STRUCTURED MEMORY LINKING OBJECTS AND CONCEPTS FOR EMBODIED MANIPULATION AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon embodied manipulation requires agents to reuse interaction knowledge while tracking changing objects and scenes. A past successful action may become inapplicable when an object's state changes, and its motion parameters may not transfer to another instance with different geometry. Reusing manipulation experience therefore requires memory to connect object-part geometry with action applicability and execution parameters, while tracking changing object states and scene relations. These requirements motivate a part-based representation with reusable structure, instance-specific geometric parameters, and interfaces to manipulation functions. We propose **Object–Concept Linked Memory (OCLM)**, which uses object code representations to link grounded objects and reusable part templates to scene, transition, and skill memories. At runtime, the agent retrieves relevant objects, scene states, action outcomes, and skills to check applicability and select or parameterize template-grounded or policy-grounded execution. Observations and execution feedback update the associated memories, retaining reusable structural knowledge alongside evolving interaction history. Experiments on memory-dependent manipulation, articulated-object transfer, and real-world tasks show improvements in task success, retrieval accuracy, object re-identification, and cross-object skill transfer over the evaluated unstructured and embedding-based memory baselines.

1 INTRODUCTION

Embodied agents are expected to perform long-horizon robotic manipulation tasks in dynamic physical environments (Fang et al., 2019; Gervet et al., 2023; Fung et al., 2025). Such tasks require agents not only to perceive the current scene and execute actions, but also to remember persistent objects, track changing states, reuse previous interaction knowledge, and recover from execution failures. Without an explicit memory mechanism, an agent may repeatedly rediscover the same object, lose track of state changes, or fail to exploit prior experience when facing structurally similar objects and tasks (Hu et al., 2025).

Recent agent memory systems have explored storing interaction histories, scene observations, textual summaries, visual embeddings, or episodic traces. While these memories provide useful context, they are often insufficient for robotic manipulation. Unstructured memories make it difficult to retrieve task-relevant object and state information efficiently. Embedding-based memories (Zhang et al., 2025b; Wang et al., 2025a) may capture similarity, but often lack explicit access to object parts, affordances, and action effects. Scene-level (Wang et al., 2025b) or episode-level memories (Zhu et al., 2024) usually describe what happened, but do not directly represent how object structures support manipulation or how actions change object states.

Current approaches to long-horizon embodied tasks often follow two complementary routes: end-to-end VLA policies (Shi et al., 2026; Li et al., 2026) provide strong visuomotor execution but keep task reasoning and memory largely implicit, while agentic systems (Anwar et al., 2025; Lei et al., 2026; Tan et al., 2025) make planning and tool use explicit but often depend on manually specified skills or task-specific robot APIs. However, neither line provides an explicit memory interface that is simultaneously language-addressable, object-structured, and execution-grounded. As a result, the agent may retrieve semantically relevant memories that are physically inapplicable, or select skills without verifying object states and part-level constraints.

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.

Meeting these requirements calls for a representation that makes object structure and manipulation interfaces explicit. Code provides a common medium for expressing such information: Real2Code (Zhao et al., 2025) describes object parts and joints through code, while Code as Policies (Liang et al., 2023) expresses manipulation procedures as programs that compose perception and control APIs. Analytic concepts (Sun et al., 2024; 2025) similarly represent objects through code-defined, parametric part templates, connecting reusable structures with geometric parameters and associated manipulation functions. This object-centered representation provides a basis for organizing manipulation experience across interactions, where shared structural knowledge must be retained alongside changing object states. We propose **Object–Concept Linked Memory** (**OCLM**), which links persistent object instances to reusable part concepts and connects them to scene states, action transitions, and executable skills. These links organize instance-specific interaction history around reusable templates, allowing past experience to inform subsequent manipulation as objects and scenes evolve.

At runtime, the agent queries memory using the current observation and subtask, retrieves relevant object and scene states, and associates them with past action outcomes and candidate skills. It then checks skill applicability and either binds grounded affordances and geometric parameters to manipulation functions or invokes a learned policy executor. When no usable memory is available, the agent re-analyzes the observation and calls grounding tools. Execution feedback updates the corresponding object, scene, transition, and skill records, so subsequent decisions can use the revised state and accumulated experience. This closed loop makes the organization of memory directly relevant to planning and execution.

Our contributions are threefold:

- We introduce a linked memory organization for embodied manipulation that connects persistent objects and scene states with action transitions and executable skills, while separating stable structural priors from dynamic interaction experience.
- We develop a memory-guided execution loop that combines structured retrieval, state-dependent applicability checking, and feedback-driven updates, supporting both template-grounded manipulation functions and learned policy executors.
- We evaluate the framework on memory-dependent manipulation, articulated-object transfer, and real-world tasks, comparing task completion and memory retrieval performance with policy-based, retrieval-based, and structured-memory baselines.

2 Related Work

2.1 Embodied Agentic Systems

Recent embodied AI systems increasingly adopt agentic frameworks for long-horizon and language-conditioned manipulation tasks (Salimpour et al., 2025; Liang et al., 2025). Hierarchical embodied agents typically decompose high-level goals into executable subtasks or reusable skills, combining planning modules with low-level robot controllers. Early systems such as SayCan (Ahn et al., 2022) and Code as Policies (Liang et al., 2023) leverage language models to select skills, invoke APIs, or synthesize executable programs for robot control. More recent approaches integrate stronger visual-language-action (VLA) (Brohan et al., 2022; Kim et al., 2024) policies into agentic execution loops, enabling improved planning, replanning, and skill composition for long-horizon manipulation (Yang et al., 2025; Lei et al., 2026). Recent works further explore modular policy composition and skill-centric execution. ThinkAct (Huang et al., 2026) combines reasoning modules with low-level executors for closed-loop embodied execution, while LiLo-VLA (Yang et al., 2026) demonstrates that modular object-centric policies can improve compositional manipulation through dynamic replanning and skill reuse. These works highlight the importance of hierarchical reasoning and modular execution in embodied agents. However, existing systems primarily focus on planning and policy orchestration, while memory is often limited to short interaction histories, latent context windows, or unstructured retrieval mechanisms.

2.2 MEMORY FOR AGENTS

Recent advances in agent memory have established memory as a fundamental component of LLM-based agents, enabling long-horizon reasoning, continual adaptation, and environment interaction. Existing studies mainly categorize memory into token-level, parametric, and latent forms, further supporting factual, experiential, and working memory functionalities (Hu et al., 2025). Most current systems adopt token-level memory, where dialogue histories, trajectories, summaries, or experience buffers are explicitly stored and retrieved, as exemplified by frameworks (Packer et al., 2023; Chhikara et al., 2025; Zhang et al., 2025a).

However, most existing memory systems target language/web agents or simplified simulations, and do not directly meet the needs of embodied robotics (Zheng et al., 2025; Ye et al., 2025; Fang et al., 2025). In real-world interaction, naively storing multimodal trajectories incurs high storage and retrieval overhead under real-time constraints. Moreover, many representations remain semantic-centric and weakly connected to grounded affordances and executable manipulation skills.

As a result, existing approaches often memorize task-specific experiences but struggle to generalize manipulation knowledge across diverse objects and environments. These limitations motivate our object–concept linked memory, which aims to unify object semantics, environment understanding, and manipulation skills into a compact and transferable memory representation for embodied agents.

3 PRELIMINARIES

Language-conditioned embodied manipulation. We study long-horizon language-conditioned embodied manipulation, where an agent receives an instruction L and a sequence of visual observations $\{o_t\}$. We model execution hierarchically. At the task level, L is decomposed into subtasks $\mathbf{l} = \{l_1, \ldots, l_N\}$; at the execution level, each subtask l_i is realized by a sequence of low-level robot actions $\boldsymbol{a}_i = \{a_{i,1}, \ldots, a_{i,K_i}\}$. The full trajectory is $\boldsymbol{a} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N\}$, where each $a_{i,j}$ denotes an executable robot command, such as an end-effector pose, trajectory command, or controller input.

In our setting, subtask execution is conditioned on an external memory $\mathcal{M}$:

$$\boldsymbol{a}_i = \pi(L, o_t, l_i, \mathcal{M}),$$

where $\mathcal{M}$ provides persistent object identities, part-level structure, evolving scene state, action-effect knowledge, and executable skills beyond the current observation.

Object Code Representation. We build our object code representation on the parametric part-template formulation of analytic concepts (Sun et al., 2024; 2025). Here, a *concept* denotes a reusable part-template type with geometric parameters and manipulation interfaces; an *object* is a scene instance whose parts instantiate these types with observed geometry and poses. This template–instance distinction supplies the representational basis for OCLM.

Formally, an object O is represented as

$$O = \mathcal{O}(c, \{P_i\}_{i=1}^{M}),$$

where c denotes the object category and P_i denotes the i-th semantic part. Each part is associated with a template type τ_i and instantiated as

$$P_i = \mathcal{P}(\tau_i, \Theta_i, T_i, A_i, \mathcal{F}_{\tau_i}),$$

where τ_i links the part to a reusable template in the template library, Θ_i denotes the estimated geometric parameters, T_i denotes the grounded 6D pose, A_i denotes part-level affordances, and $\mathcal{F}_{\tau_i}$ denotes the manipulation functions supported by this template type.

By *code representation*, we mean programmatic object and part records that expose grounded quantities as fields and template-supported functions as callable interfaces (Appendix A.1). The type τ_i links each part instance to its reusable concept. Building on this representation, OCLM adds persistent identities and interaction histories, with links to scene, transition, and skill records.

4 METHOD

We propose Object–Concept Linked Memory (OCLM), a framework for long-horizon embodied manipulation. As shown in Figure 1, the framework maintains a structured memory $\mathcal{M}$ that connects

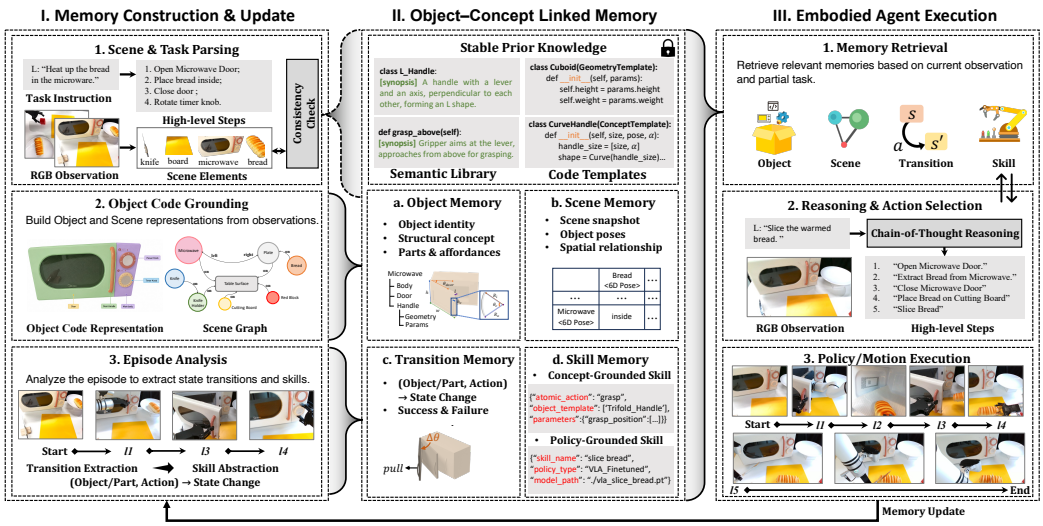

Figure 1: Overview of the proposed object–concept linked memory system. Stable semantic and template libraries provide reusable priors, while dynamic object, scene, transition, and skill memories store grounded instances, evolving states, and action effects. At runtime, the agent retrieves relevant entries to ground subtasks and select/parameterize executable skills, and then writes back observations and outcomes to continuously update memory.

object-centric concepts, scene states, action transitions, and executable skills. At a high level, the agent first grounds observations into object code representations, retrieves task-relevant memory entries for the current subtask, selects or parameterizes an executable skill, and then writes execution outcomes back to memory. This closed loop allows the agent to maintain persistent object identities, reason about state changes, and reuse manipulation knowledge across related objects and tasks.

4.1 Object–Concept Linked Memory

The memory $\mathcal{M}$ is organized around object code representations and consists of stable prior knowledge and dynamic experience memory :

$$\{\mathcal{M}^{sem}, \mathcal{M}^{temp}, \mathcal{M}^{obj}, \mathcal{M}^{scene}, \mathcal{M}^{trans}, \mathcal{M}^{skill}\}.$$

The stable components, $\mathcal{M}^{sem}$ and $\mathcal{M}^{temp}$, provide language-level concept descriptions and their parametric part templates, respectively. The dynamic components store grounded object instances, evolving scene states, action-induced transitions, and executable skills collected during interaction.

4.1.1 Stable Prior Knowledge

Stable prior knowledge contains two fixed libraries shared across episodes: a semantic library of template synopses and a template library of parametric part templates. Online observations and outcomes are written only to dynamic memory.

Semantic Library. The semantic library $\mathcal{M}^{sem}$ stores natural-language synopses for templates in $\mathcal{M}^{temp}$. Each entry is indexed by the same template type τ as the template library:

$$m_\tau^{sem} = (\tau, d_\tau, a_\tau, u_\tau),$$

where d_τ is a concise textual description of the template, a_τ describes its interaction affordances, and u_τ describes its intended functional use. For example, the synopsis of an `L_Handle` template describes it as a handle with a lever and an axis, where the lever supports grasping and the axis indicates the direction for pulling or rotation. These synopses provide a language-level interface for selecting template types during concept grounding, memory retrieval, and skill selection.

Template Library. The template library $\mathcal{M}^{temp}$ stores reusable part templates indexed by template type τ:

$$m_{\tau}^{temp} = (\tau, \Omega_{\tau}, \mathcal{F}_{\tau}),$$

where Ω_{τ} denotes the valid parameter space of the template, and $\mathcal{F}_{\tau}$ is the set of supported manipulation functions. During grounding, scene parsing modules, including object detection, part segmentation, pose estimation, and parameter estimation, bind a detected part to a template type τ_i and estimate scene-specific values $(\hat{\Theta}_i, \hat{T}_i)$, producing a grounded part concept P_i. The manipulation functions in $\mathcal{F}_{\tau}$ are implemented using the atomic primitives defined in Appendix C. Thus, templates provide reusable symbolic and geometric blueprints, while object memory stores their scene-specific executable instantiations.

4.1.2 Dynamic Experience Memory

Dynamic memory records grounded information accumulated during interaction. Unlike stable prior knowledge, dynamic memory is updated online as the agent observes the scene, executes actions, and verifies their outcomes.

Object Memory. Object memory $\mathcal{M}^{obj}$ maintains persistent object code representations grounded in the current environment:

$$m_k^{obj} = (id_k, c_k, \{P_{k,i}\}_{i=1}^{M_k}, T_k, s_k, h_k).$$

Here, T_k, s_k, and h_k denote the object's grounded 6D pose, current manipulation-relevant state (e.g., open/closed), and observation/interaction history, respectively. Each part $P_{k,i}$ instantiates a reusable template $\tau_{k,i}$ with grounded geometry, pose, affordances, and manipulation functions for retrieval and skill execution.

Scene Memory. Scene memory $\mathcal{M}^{scene}$ tracks the evolving environment state. We represent a scene snapshot as a graph

$$G_t = (V_t, E_t),$$

where nodes V_t correspond to object instances and edges E_t encode spatial, visibility, containment, support, and state relations. Scene memory enables multi-step planning by retaining relations not fully observable at the current timestep.

Transition Memory. Transition memory $\mathcal{M}^{trans}$ stores action-induced state changes over grounded objects and parts:

$$m^{trans} = (id_k, P_{k,i}, u, s_{\text{pre}}, s_{\text{post}}, r),$$

where id_k is the persistent target object, $P_{k,i}$ is the grounded target part, u is the executed skill or primitive action, s_{pre} and s_{post} are pre- and post-action states, and r is the execution outcome. This links action effects directly to object-memory instances for later precondition checking and effect prediction.

Skill Memory. Skill memory $\mathcal{M}^{skill}$ stores executable procedures:

$$m_u^{skill} = (u, type, \tau, target, pre, \Phi, \pi_{\text{exec}}, \eta),$$

where u identifies the skill, $type$ denotes template- or policy-grounded execution, τ is the associated template type when applicable, $target$ specifies the required object/part schema, pre denotes applicability conditions, Φ contains bound parameters, π_{exec} is the execution backend, and η stores execution statistics. Template-grounded skills bind $f \in \mathcal{F}_{\tau}$ to a grounded part $P_{k,i}$, while policy-grounded skills invoke learned policies for behaviors that are hard to express with explicit templates.

4.2 Memory Construction & Update

This module converts instructions, observations, and execution outcomes into memory updates. Given $(L, o_t, \mathcal{M}_t)$, it parses the task, grounds observed entities, resolves object identities, and updates object, scene, transition, and skill memories.

4.2.1 TASK AND SCENE PARSING

The first stage extracts a task-scene context from the instruction and the current observation. The instruction L is decomposed into high-level subtask candidates $\mathbf{l} = \{l_1, \ldots, l_N\}$. Meanwhile, the observation o_t is parsed to detect task-relevant objects, semantic parts, and environmental elements. The parser also extracts preliminary spatial, state, and functional relations, such as `on`, `inside`, `open`, or `graspable`. The output is a structured context

$$\mathcal{Q}_t = (\mathbf{l}, \mathcal{O}_t, \mathcal{R}_t),$$

where $\mathcal{O}_t$ denotes the set of observed entities and $\mathcal{R}_t$ denotes their preliminary relations.

The agent identifies object categories and semantic parts, then invokes the corresponding tools to obtain poses, geometric parameters, and object states. These tool outputs populate the grounded memory entries; category and part recognition by the agent is therefore distinguished from the tool-based estimation of physical quantities.

4.2.2 OBJECT AND SCENE GROUNDING

Observed entities are grounded into object code representations and used to update object and scene memory. For each detected object, the agent localizes semantic parts, matches them to templates in $\mathcal{M}^{temp}$, and instantiates a grounded object representation $\hat{O}_k$ with part parameters, poses, affordances, and candidate manipulation templates. Details of the tool-augmented perception and grounding pipeline are provided in Appendix A.1.

A key step is object identity resolution. Given a newly grounded object $\hat{O}_k$, the agent matches it to existing object memory entries through a hierarchical filtering process. It first applies category and part-level consistency checks:

$$\mathcal{C}_k = \mathrm{Filter}_{cat,part}(\hat{O}_k, \mathcal{M}^{obj}).$$

Entries that fail these checks are discarded. For each remaining candidate $m_j^{obj} \in \mathcal{C}_k$, the agent computes

$$S_{k,j} = \lambda_g S_{\mathrm{geom}} + \lambda_T S_{\mathrm{pose}} + \lambda_r S_{\mathrm{rel}}.$$

If the best score exceeds a threshold, the observation is merged with the corresponding persistent identity; otherwise, a new object entry is registered. After identity resolution, the agent updates the scene graph with persistent object nodes and observed spatial, visibility, containment, support, and state relations.

4.2.3 EPISODE-LEVEL TRANSITION AND SKILL

After executing a subtask or episode, the agent consolidates observed outcomes into transition and skill memories. Each transition records the executed action, pre- and post-action states, and outcome, linked to the persistent target object and grounded part.

Successful template-based executions are further abstracted into template-grounded skills by associating the successful manipulation function with its template type, target schema, bound parameters, and corresponding transitions. Learned policies, such as finetuned VLA models, are stored as policy-grounded skills with their input prompts, applicability conditions, parameters, and outcome statistics.

4.3 STRUCTURED MEMORY RETRIEVAL

Memory entries are indexed by object-centric and manipulation-aware keys, including object identities, part types, affordances, geometric signatures, scene relations, object-part-action tuples, applicability conditions, and statistics. Given (L, o_t, l_i), retrieval follows a coarse-to-fine process:

$$\begin{cases} \mathcal{C}_t = \mathrm{Retrieve}_{coarse}(L, o_t, l_i), \\ \mathcal{R}_t = \mathrm{Retrieve}_{fine}(\mathcal{C}_t, \mathcal{M}_t). \end{cases}$$

The coarse stage identifies candidate categories, templates, and manipulation contexts, while the fine stage retrieves object, scene, transition, and skill entries. Fine retrieval uses hard filters such as

category, part, and affordance consistency, followed by ranking based on geometry, pose, scene state, transition applicability, and skill statistics.

The agent formulates memory queries from the current instruction, observation, and subtask. When no usable entry is returned, it re-analyzes the current observation and invokes perception and grounding tools to construct the information needed for planning. This fallback draws on reusable templates, perception tools, and execution capabilities to support planning even when relevant experience has not yet been stored.

4.4 Memory-Grounded Reasoning and Execution

After retrieval, the agent integrates the retrieved object, scene, transition, and skill entries with the current subtask to select an executable action plan. The reasoning process checks whether candidate skills are applicable under the current scene state and whether their predicted effects are consistent with the task goal.

For a candidate skill u, the agent first checks its precondition against the current scene state:

$$\text{Applicable}(u, G_t) = \mathbb{I}[pre(u) \subseteq state(G_t)].$$

If the skill is applicable, transition memory is used to estimate its expected effect:

$$\hat{s}_{t+1} = \text{Apply}(state(G_t), effect(u)).$$

This allows the agent to avoid redundant or inconsistent actions. For example, if the scene memory already records a microwave door as `open`, the agent should not select another `open-door` skill before placing the object inside.

Once a grounded subtask is determined, the agent selects an executable skill from R_t^{skill}. Template-grounded skills are instantiated from manipulation templates with object-specific poses, part parameters, and execution constraints, and are executed by a motion planner or controller. Structured memory supplies grounded affordances and manipulation functions to support this planning process. Policy-grounded skills invoke learned policies, such as finetuned VLA models, for complex or perception-driven behaviors that are difficult to represent with explicit templates.

The selected skill produces a low-level action sequence

$$\boldsymbol{a}_i = \{a_{i,1}, \ldots, a_{i,K_i}\},$$

which is executed in the environment. The observed outcome o_{t+1} is then compared with the predicted postcondition. If the outcome is verified, object states, scene relations, transitions, and skill statistics are updated accordingly. If execution fails or the observed state contradicts the prediction, the corresponding skill attempt is recorded as unsuccessful and the agent can re-query memory for an alternative skill. This closes the loop between structured retrieval, state-consistent reasoning, physical execution, and memory update. The overall closed-loop procedure is summarized in Algorithm 1.

5 Experiments

5.1 Memory-dependent Manipulation Results

This section answers: *Does memory improve agents on memory-dependent manipulation tasks?*

Benchmark. We evaluate on *RMBench* (Chen et al., 2026), following the official Mem-0 protocol with 100 rollout episodes per reported task. Ours(CS) and Ours(DP) use GPT-5.6 Sol (high) for high-level reasoning. Appendix B.1 details the evaluation protocol.

Baselines. We compare against representative baselines aligned with the table columns. **DP** denotes the standalone DP policy baseline. $\boldsymbol{\pi_{0.5}}$ (Black et al., 2025) is a reactive finetuned VLA that acts only on the current observation. **Mem-0** and **Mem-VLA** (Shi et al., 2026) are memory-oriented robotic VLA policies. **MemER** (Sridhar et al., 2026) follows a dual-system design with keyframe retrieval. **RoboEXP** (Jiang et al., 2024) is included as an additional structured-memory baseline based on an action-conditioned scene graph and interactive exploration. For our variants, **Ours(CS)** uses only Concept-Grounded Skills, while **Ours(DP)** uses the same DP model as the **DP** baseline for low-level skill execution. The difference is the addition of the OCLM-based agent, which performs memory-grounded subtask decomposition and skill selection using this shared executor.

Table 1: Success rate (SR, %) on RMBench. TMC denotes Task Memory Complexity, with levels $M(1)$ and $M(n)$ following RMBench. DP and Ours(DP) share the same DP execution model. *Total Avg* is the average across the reported tasks. * denotes manually implemented testing.

Task	TMC	DP	$\pi_{0.5}$	Mem-0*	Mem-VLA*	MemER	RoboEXP*	Ours(CS)	Ours(DP)
Rearrange Blocks	$M(1)$	0%	13%	89%	20%	24%	48%	**95%**	92%
Put Back Block	$M(1)$	0%	11%	90%	35%	42%	64%	88%	**92%**
Swap Blocks	$M(1)$	11%	24%	67%	42%	51%	24%	**73%**	70%
Swap T	$M(1)$	20%	15%	14%	55%	60%	60%	73%	**88%**
Battery Try	$M(n)$	10%	16%	28%	12%	14%	12%	48%	**54%**
Blocks Ranking Try	$M(n)$	10%	6%	18%	0%	2%	10%	25%	**29%**
Cover Blocks	$M(n)$	0%	0%	68%	31%	25%	36%	68%	**69%**
Total Avg	–	7%	12%	53%	28%	31%	36%	67%	**71%**

Results. As shown in Table 1, Ours(CS) and Ours(DP) achieve average success rates of 67% and 71%, respectively, compared with 53% for the strongest baseline, Mem-0, and 36% for the structured-memory baseline, RoboEXP. With the DP executor held fixed, Ours(DP) improves average SR from 7% to 71%, supporting the contribution of memory-grounded planning and execution. Ours(CS) performs best on Rearrange Blocks and Swap Blocks, while Ours(DP) achieves the highest SR on the remaining reported tasks. Both execution variants outperform the strongest baseline in average SR, by 14 and 18 percentage points, respectively. These gains demonstrate the effectiveness of OCLM-based planning with both concept-grounded skills and policy-based execution.

5.2 Manipulation Knowledge Transfer

This section answers: *Does object–concept linked memory improve manipulation knowledge transfer across articulated objects that share similar part-level structures and affordances?*

Benchmark and task families. We evaluate on articulated object manipulation tasks from *PartNet-Mobility* (Xiang et al., 2020), following a commonly used generalization protocol (e.g., Where2Act (Mo et al., 2021)). The focus is manipulation knowledge transfer across articulated objects that share similar structure and affordances. The experimental details are provided in Appendix A.3. We evaluate two transfer settings: **cross-instance transfer**, which tests generalization to unseen instances within the same category (with varying geometry/scale/pose), and **cross-object transfer**, which tests transferring part-level manipulation knowledge across categories with similar articulation/affordances.

Baselines. We compare against representative MLLM-based manipulation baselines, **ManipLLM** (Li et al., 2024) and **A3VLM** (Huang et al., 2024), which use multimodal reasoning to infer affordances and actions from RGB-D observations and language inputs. We also include a **Semantic-memory** baseline that stores object categories and affordances but lacks explicit part-structured templates and manipulation knowledge. In contrast, our method augments MLLM reasoning with grounded object code representations and linked memory for retrieval and reuse.

Results. Table 2 summarizes manipulation knowledge transfer results under both transfer settings. We report success rate (SR, %) on unseen instances/categories, averaged within each category. Overall, our approach consistently improves SR across all categories. For *cross-instance transfer*, we outperform the strongest baseline (typically A3VLM), indicating better robustness to geometric and pose variations within a category. *Cross-object transfer* is more challenging for prior methods (e.g., A3VLM reaches at most 42.5 SR), while our method improves over the strongest baseline in each category: +8.1 on *Washing Machine*, +3.1 on *Kitchen Pot*, +10.6 on *Safe*, and +10.3 on *Table*. Notably, **Semantic-memory** lags far behind, suggesting that storing coarse category/affordance cues alone is insufficient; explicitly modeling reusable part-level templates and action effects is key to transferring manipulation knowledge across categories.

Table 2: Manipulation knowledge transfer: SR (%) on unseen instances/categories.

	Category	ManipLLM	A3VLM	Semantic-mem	Ours
Cross-instance	Microwave	41.5	49.7	20.4	**52.2**
	Storage Furniture	47.0	53.3	10.5	**60.3**
	Refrigerator	36.4	40.8	18.5	**45.9**
	Door	50.2	56.2	28.5	**60.0**
	Faucet	16.5	24.1	17.1	**32.4**
	TrashCan	15.0	18.2	15.0	**22.8**
Cross-object	Washing Machine	19.1	22.3	8.6	**30.4**
	Kitchen Pot	12.4	11.0	2.5	**15.5**
	Safe	37.0	40.0	10.5	**50.6**
	Table	39.1	42.5	32.4	**52.8**

Table 3: Real-world manipulation results. We compare our object–concept linked memory with a MemER-style keyframe retrieval baseline.

Task	Keyframe retrieval			Ours		
	SR ↑	RA ↑	RE ↓	SR ↑	RA ↑	RE ↓
Slice Bread	75%	–	0.34	80%	–	0.09
Search	40%	–	0.89	75%	–	0.14
Return	80%	100%	0.30	90%	100%	0.13
Cube to Cup	30%	30%	0.35	90%	100%	0.15
Replace Block	55%	75%	0.38	85%	95%	0.14
Avg.	56%	68%	0.45	**84%**	**98%**	**0.13**

5.3 Real-World Memory Evaluation

We evaluate memory capability on five real-world tabletop tasks, with 20 trials per task for each method. Detailed protocols and task descriptions are provided in Appendix A.4. These tasks require the agent to maintain object identity, track scene relations and object locations, remember searched regions, reason over state transitions, and retrieve reusable skills. We compare our method with a MemER-style keyframe retrieval baseline, which stores and retrieves visual keyframes but does not explicitly structure memory by object concepts, scene states, transitions, or skills. We report task success rate, retrieval accuracy, and retrieval efficiency.

Results. Table 3 shows that our method improves average task success from 56% to 84%, retrieval accuracy from 68% to 98%, and improves retrieval efficiency (RE), reducing average end-to-end query latency from 0.45 to 0.13. Explicit links among object identities, scene relations, transitions, and skills support task-relevant retrieval and state-dependent applicability checks beyond visual keyframe similarity.

5.4 Ablation Study

On RMBench, we remove individual histories or replace structured records while retaining execution capabilities. Full OCLM achieves the highest SR and RA (Table 4). Full OCLM gains 15/14 SR points over object/scene-history ablations and 7/5 over transition/skill-history ablations. Flat/generic records lower SR by 12/9 points and Re-ID by 13/7 points. These results highlight the value of persistent object and scene history, linked retrieval, and template-based representations for task success and identity matching.

Table 4: Memory ablation results (%).

Variant	SR ↑	RA ↑	Re-ID ↑
Full OCLM	71	95	85
w/o object history	56	76	–
w/o scene history	57	80	85
w/o transition history	64	88	85
w/o skill experience	66	89	85
Flat records	59	81	72
Generic object records	62	84	78

SR: task success; RA: joint object–skill retrieval accuracy; Re-ID: object re-identification accuracy (all %). –: not applicable without persistent identity.

Appendices B.1–B.3 provide protocols, LLM sensitivity, and a memory scalability evaluation.

6 Conclusion

We introduced Object–Concept Linked Memory (OCLM) for embodied manipulation agents, linking reusable object templates with scene states, action transitions, and executable skills. Experiments on memory-dependent manipulation, manipulation knowledge transfer, and real-world tasks demonstrate improved task success, retrieval accuracy, and skill transfer. Ablations highlight the contributions of persistent history and structured representations. Future work will extend the template library to deformable objects and explore learned retrieval and more robust perception.

AI USE STATEMENT

We used generative AI tools solely to assist with manuscript writing and language polishing. All AI-assisted text was reviewed and edited by the authors, who take full responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

To support reproducibility, we specify the memory schemas, retrieval procedure, and update rules in the method section, with the closed-loop execution procedure summarized in Algorithm 1. Appendix A.1 describes the perception tools, parameter-estimation architecture, and object code interfaces; Appendix A.2 details verification and conflicting-memory updates. The manipulation knowledge transfer and real-world evaluation protocols, including initialization, success criteria, and retrieval metrics, are described in Appendices A.3 and A.4. Appendix B.1 specifies the RMBench rollout protocol, seed handling, and memory-ablation interventions. Appendix B.2 identifies the evaluated reasoning models, and Appendix C defines the atomic manipulation primitives used by template-grounded skills.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Abrar Anwar, John Welsh, Joydeep Biswas, Soha Pouya, and Yan Chang. Remembr: Building and reasoning over long-horizon spatio-temporal memory for robot navigation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2838–2845. IEEE, 2025.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y Galliker, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Tianxing Chen, Yuran Wang, Mingleyang Li, Yan Qin, Hao Shi, Zixuan Li, Yifan Hu, Yingsheng Zhang, Kaixuan Wang, Yue Chen, et al. Rmbench: Memory-dependent robotic manipulation benchmark with insights into policy design. *arXiv preprint arXiv:2603.01229*, 2026.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 538–547, 2019.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*, 2025.

Pascale Fung, Yoram Bachrach, Asli Celikyilmaz, Kamalika Chaudhuri, Delong Chen, Willy Chung, Emmanuel Dupoux, Hongyu Gong, Hervé Jégou, Alessandro Lazaric, et al. Embodied ai agents: Modeling the world. *arXiv preprint arXiv:2506.22355*, 2025.

Theophile Gervet, Soumith Chintala, Dhruv Batra, Jitendra Malik, and Devendra Singh Chaplot. Navigating to objects in the real world. *Science Robotics*, 8(79):eadf6991, 2023.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025.

Chi-Pin Huang, Yueh-Hua Wu, Min-Hung Chen, Frank Wang, and Fu-En Yang. Thinkact: Vision-language-action reasoning via reinforced visual latent planning. *Advances in Neural Information Processing Systems*, 38:82782–82802, 2026.

Siyuan Huang, Haonan Chang, Yuhan Liu, Yimeng Zhu, Hao Dong, Peng Gao, Abdeslam Boularias, and Hongsheng Li. A3vlm: Actionable articulation-aware vision language model. *arXiv preprint arXiv:2406.07549*, 2024.

Hanxiao Jiang, Binghao Huang, Ruihai Wu, Zhuoran Li, Shubham Garg, Hooshang Nayyeri, Shenlong Wang, and Yunzhu Li. Roboexp: Action-conditioned scene graph via interactive exploration for robotic manipulation. *arXiv preprint arXiv:2402.15487*, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Zixing Lei, Changxing Liu, Yichen Xiong, Minhao Xiong, Yuanzhuo Ding, Zhipeng Zhang, Weixin Li, and Siheng Chen. Towards long-horizon embodied agents with tool-aligned vision-language-action models. *arXiv preprint arXiv:2605.13119*, 2026.

Hang Li, Fengyi Shen, Dong Chen, Liudi Yang, Xudong Wang, Jinkui Shi, Zhenshan Bing, Ziyuan Liu, and Alois Knoll. Remem-vla: Empowering vision-language-action model with memory via dual-level recurrent queries. *arXiv preprint arXiv:2603.12942*, 2026.

Xiaoqi Li, Mingxu Zhang, Yiran Geng, Haoran Geng, Yuxing Long, Yan Shen, Renrui Zhang, Jiaming Liu, and Hao Dong. Manipllm: Embodied multimodal large language model for object-centric robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18061–18070, 2024.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Wenlong Liang, Rui Zhou, Yang Ma, Bing Zhang, Songlin Li, Yijia Liao, and Ping Kuang. Large model empowered embodied ai: A survey on decision-making and embodied learning. *arXiv preprint arXiv:2508.10399*, 2025.

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

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. Sam 2: Segment anything in images and videos. In *International Conference on Learning Representations*, volume 2025, pp. 28085–28128, 2025.

Sahar Salimpour, Lei Fu, Kajetan Rachwał, Pascal Bertrand, Kevin O'Sullivan, Robert Jakob, Farhad Keramat, Leonardo Militano, Giovanni Toffetti, Harry Edelman, et al. Towards embodied agentic ai: Review and classification of llm-and vlm-driven robot autonomy and interaction. *arXiv preprint arXiv:2508.05294*, 2025.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations*, volume 2026, pp. 18567–18602, 2026.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Scaling up memory for robotic control via experience retrieval. In *International Conference on Learning Representations*, volume 2026, pp. 97142–97166, 2026.

Jianhua Sun, Yuxuan Li, Longfei Xu, Nange Wang, Jiude Wei, Yining Zhang, and Cewu Lu. Conceptfactory: Facilitate 3d object knowledge annotation with object conceptualization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 75454–75467. Curran Associates, Inc., 2024. doi: 10.52202/079017-2402.

Jianhua Sun, Yuxuan Li, Jiude Wei, Longfei Xu, Nange Wang, Yining Zhang, and Cewu Lu. Artipg: A toolbox for procedurally synthesizing large-scale and diverse articulated objects with rich annotations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6396–6405, October 2025.

Huajie Tan, Xiaoshuai Hao, Cheng Chi, Minglan Lin, Yaoxu Lyu, Mingyu Cao, Dong Liang, Zhuo Chen, Mengsi Lyu, Cheng Peng, et al. Roboos: A hierarchical embodied framework for cross-embodiment and multi-agent collaboration. *arXiv preprint arXiv:2505.03673*, 2025.

Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian Mcauley, Dan Gutfreund, Rogerio Feris, and Zexue He. M+: Extending memoryllm with scalable long-term memory. In *International Conference on Machine Learning*, 2025a.

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, 2025b.

Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 17868–17879, 2024.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11097–11107, 2020.

Yue Yang, Shuo Cheng, Yu Fang, Homanga Bharadhwaj, Mingyu Ding, Gedas Bertasius, and Daniel Szafir. Lilo-vla: Compositional long-horizon manipulation via linked object-centric policies. *arXiv preprint arXiv:2602.21531*, 2026.

Zhejian Yang, Yongchao Chen, Xueyang Zhou, Jiangyue Yan, Dingjie Song, Yinuo Liu, Yuting Li, Yu Zhang, Pan Zhou, Hechang Chen, and Lichao Sun. Agentic robot: A brain-inspired framework for vision-language-action models in embodied agents, 2025.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025a.

Lingfeng Zhang, Yuecheng Liu, Zhanguang Zhang, Matin Aghaei, Yaochen Hu, Hongjian Gu, Mohammad Ali Alomrani, David Gamaliel Arcos Bravo, Raika Karimi, Atia Hamidizadeh, et al. Mem2ego: Empowering vision-language models with global-to-ego memory for long-horizon embodied navigation. *arXiv preprint arXiv:2502.14254*, 2025b.

Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 16259–16268, 2021.

Mandi Zhao, Yijia Weng, Dominik Bauer, and Shuran Song. Real2Code: Reconstruct articulated objects via code generation. In *International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=CAssIgPN4I`.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Yichen Zhu, Zhicai Ou, Xiaofeng Mou, and Jian Tang. Retrieval-augmented embodied agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17985–17995, 2024.

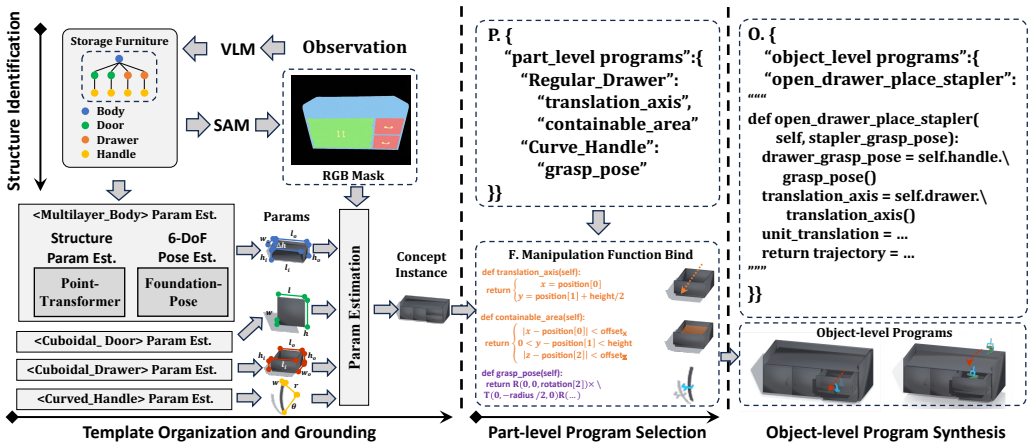

Figure 2: **Perception and object code grounding.** The agent invokes visual foundation models and parameter estimators as tools to convert observations into grounded object code representations. VLM-guided structure identification and SAM-based segmentation provide object and part masks, while pose and parameter estimation tools instantiate matched templates into grounded part concepts. The resulting part-level programs are composed into object-level programs for downstream manipulation.

A Implementation Details

A.1 Perception and Object Code Grounding

We implement perception and object code grounding as a tool-augmented process, as illustrated in Fig. 2. Instead of relying on a monolithic perception model, the agent invokes a set of vision foundation models and parameter estimators as external tools. Given an RGB-D observation, the agent first identifies task-relevant objects and candidate parts with VLM-guided object/part recognition and SAM-based (Ravi et al., 2025) segmentation. The resulting masks are then passed to pose and parameter estimation tools, such as FoundationPose (Wen et al., 2024) for 6D pose estimation and template-specific parameter estimators for structural fitting.

For structural parameter estimation, we adopt a Point-Transformer (Zhao et al., 2021) encoder that extracts 128 groups of points with group size 32 from the input point cloud and applies 12 attention layers with 6 heads each. We then apply average pooling to obtain a global point-cloud feature, followed by a three-layer MLP with ReLU activations to regress the template-specific structural parameters.

The grounding process is organized around template types. For each detected part, the agent selects a candidate template type τ_i according to the semantic synopsis in $\mathcal{M}^{sem}$ and the structural template in $\mathcal{M}^{temp}$. The perception tools estimate scene-specific parameters and poses, denoted as $(\hat{\Theta}_i, \hat{T}_i)$, which are then bound to the matched template $m^{temp}_{\tau_i}$. This produces a grounded part code representation

$$P_i = \mathcal{P}(\tau_i, \hat{\Theta}_i, \hat{T}_i, A_i, \mathcal{F}_{\tau_i}),$$

where A_i denotes the inferred part affordance and $\mathcal{F}_{\tau_i}$ denotes the manipulation functions supported by the corresponding template type.

After part-level grounding, the agent assembles grounded parts into an object-level code representation. The object record stores the persistent object identity, template-instantiated parts, estimated geometry, grounded poses, and supported manipulation functions. These grounded representations are written into object memory and linked to scene memory through spatial and state relations. In this way, the output of visual tools is not stored as raw masks or embeddings, but converted into object code entries with callable manipulation interfaces that can be retrieved, reasoned over, and reused by the embodied agent.

Programmatic interface. The following schematic illustrates an object record and its part interface; field names are illustrative rather than an exact serialization format:

```
object: {id, category, pose, state, parts, history}
part: {template_type, geometry, pose, affordances}
handle = object.parts["handle"]
f = template_library[handle.template_type].grasp_pose
target_pose = f(handle.geometry, handle.pose)
```

The part's template type identifies a reusable concept, while its geometry and pose specify the current instance. The function call derives a manipulation target from those grounded fields; the execution backend then uses that target. This interface connects structured records to callable procedures. The template-based geometric representation follows analytic concepts (Sun et al., 2024; 2025); OCLM organizes its instantiations into persistent, linked memory records.

In addition to structural grounding, the same template linkage is used to derive part-level and object-level programs. Part-level programs expose reusable functions defined by the matched templates, such as estimating a drawer translation axis, computing a containable region, or generating a handle grasp pose. Object-level programs compose these part-level functions into executable routines, such as opening a drawer and placing an object inside. Thus, concept grounding provides both a structured memory entry and a programmatic interface for downstream skill selection and execution.

Verification-based updates preserve memory consistency under incomplete or uncertain perception. Missing, low-confidence, or inconsistent tool outputs are retained as partial observations, and existing object records are updated after verification by subsequent observations or execution feedback. This design combines the grounding capabilities of vision foundation models with persistent, explicitly structured memory.

A.2 Uncertainty and Conflicting Memory Updates

We propose a lightweight field-level update protocol using the existing agent and perception tools.

Field records. Each mutable field, such as pose, articulation state, or a scene relation, stores its value, observation timestamp, source tool, and status (`verified`, `stale`, or `pending`). A missed detection retains the object's identity and history, marks its visibility as unknown, and leaves unobserved fields stale. A conflicting estimate is stored as a pending candidate without overwriting the previous value.

Verification and updates. The agent checks pending observations against recent actions and requests a new observation or a targeted tool estimate. Discrete states are checked for agreement; geometric estimates are checked using tool-specific tolerances. A confirmed candidate updates the current field, while the previous value remains in the interaction history. Unresolved object associations remain pending, so candidate identities and their histories are not merged. Confidence scores are used only when provided by the tool.

Execution checks. Before planning, the agent refreshes stale or pending fields required by the selected skill. If they cannot be verified, it defers that skill and re-analyzes the scene. After execution, an observation or state-estimation tool checks the physical postcondition. The outcome is recorded as `success`, `failure`, or `unverified`; verified observations update object and scene memory, and outcome records update transition and skill memory. Predicted effects are not written as observed states merely because the planner reports completion.

A.3 Manipulation Knowledge Transfer: Experimental Details

This subsection provides additional details for the manipulation knowledge transfer experiments in Section 5.2.

Environments and categories. We evaluate articulated object manipulation in simulation using the SAPIEN simulator (Xiang et al., 2020), with tasks drawn from *ManiSkill* and assets from *PartNet-*

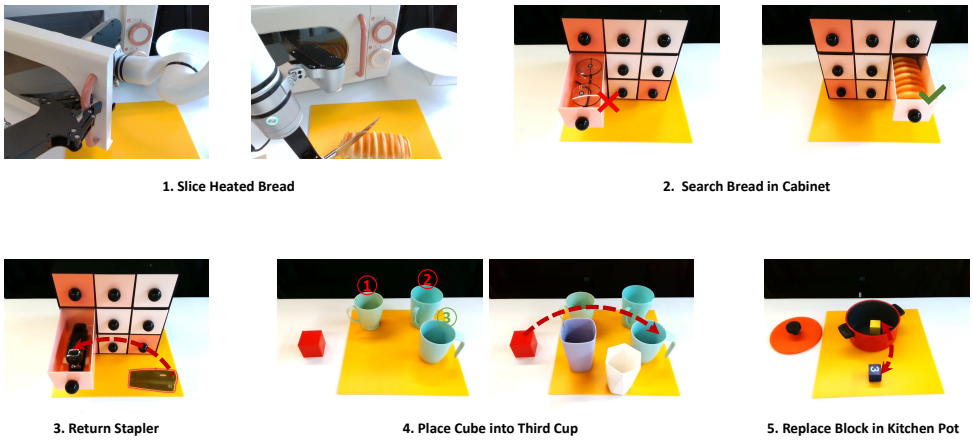

Figure 3: Real-world experimental setup for the tabletop manipulation tasks in Section 5.3.

Mobility. Each episode contains a single articulated target object with a manipulable part (e.g., a door, lid, or handle) whose kinematic state is observable only through perception and interaction.

Episode initialization. In each trial, the target object is placed at the center of the scene. The articulated joint configuration is randomized, with a 50% probability of starting from the fully closed state and a 50% probability of starting from an open configuration sampled randomly along the joint's range of motion. We use an RGB-D camera with known intrinsic parameters that points toward the scene center and is positioned on the upper hemisphere, with azimuth sampled uniformly from $[0°, 360°)$ and elevation sampled uniformly from $[30°, 60°]$.

Success criteria and metrics. We follow the interaction-success criterion of Where2Act (Mo et al., 2021). A valid interaction must produce a target-part joint displacement $\Delta q = |q_{\text{end}} - q_{\text{start}}|$ satisfying $\Delta q > 0.01$ or $\Delta q/(q_{\max} - q_{\min}) > 0.5$. The absolute threshold is expressed in the joint's native coordinate units (length for prismatic joints and radians for revolute joints); the relative threshold is dimensionless. Following the released evaluator, pulling and directional pushing/pulling additionally require the contact-point displacement to form an angle smaller than $60°$ with the intended action direction; plain pushing uses the validity and motion-magnitude checks. We report success rate (SR, %) as the fraction of successful trials within each category in Table 2, using identical success predicates for all methods.

Interaction budget. We adopt an interaction budget of 5 for each action proposal.

Reporting. Unless otherwise stated, we use the same prompting templates and output schemas across methods within each setting, and we aggregate results across randomized initializations to reduce sensitivity to a single pose or viewpoint.

A.4 Real-Robot Experimental Setup

Robot platform. We conduct the real-world experiments on a dual-arm platform comprising two RealMan RM75 robotic arms, each with seven degrees of freedom.

Perception. We use three RGB-D cameras (Intel RealSense D435): two wrist-mounted cameras and one egocentric camera.

Compute. The system runs on a workstation equipped with an Intel Core i9-14900K processor, 64GB of RAM, and an NVIDIA RTX 4090 GPU, enabling real-time inference and planning. For high-level reasoning in the real-robot experiments, we use GPT-5.6 Sol (high), the same model as in the RMBench main and memory-ablation experiments.

Table 5: Real-world tabletop tasks for evaluating object–concept linked memory. The tasks cover different memory capabilities, including object identity, scene relations, state transitions, search history, and reusable skill selection.

Task	**Slice Heated Bread**
Preceding task	Heat the bread in a microwave.
Task description	Retrieve the heated bread, place it on the cutting board, and slice it with the knife.
Main challenge	Scene memory for bread location; skill memory for microwave operation and slicing.
Task	**Search Bread in Cabinet**
Preceding task	–
Task description	Search the cabinet and find the bread.
Main challenge	Scene memory for searched locations.
Task	**Return Stapler**
Preceding task	Observe the stapler at its initial location and move it to another workspace region.
Task description	Put the stapler back to its original location.
Main challenge	Object memory for persistent identity; scene memory for original location.
Task	**Place Cube into Third Cup**
Preceding task	Arrange multiple cups in order; optionally change cup positions.
Task description	Pick up the cube and place it into the third cup.
Main challenge	Transition memory for tracking cup order; object memory for target cup disambiguation.
Task	**Replace Block in Kitchen Pot**
Preceding task	A block with a random color is placed into the Kitchen Pot.
Task description	Replace the block in the Kitchen Pot with another specified color block.
Main challenge	Scene memory for the hidden block location.

Evaluation trials. Each method is evaluated over 20 trials per task. Task success rate is the number of successful trials divided by 20; the average SR in Table 3 is the unweighted mean across tasks.

Metrics. In addition to success rate, we emphasize retrieval correctness and efficiency:

- **Success rate (SR):** fraction of episodes with full task completion within the episode horizon.
- **Retrieval accuracy:** for each subtask-level retrieval query, we mark it correct only if *both* (i) the retrieved target object identity matches the intended object instance and (ii) the retrieved skill (template- or policy-grounded) matches the intended manipulation procedure. For each task, retrieval accuracy is the mean correctness over its subtask queries. The average in Table 3 is computed over the three tasks with reported RA values; entries marked – are excluded.
- **Retrieval efficiency (RE):** mean elapsed time in seconds per memory retrieval query, measured from submission of a formed query to the retrieval module until the retrieved records are returned to the agent. This interval covers query processing and record retrieval, including filtering, ranking, and memory joins where applicable, but excludes LLM query generation, subsequent reasoning, and robot execution. For each task, we average latency over retrieval queries across evaluation episodes; the *Avg.* row in Table 3 reports the unweighted mean of the task-level values. Lower RE indicates faster retrieval.

Tasks. We evaluate on five real-world tabletop tasks (Table 5), each specified by a preceding context task and a target task that stresses particular memory capabilities.

B Additional Ablation Details

B.1 Memory Organization and Representation

Ablation setup. All memory-ablation variants use GPT-5.6 Sol (high) for high-level reasoning, matching the RMBench main experiments. Table 6 defines the variants evaluated on RMBench,

Table 6: Memory ablation definitions. Each intervention is applied separately to full OCLM. All variants retain the same available execution capabilities.

Variant	Intervention
Full OCLM	All memory components and associations enabled.
w/o object history	Retain current grounded objects; disable persistent identity and access to previous object records.
w/o scene history	Rebuild relations and states from current observations; do not retrieve previous scene snapshots.
w/o transition history	Disable retrieval of observed action effects; retain current-state checks and skill definitions.
w/o skill experience	Retain all callable skills; disable retrieval of previous skill bindings and outcome statistics.
Flat records	Serialize the same stored facts; use embedding retrieval without indexed cross-memory joins.
Generic object records	Preserve grounded quantities and skill access; omit template types and part-template links from the memory interface.

with results reported in Table 4. Each variant changes one memory component or representation relative to full OCLM. The component ablations disable access to object, scene, transition, or skill history, while retaining the available perception tools, callable skills, and execution capabilities. Thus, removing skill experience disables retrieval of past bindings and outcomes rather than removing the skills themselves.

Evaluation episodes and randomization. For the RMBench main and memory-ablation experiments, we follow the official Mem-0 evaluation script and default configuration (Chen et al., 2026), using 100 rollout episodes per task and method or variant. The evaluator starts candidate environment seeds at $100000(1 + s)$, where s is the configured `seed` argument, and increments the candidate seed sequentially. Scenes that fail the expert feasibility check are skipped before policy evaluation; policy successes and failures on the accepted scenes both count toward the 100 evaluated episodes. Task SR is the fraction of successful rollouts, and aggregate SR is the unweighted mean across the reported tasks. The 100 rollouts are episode-level evaluations, not 100 independent training runs. The default script performs one evaluation batch per invocation; it does not automatically average results over multiple training seeds.

Organization versus representation. The flat-record variant stores the same facts, including relations serialized as text, but uses embedding retrieval instead of indexed cross-memory joins. This comparison examines the combined memory organization and retrieval design. The generic-object-record variant preserves grounded geometry, poses, affordances, and access to callable skills, but omits part-template types and part-template links from the memory interface. It therefore tests the template-based memory representation rather than removing template priors from the entire perception pipeline.

Metrics and results. Table 4 reports task success rate (SR), joint object–skill retrieval accuracy (RA), and object re-identification accuracy (Re-ID). Re-ID is not applicable to the variant without object history because persistent identity is disabled. Relative to full OCLM, removing object or scene history reduces SR by 15 or 14 percentage points, respectively, compared with 7 or 5 points for transition or skill history. Flat and generic object records reduce SR by 12 and 9 points and Re-ID by 13 and 7 points, respectively. These comparisons support the contributions of persistent history, linked retrieval, and template-based representations. Sensitivity to the reasoning LLM is examined separately in Appendix B.2.

B.2 LLM Model Sensitivity

We vary the LLM model used by the high-level reasoning modules (e.g., subtask decomposition, concept/skill selection, and memory query formulation), while keeping all non-LLM components fixed. We use the same prompts and output schemas across models.

Table 7: Sensitivity to the LLM model used in high-level reasoning. We report success rate on RMBench, retrieval accuracy, and object re-identification accuracy.

LLM model	RMBench SR ↑	Retrieval acc. ↑	Obj. re-id acc. ↑
GPT-5.6 Sol (high)	71%	95%	85%
GPT-5.5	69%	95%	80%
Gemini-3.1-pro	65%	89%	75%
GPT-4o	67%	91%	80%
Qwen3.7-Plus	65%	88%	75%

Table 7 demonstrates that OCLM supports multiple reasoning models, achieving 65–71% task success, 88–95% retrieval accuracy, and 75–85% object re-identification accuracy across the five evaluated LLMs. GPT-5.6 Sol (high) achieves the strongest overall results, with the highest SR (71%) and re-identification accuracy (85%), while matching GPT-5.5 for the highest retrieval accuracy (95%). Compared with GPT-5.5, it improves SR and re-identification by 2 and 5 percentage points, respectively. With Gemini-3.1-pro and Qwen3.7-Plus, OCLM achieves the same SR of 65%, retrieval accuracies of 89% and 88%, and the same re-identification accuracy of 75%. The consistently high retrieval accuracy across these model families supports the applicability of the structured memory interface to different high-level reasoning backbones.

B.3 Memory Scalability

Data and evaluation scope. We construct manipulation-task data in ManiSkill 3 using objects selected from PartNet-Mobility (Xiang et al., 2020). This evaluation examines retrieval of object manipulation skills as task experience accumulates. We report results at 50 and 100 episodes, followed by increments of 100 up to 1000 episodes. The horizontal axis counts task episodes, not distinct object assets or scene layouts.

Retrieval target and metrics. The retrieval target is a manipulation skill applicable to an object and its manipulation goal. We report retrieval latency in milliseconds and skill-retrieval accuracy in percent. These metrics characterize memory retrieval rather than physical task completion; skill-retrieval accuracy is distinct from the joint object–skill retrieval metric used in the main experiments. The intended timing interval runs from submission of a formed query to return of skill records, excluding LLM query generation and subsequent reasoning.

Results. Figure 4 shows that OCLM maintains efficient and accurate skill retrieval as accumulated task experience grows from 50 to 1000 episodes. This twentyfold increase in experience raises retrieval latency by only approximately 2.17 times, from 89 to 193 ms, keeping the reported latency below 200 ms at every evaluated scale. At 1000 episodes, skill-retrieval accuracy remains at 84.7%, compared with 94.0% at 50 episodes. Accuracy is particularly stable from 500 to 1000 episodes, ranging from 84.5% to 86.7% as the episode count doubles. Together, these results demonstrate that OCLM accommodates substantially more manipulation experience while preserving low retrieval latency and consistently high skill-retrieval accuracy across the evaluated scales.

C Atomic Manipulation Primitives

The manipulation functions in $\mathcal{F}_\tau$ are implemented by composing a small set of *atomic* manipulation primitives (Table 8). Each primitive corresponds to a low-level, robot-executable operation (e.g., reaching, grasping, or contact-driven pushing) with a standardized interface over grounded objects/parts and geometric parameters. Given a grounded part representation $P = \mathcal{P}(\tau, \Theta, T, A, \mathcal{F}_\tau)$, a template function $f \in \mathcal{F}_\tau$ first derives the required arguments (target poses, axes, or goal positions) from (Θ, T, A) and the current scene state, and then invokes the appropriate primitive(s). More complex behaviors—such as opening an articulated component or inserting an object—are realized as short sequences of primitives with intermediate state checks, enabling consistent execution and outcome logging for transition and skill memory updates.

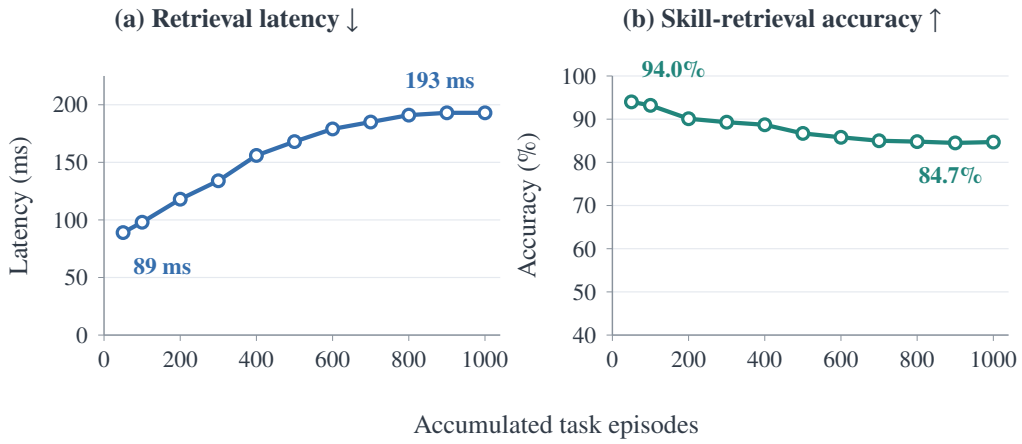

Figure 4: OCLM skill retrieval as manipulation-task experience accumulates in ManiSkill 3 using PartNet-Mobility objects. (a) Retrieval latency; (b) skill-retrieval accuracy. Markers show the reported measurements at each episode count; connecting lines guide the eye. Across a twentyfold increase in task episodes, retrieval latency stays below 200 ms and skill-retrieval accuracy remains above 84%, reaching 193 ms and 84.7% at 1000 episodes.

D Potential Risks

This work focuses on controlled simulation and tabletop manipulation. Potential risks mainly arise from the broader deployment of memory-enabled robotic agents. In less controlled environments, incorrect retrieval or inappropriate skill execution may cause unintended object movement, object damage, or unsafe robot behavior. Such systems should therefore be used with workspace constraints, execution monitoring, and human oversight.

Memory-enabled agents may also store information about objects, scene layouts, and user environments. Although our experiments only record task-relevant object states and manipulation outcomes, real-world deployments should include memory management policies, such as limiting stored content, supporting memory deletion, and protecting stored information from unauthorized access. Our system is intended as a research framework for controlled embodied manipulation, not for safety-critical or human-facing deployment.

Algorithm 1 Closed-Loop Object–Concept Linked Memory Agent

Require: Stable semantic library $\mathcal{M}^{sem}$, template library $\mathcal{M}^{temp}$
Initialize dynamic memories $\mathcal{M}^{obj}, \mathcal{M}^{scene}, \mathcal{M}^{trans}, \mathcal{M}^{skill}$
$\mathcal{M} \leftarrow \{\mathcal{M}^{sem}, \mathcal{M}^{temp}, \mathcal{M}^{obj}, \mathcal{M}^{scene}, \mathcal{M}^{trans}, \mathcal{M}^{skill}\}$
while agent is active **do**
  Receive an episode instruction L
  Observe current scene o_t
  ▷ **Memory-guided task parsing**
  Retrieve relevant past objects, states, transitions, and skills from $\mathcal{M}$
  Decompose L into subtasks $\mathbf{l} = \{l_1, \ldots, l_N\}$ conditioned on retrieved memory
  for each subtask $l_i \in \mathbf{l}$ **do**
    Observe current scene o_t
    ▷ **Phase I: Memory construction and grounding**
    Parse o_t to obtain candidate objects, parts, and scene relations
    Invoke perception tools for detection, segmentation, pose estimation, and parameter estimation
    for each detected part **do**
      Match it to a template type τ using $\mathcal{M}^{sem}$ and $\mathcal{M}^{temp}$
      Instantiate grounded part concept

$$P_{k,j} = \mathcal{P}(\tau_{k,j}, \hat{\Theta}_{k,j}, \hat{T}_{k,j}, A_{k,j}, \mathcal{F}_{\tau_{k,j}})$$

    end for
    Resolve persistent object identities and update object memory $\mathcal{M}^{obj}$
    Update scene memory $\mathcal{M}^{scene}$ with object states and scene relations
    ▷ **Phase II: Memory-grounded execution**
    Retrieve task-relevant memory entries:

$$\mathcal{R}_t = \mathrm{Retrieve}_{fine}(\mathrm{Retrieve}_{coarse}(L, o_t, l_i), \mathcal{M})$$

    Obtain candidate objects, scene states, transitions, and skills from $\mathcal{R}_t$
    for each candidate skill u **do**
      Check preconditions with current scene state:

$$\mathrm{App}(u, G_t) = \mathbb{I}[pre(u) \subseteq state(G_t)]$$

      Predict expected postcondition using transition memory
    end for
    Select an applicable skill u^* consistent with subtask l_i
    Instantiate u^* with grounded object and part parameters
    Send the resulting low-level action sequence to the robot executor
    ▷ **Feedback and memory update**
    Observe execution feedback and updated observation o_{t+1}
    if the predicted postcondition is verified **then**
      Update object and scene states in $\mathcal{M}^{obj}$ and $\mathcal{M}^{scene}$
      Record transition

$$m^{trans} = (id_k, P_{k,j}, u^*, s_{\mathrm{pre}}, s_{\mathrm{post}}, r)$$

      Update skill statistics in $\mathcal{M}^{skill}$
    else
      Mark the skill attempt as failed and update skill statistics
      Re-query memory or select an alternative skill in the next loop
    end if
    $t \leftarrow t + 1$
  end for
end while

Table 8: Definitions of atomic manipulation primitives used for template-grounded skill execution.

Primitive	Definition
`grasp(obj)`	The robot establishes a stable grasp on the target object by closing the gripper or applying a holding constraint through the end-effector. The action results in the object becoming attached to the manipulator for subsequent manipulation.
`reach(obj)`	The manipulator moves toward the target object while maintaining the current gripper state unchanged. The action terminates before grasping, contact-driven translation, or any interaction that changes the object state occurs.
`move(pos, obj)`	The manipulator translates the end-effector or a grasped object toward a target position defined relative to a reference object or workspace region, while maintaining the current grasp configuration unchanged. The action represents spatial relocation without intentional contact interaction.
`flip(obj, axis)`	The robot rotates the target object around a specified axis, typically by approximately 180°, such that the object's vertical orientation, exposed surface, or functional side becomes inverted. Unlike general rotational adjustment, this primitive emphasizes discrete orientation reversal.
`rotate(obj, axis, angle)`	The robot changes the orientation or articulated state of the target object through rotational motion without causing vertical inversion. This primitive includes in-plane pose adjustment, 3D orientation alignment, and manipulation of rotational joints or mechanisms.
`pull(obj, pos)`	The end-effector establishes pulling contact with the target object and moves backward relative to the object, generating traction that causes the object to translate toward the target position. The action terminates once object motion ceases and contact is released.
`push(obj, pos)`	The end-effector makes contact with the target object and applies forward translational force, causing the object to move toward a target position under sustained contact. The action ends when the object stops moving and the pushing contact is terminated.
`insert(obj1, obj2)`	The robot aligns object `obj1` with an opening, slot, cavity, or receptacle associated with `obj2`, and subsequently translates `obj1` along the insertion axis until partial or full engagement is achieved. The action emphasizes constrained motion under geometric alignment.
`release(obj)`	The robot removes the grasping constraint from the target object by opening the gripper or disengaging the holding mechanism. After completion, the object is no longer mechanically constrained by the manipulator and remains free in the environment.