy reasoning. MMR promotes generic semantic diversity, Facility Location preserves set-level representativeness, and linear requirement coverage explicitly models functional substitutes. PORTER further weights coverage loss by the fragility of the surviving requirement support. This difference is most visible in the compression tail: at 99.5% offloading, PORTER retains 50.82% of Keep-All mR@3, compared with 40.98% for linear requirement coverage, 36.07% for Facility Location, and 19.67% for MMR. At 99.9%, PORTER retains 31.15%, while MMR and Facility Location reach zero. These results are consistent with redundancy modeling explaining much of the robustness at moderate compression, while fragility-aware requirement protection becomes increasingly useful after the easily removable redundancy has been exhausted.

A.7 Case Analysis: Residency And Task-Aligned Abstraction In CLIO

PORTER primarily serves as a residency manager, reducing the edge-resident working set while preserving downstream task capability as shown in Figure 2. Beyond this primary memory benefit, reshaping the local working set can also reduce the interference faced by downstream object retrieval. We use Clio (Maggio et al., 2024), a task-driven 3DSG builder, to make this interaction particularly explicit: changes in local residency are directly reflected in the set and composition of object candidates exposed to the task. We illustrate two representative outcomes below—one in which PORTER recovers an initially failed object retrieval, and another in which an already-correct retrieval is preserved under substantial compression.

A.7.1 Case 1: find spice bottles

Figure 8 shows the "find spice bottles" task in the Apartment scene. In Clio, 511 local segments are clustered into 16 objects which are then ranked by their CLIP similarity to the task query. The highest-scoring object is a compact, semantically plausible distractor, but lies outside the ground truth, yielding an IoU of 0. Meanwhile, the actual target is buried in a noisy 14-segment cluster and ranks only seventh with a CLIP similarity of 0.2156. PORTER changes both sides of this competition. It reduces the local working set to 65 segments and 8 objects, pushing out part of the previously higher-ranked distracting evidence. At the same time, removing noisy segments allows the target to be re-clustered into a much tighter 2-segment object. Its CLIP similarity increases to

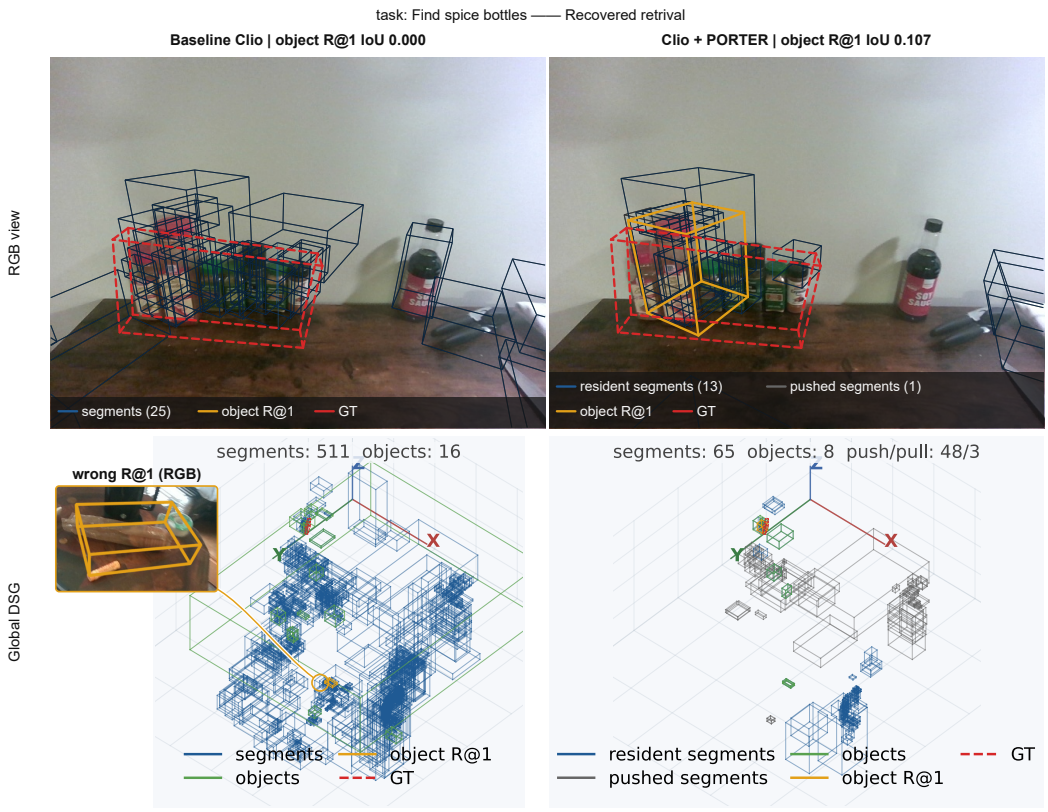

Figure 8: **Visualization of the find spice bottles case.** Clio first organizes the scene segments (blue box) into object abstractions (green box) using its task-driven clustering, and retrieves the object with the highest CLIP similarity to the task query, highlighted in yellow; the red dashed box denotes the ground truth. PORTER does not directly manage objects, but instead controls the residency of their underlying segments by push/pull operations: resident segments and pulled segments remain blue, while pushed segments between task injection and query are shown in gray, and an object becomes locally unavailable when its constituent segments are offloaded. The top row shows a local RGB view of these structures, whereas the bottom row shows the corresponding global 3DSG context. The inset on the lower left gives the RGB appearance and global location of the incorrect Top-1 object in Clio baseline. Segments/objects counts report the complete query-time 3DSG state, and push/pull records the segment-payload transfers performed between task injection and query.

0.2377, promoting it from rank 7 to the highest-similarity Top-1 retrieval, with an IoU of 0.107. This case therefore illustrates PORTER's retrieval-side denoising effect: a smaller local working set suppresses distracting evidence while yielding a cleaner representation of the true target.

A.7.2 Case 2: clean backpacks

Figure 9 shows the complementary "clean backpacks" case. Clio already retrieves the target correctly, with an IoU of 0.408. PORTER then reduces the local working set from 344 to only 36 segments, while preserving all six object-level abstractions. The retrieved object remains aligned with the same backpack region, with an IoU of 0.384. In this case, PORTER removes a large amount of redundant local evidence while preserving the object representation that is already useful for retrieval.

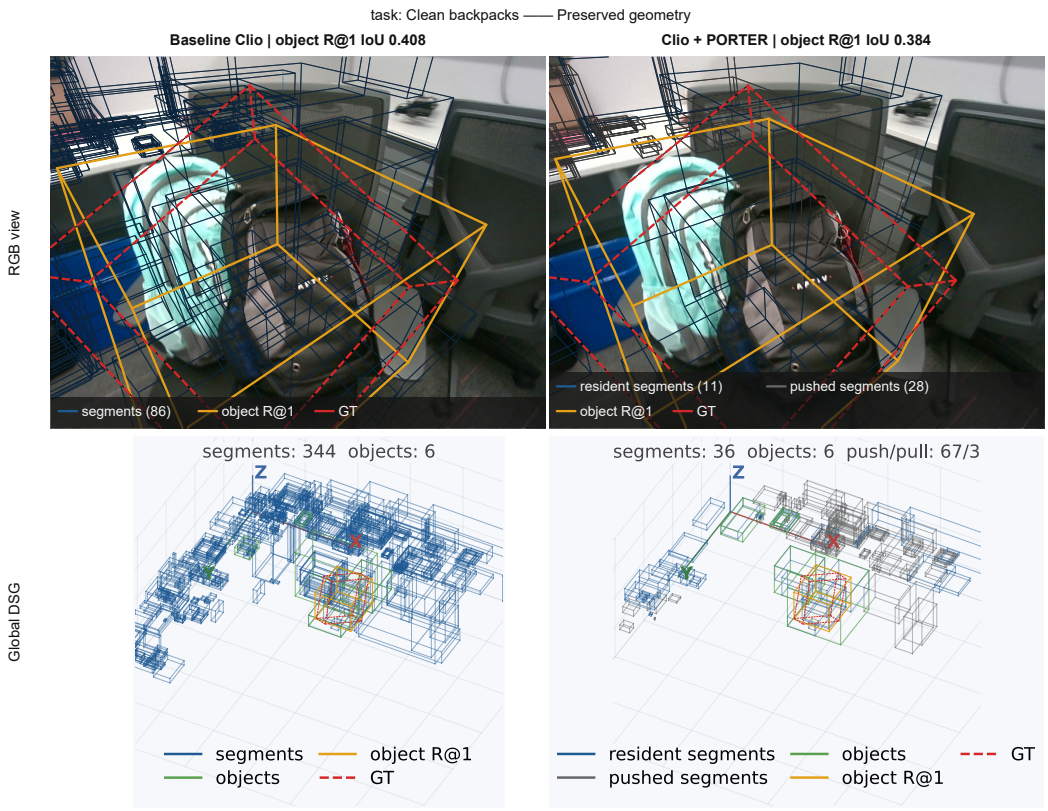

Figure 9: **Visualization of the clean backpacks case.** Clio clusters the blue scene segments into green object abstractions with its task-driven Information Bottleneck (IB) objective, and the yellow box marks the object with the highest CLIP similarity to the query; the red dashed box shows the ground truth. PORTER operates only on segment residency by push/pull operations: locally retained segments and pulled segments remain blue, pushed segments between task injection and query are shown in gray. The RGB panels provide a local view around the queried target, while the lower panels show the corresponding global 3DSG and the overall reduction in resident scene content. Segment/object counts are measured over the complete query-time 3DSG, and push/pull denotes the segment-payload transfers between task injection and query.