MEMORY-AS-SKILL FOR COMPOSITIONAL GENERALIZATION IN EMBODIED MULTI-AGENT COOPERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world embodied cooperation happens under partial observation and without a central coordinator, so each agent has to learn its own abilities and its teammates' behavior from past interaction. Its memory therefore has to carry both what the agent can do on its own and how it coordinates with teammates. Such a memory is most useful when it generalizes, because new tasks often recombine familiar elements in unseen ways. A memory of raw trajectories supports this poorly, since each trajectory binds actions and coordination to a single episode. We propose Memory-as-Skill, which turns multi-agent interaction traces into a library of individual and cooperation skills, each an abstract behavior with grounded instances. An LLM proposes skills from segmented traces, and a proposal enters the library only after it passes an execution-based admission gate on separate validation episodes. The library is then retrieved at test time to guide execution. On two embodied multi-agent benchmarks, the admitted library transfers to an unseen subtask combination, to held-out task families, and to unseen constraint compositions. It exceeds existing memory methods built from the same traces, by a wide margin on one benchmark and a small one on the other. Ablations show that admission, cooperation skills, and the skill hierarchy each contribute to this transfer.

1 INTRODUCTION

Embodied multi-agent cooperation involves several LLM-driven agents completing long-horizon household or multi-robot tasks together (Zhang et al., 2023; Chang et al., 2025; Kang et al., 2026). In real deployments each agent sees only part of the scene, and a central coordinator or a reliable communication channel is often unavailable. Each agent therefore has to learn what it can do and how its teammates behave from past interaction. Memory provides this, since tasks share structure across episodes and an agent that retains past experience can reuse it (Zhang et al., 2025b; Xu et al., 2025; Shinn et al., 2023). In a decentralized system this memory has to carry both individual ability, such as fetching an object, and coordinated behavior, such as dividing labor or handing an object over (Dorigo et al., 2021; Zhang et al., 2018). Without the latter, agents take redundant or conflicting actions from local observations alone. Explicit communication (Zhang et al., 2023) and partner-belief modules (Zhang et al., 2024a; Li et al., 2023) can supply the coordination at the cost of messages or inference. We ask instead whether memory can hold it. Whether such a memory helps then depends on generalization. New tasks recombine familiar objects, rooms, and partners, and a memory is only useful if what it stores carries over to these unseen combinations (Lake & Baroni, 2018).

Existing agent memory takes a few main forms. Case-based methods retain whole trajectories and retrieve them by similarity (Zhong et al., 2023; Zhang et al., 2025a). A trajectory ties the reusable behavior to the objects, rooms, and partners of one episode, so retrieval fails when a test task recombines parts that memory holds only in separate trajectories. Skill-based and hybrid methods distill procedures from experience (Wang et al., 2023; 2024; Zhao et al., 2024; Tang et al., 2025), but they are designed for a single agent. A skill is admitted once it succeeds in the episode from which it is extracted, without being tested on episodes with different objects, rooms, or partners. Multi-agent memory systems organize interaction histories (Zhang et al., 2025a), but they neither separate cooperation knowledge from individual knowledge nor check that a stored entry can be

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.

executed again. None of them stores behavior in a form that is abstracted from one episode, covers cooperation as well as individual action, and is verified before reuse.

We propose Memory-as-Skill, which organizes experience around executable skills rather than complete interaction histories. Each skill separates what to do from where and on what to do it. It is written with variables in place of specific objects, rooms, and partners, and its instances record the concrete bindings from past episodes, so it can be rebound to a new episode. The library holds individual and cooperation skills, and a cooperation skill records a response to a partner situation rather than to a partner identity (Hu et al., 2025). An LLM proposes candidates from segmented traces, and a candidate enters the library only after it executes on separate validation episodes and passes an admission gate. At test time, an agent retrieves matching skills, binds their roles to the entities of the current episode, and follows their demos. It does so one action at a time with replanning when the benchmark takes one action per round, and in one pass when it takes one complete plan. Coordination comes from the retrieved cooperation skills, which encode the interaction pattern at extraction time, so the method adds neither messages nor a partner-belief module.

Our contributions are as follows. First, a skill library as memory for embodied multi-agent cooperation, holding individual and cooperation skills written with variables and paired with grounded instances. Second, execution-based admission for building it. A candidate enters the library only if it executes after its entities are rebound on separate episodes, covers its source segment, and recurs across traces. On VIKI-R (Kang et al., 2026) a library mined by replaying reference plans and certified by the same gate transfers as well as the proposed one, so the gate rather than the proposer determines the library. Third, Memory-as-Skill transfers to a held-out subtask composition and to held-out task families on VIKI-R, a multi-robot planning benchmark, with the larger backbones, and to constraint compositions on PARTNR (Chang et al., 2025), a decentralized household benchmark. On VIKI-R this is an end-to-end result in which our method grounds skill demos by role binding while the baselines write primitive plans, and controlled comparisons attribute it to the admitted library together with that grounding step. On PARTNR all memory methods share the same in-context use and none exchanges messages, and ours outperforms existing memory methods across open-weight backbones by small margins. Admission separates a library that transfers from one that does not on both benchmarks, and removing the cooperation skills or the skill hierarchy removes the transfer to constraint compositions.

2 Related Work

Memory in large language model agents. LLM-agent memory has progressed from external stores such as MemoryBank (Zhong et al., 2023) to structured systems such as structural memory (Zeng et al., 2024) and Memory^3 (Yang et al., 2024), and recent work treats memory as part of reasoning and action selection (Xu et al., 2025; Zhang et al., 2024b). Skill libraries such as Voyager (Wang et al., 2023) and Agent Workflow Memory (Wang et al., 2024) store programs or workflows induced from an agent's own experience. A related line lets an LLM propose symbolic operators or world models and checks them by planning or execution (Wong et al., 2023; Guan et al., 2023). These methods target a single agent, draw candidates from the model's prior or a task description rather than from recorded traces, and verify a proposal in the setting that produced it. Our candidates come from traces, are checked by rebinding and execution on other episodes, and include cooperation behavior. A library proposed from instructions alone by the same model passes the same gate with two skills and solves nothing (Section 5.2). Hybrid systems such as ExpeL (Zhao et al., 2024) and Agent KB (Tang et al., 2025) combine concrete episodes with abstract guidance, and G-Memory (Zhang et al., 2025a) organizes multi-agent experience in a hierarchical graph. None of them admits a skill by executing it after rebinding on separate episodes, and none separates cooperation skills from individual ones.

Decentralized multi-agent systems. LLM-based multi-agent research studies collaboration (Guo et al., 2024; Tran et al., 2025), decentralized execution (Chen et al., 2024; Yang et al., 2025), and embodied cooperation (Zhang et al., 2023). Multi-agent memory systems (Wang & Chen, 2025; Zhang et al., 2025a) preserve information across agents. We instead study how cooperation demonstrations that each agent retrieves on its own shape decentralized execution, measuring the resulting object conflicts, failed actions, and navigation errors directly.

Compositional generalization. Compositional generalization asks whether a learner recombines known parts into unseen wholes (Lake & Baroni, 2018). We study it at the level of agent memory. VIKI-R tests recombination of subtasks from different training families, and PARTNR tests transfer from elementary constraint types to their compositions.

3 PRELIMINARIES

We consider an embodied environment, such as a simulated household or a multi-robot workspace, in which two LLM-driven agents complete a task together. We model it as a decentralized partially observable Markov decision process (Dec-POMDP) (Oliehoek et al., 2016) $(\mathcal{D}, S, A, T, \Omega, O)$ with agent set $\mathcal{D} = \{1, 2\}$, state and action spaces S and A, transition function T, observation space Ω, and observation function O. At step t, agent j receives a partial text or visual observation $o_t^j \in \Omega$ of state $s_t \in S$. All agents receive a natural-language instruction I at the start of an episode, and agent j samples $a_t^j \sim \pi^j(\cdot \mid I, \tau_t^j)$ from its local history $\tau_t^j = (o_1^j, a_1^j, \ldots, o_{t-1}^j, a_{t-1}^j, o_t^j)$.

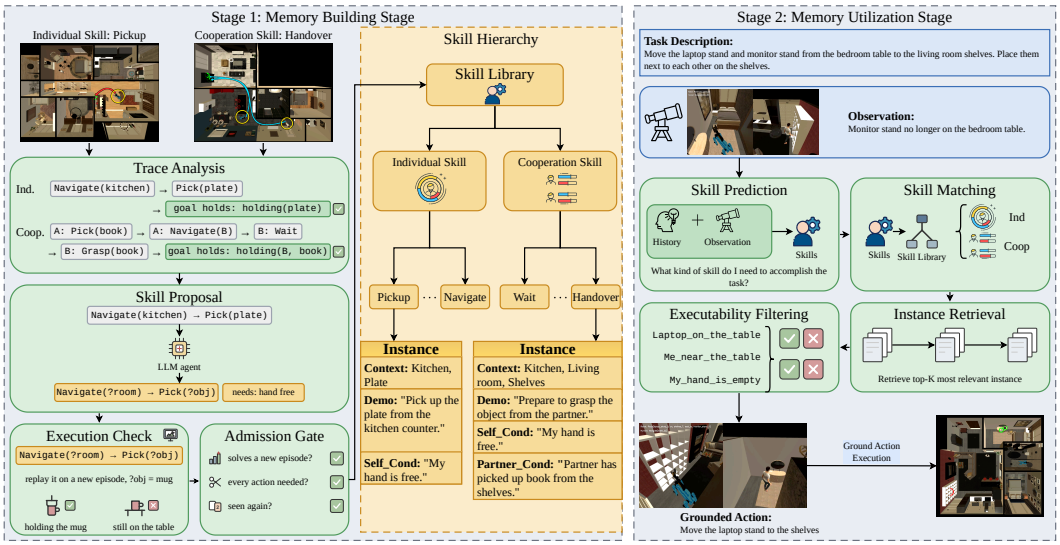

Figure 1: Memory-as-Skill overview. Stage 1 cuts recorded traces into segments, proposes individual and cooperation skills, and admits a candidate only if it executes on separate validation episodes, retains the necessary actions, and recurs across traces. Admitted skills form a hierarchy with grounded instances. Stage 2 runs in every round. The agent queries the library with its instruction, observation, held object, and recent state changes, filters the retrieved instances by preconditions, and the LLM selects a demonstration and outputs the next action (shown on PARTNR, Section 4.2).

4 METHOD

Memory-as-Skill has two stages (Figure 1). Stage 1 builds a library of skills from recorded traces offline, admitting a skill only if it executes on validation episodes that differ from its source. Stage 2 freezes the library and retrieves from it at test time, grounding the retrieved skills in the current episode one action at a time or, for a complete plan, in one pass (Appendix A.2).

4.1 LIBRARY CONSTRUCTION (STAGE 1)

Stage 1 runs once, offline (left panel of Figure 1). Every candidate the LLM proposes passes through the four modules below, so the library is decided by execution rather than by the model that wrote the candidate.

Trace analysis. A recorded trace $\tau = (o_1, a_1, \ldots, o_T, a_T, o_{T+1})$ holds the observations and actions of one agent in one episode (reference plans on VIKI-R, heuristic-agent rollouts on PARTNR), and each observation o_t includes the partner's actions visible to this agent. An LLM reads each trace in

temporal order and marks the steps where a subgoal of the instruction becomes satisfied. These steps cut the trace into segments $\sigma = (a_p, \ldots, a_q)$, each a contiguous run of actions ending at one subgoal boundary. The effect of a segment is the difference between o_p and o_{q+1}, and it can include changes the partner causes during the segment, provided the acting agent observes them. Trace analysis only reads the recorded trace and does not re-execute it.

Skill proposal. For each segment, an LLM proposes a candidate skill by replacing the specific objects, rooms, and partners in the segment's actions and effect with roles, so the candidate is the smallest reusable behavior that produces this effect. In Figure 1, Navigate(kitchen) then Pick(plate) becomes Navigate(?room) then Pick(?obj). A candidate has a semantic `name`, a one-sentence description `desc` used for retrieval, a `type` (individual or cooperation), an `effect` over roles such as `holding(?obj)`, and preconditions `pre`. It also carries one instance made of a `demo`, the segment's actions in natural language over the same roles, and a `context` that binds the roles to the source entities, for example `(Kitchen, Plate)`. For an individual skill, `pre` is a `self_cond` on the acting agent, such as a free hand. For a cooperation skill, the effect includes a partner-induced change, and `pre` also contains a `partner_cond`. The `partner_cond` describes the partner situation by its observable effect, such as an object no longer being on the table, rather than by an action token or a partner identity, so the same skill applies to new objects, rooms, and partners. In the PARTNR libraries this description is free text read only by the LLM.

Execution check. A source trace shows that a behavior occurred once, not that the candidate still executes after its roles are bound to new entities. We therefore select validation episodes whose initial states satisfy the candidate's preconditions, bind its roles to their entities, execute the demo under this binding, and check whether it produces the stated effect without invalidating the task goal. The check is applied to the candidate, not to the source episode.

Admission gate. A candidate c is tested on validation episodes e that are disjoint from its source episode and satisfy its preconditions. $\mathrm{exec}(c, e)$ holds if binding c to the entities of e and executing its demo produces the stated effect without invalidating the goal of e. $\mathrm{cover}(c)$ holds if the demo of c contains every action of its source segment that the effect requires. $\mathrm{supp}(c)$ is the number of recorded traces that contain a segment matching c. The candidate is admitted when

$$\big|\{\, e : \mathrm{exec}(c, e)\,\}\big| \geq 2 \quad \wedge \quad \mathrm{cover}(c) \quad \wedge \quad \mathrm{supp}(c) > 1. \tag{1}$$

The three conjuncts reject candidates that fail once rebound to new entities, that omit a necessary action, and that occur in a single trace only. Admitted candidates with the same `name` and `type` are merged into one skill $s = (\mathtt{name}, \mathtt{type}, \mathtt{effect}, \mathtt{pre}, \mathcal{I}_s)$, where $\mathcal{I}_s$ collects their $(\mathtt{context}, \mathtt{demo})$ instances. The skill states what the behavior achieves and when it applies, and its instances record where and on what it was carried out, so it can be rebound to new objects, rooms, and partners. Validation episodes are drawn from the induction pool and never from the evaluation splits. The library $\mathcal{L}$ keeps individual skills and cooperation skills at two separate levels, which we call the hierarchy, and is frozen before evaluation.

4.2 Library Use (Stage 2)

Stage 2 reads the frozen library at test time. Each decision round costs one embedding retrieval and one LLM call.

Skill retrieval. Retrieval runs before the LLM call and involves no LLM. Agent j builds a query q_t^j from a text template filled with the instruction I and its current observation o_t^j (seen objects with locations, known rooms, the held object, and the state changes since its previous turn). It also builds a context text c_t^j from the same state to rank instances. Retrieval has two levels.

$$\begin{aligned} \mathcal{S}_t^j &= \{\, s \in \mathcal{L} : \mathrm{sim}(q_t^j, \mathtt{name}(s) \oplus \mathtt{desc}(s)) > \theta \,\}, \\ r(u) &= \mathrm{sim}(c_t^j, \mathtt{context}(u) \oplus \mathtt{demo}(u)), \quad u \in \mathcal{I}_s,\ s \in \mathcal{S}_t^j, \\ \mathrm{score}(s) &= \mathrm{sim}(q_t^j, \mathtt{name}(s) \oplus \mathtt{desc}(s)) \cdot \tfrac{1}{|\mathcal{I}_s|} \textstyle\sum_{u \in \mathcal{I}_s} r(u). \end{aligned} \tag{2}$$

Here $\mathrm{sim}(\cdot, \cdot)$ is cosine similarity over sentence embeddings, $\oplus$ is text concatenation, and θ is a fixed threshold. The first level keeps every skill above θ (Skill Matching in Figure 1), and the second

ranks the instances of each kept skill by context similarity (Instance Retrieval). The top k skills by score(s) enter the prompt with their highest-ranked instances. We use $\theta = 0.3$ and $k = 5$ throughout (Appendix A.7).

Instance filter. A rule-based filter checks the `self_cond` of each retrieved skill against the agent's current state (Executability Filtering in Figure 1), removing for example a skill that requires a held object while the hand is empty. The survivors are the demonstrations for the round.

Action selection. The prompt holds the demonstrations, the execution receipts of the actions taken so far, and the state changes since the agent's previous turn. The agent states what has changed, selects the skill type it needs (Skill Prediction in Figure 1) and the demonstration it follows, and outputs the next action, without querying the library again. These four steps are the `[Diff]`, `[Skill]`, `[Ref]`, and `[Plan]` fields of the prompt in Appendix A.10. A change in object state, including one caused by the partner, triggers replanning with an updated query. If no skill passes the threshold, the agent falls back to zero-shot generation from its own history for that round (Appendix A.7). Agents exchange no messages and keep no partner-belief state. Coordination comes from the cooperation demonstrations each agent retrieves on its own and from replanning when the world state changes.

5 EXPERIMENTS

We organize the experiments around three research questions.

- **RQ1 (transfer):** Does the admitted skill library transfer to unseen compositions and held-out tasks?
- **RQ2 (induction):** Does the admitted library depend on the proposer or on the gate?
- **RQ3 (attribution):** Which parts of the library carry the transfer?

5.1 EXPERIMENTAL SETUP

Benchmarks. On VIKI-R (Kang et al., 2026) we use its level-2 split, VIKI-L2, which gives one instruction per episode and requires one complete multi-robot plan, a single joint action sequence chosen from the instruction and the initial observation. The library is induced once from the reference plans of the induction episodes with Qwen2.5-VL-72B as the proposing LLM and shared by all three backbones, so the 30B and 7B results measure reuse of a library that a larger model induced. The prompt holds the admitted skills and also lists the entity names of the scene and three ordering rules that the proposing LLM summarizes from the induction episodes. The LLM selects, for each subgoal, a skill and the entities that fill its roles, and the demo of that skill is written out under this binding as primitive actions (Appendix A.2). Baselines receive the same entity names and ordering rules and write primitive plans directly, so the two differ only in whether demos are grounded by binding or transcribed by the LLM. We report four splits. In-distribution (ID) draws test instructions from the task families in the library. Single-family OOD withholds one task family from the library and tests on it, and strict OOD additionally withholds the families similar to the held-out one. Compositional generalization (CG) requires a two-robot cutting subtask and an independent delivery subtask in the same plan. Both appear in the induction episodes but never in one induction instruction, and the split is run with and without images (Appendix A.3). The first three splits have $n = 924$ each and CG has $n = 297$ per condition. PARTNR (Chang et al., 2025) gives two-agent household tasks in alternating rounds. One agent acts while the other emits `wait`, and a world-state change between turns triggers replanning. Its library is induced from heuristic-agent traces with Llama-70B as the proposer, and each agent reads it in every round (Section 4.2). Tasks are grouped by the constraint types in their goals, rearrangement (R), spatial relation (S), heterogeneous state change (H), and temporal dependency (T), and their compositions (Appendix A.5). Results are reported per type, and the compositions test transfer across constraint types. An episode succeeds if its terminal state satisfies the instruction, and we report the success rate, plus on PARTNR the completion rate, the fraction of goal propositions satisfied at the terminal state. Backbones are Qwen2.5-VL-7B, Qwen2.5-VL-72B, and Qwen3-VL-30B on VIKI-L2, and Llama-3.1-8B, Llama-3.3-70B, Qwen2.5-7B, and Qwen2.5-72B on PARTNR.

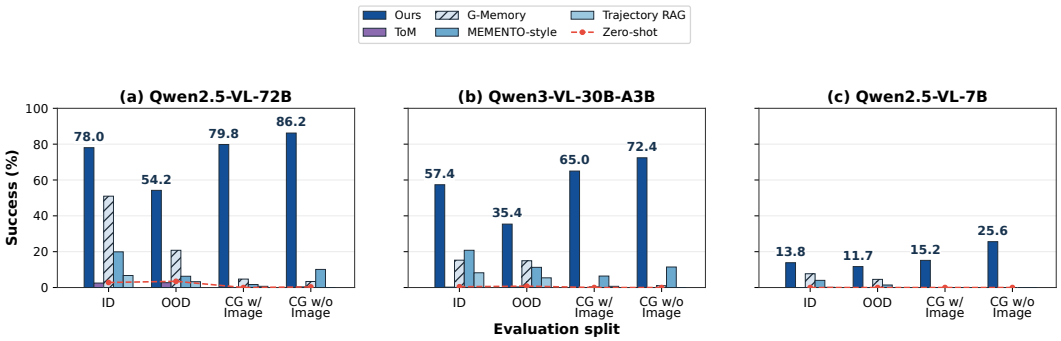

Figure 2: VIKI-L2 success on the three backbones, an end-to-end comparison in which our method grounds skill demos by role binding and every baseline writes primitive plans. OOD denotes single-family withholding, and CG denotes compositional generalization with or without image input. Numeric labels mark our method. MEMENTO-style is the port of MEMENTO to VIKI-L2. Strict OOD appears in Appendix A.3.

Baselines. Baselines are zero-shot (no memory) and ToM (Li et al., 2023) (a partner-belief module that infers the partner's intent before acting), plus three memory methods. Trajectory RAG (Lewis et al., 2020) retrieves the most similar whole trajectories into the prompt, MEMENTO (Kwon et al., 2026) retrieves episodic memory by case similarity, and G-Memory (Zhang et al., 2025a) stores multi-agent histories in a graph and retrieves relevant nodes per agent. Every memory baseline is built offline from the same traces as our library (Appendix A.1).

5.2 Transfer to Unseen Compositions and Task Families (RQ1)

Where the VIKI-L2 transfer comes from. We run four controls that change one thing each while keeping the prompt and the grounding step fixed. First, we remove the traces. A library proposed from the instructions alone passes the same gate with two skills and solves no episode on any split or backbone. The content of the library therefore comes from the traces. Second, we remove admission. The library without admission averages 21.9% over the four splits at 72B, while the admitted library averages 74.6%. The admitted library wins all 12 split-level comparisons across the three backbones (57.5% against 39.9% at 30B, 16.6% against 2.2% at 7B). The gap is smallest on the two CG splits at 30B, where the library without admission comes within a few points (Figure 5c, Appendix A.3), so admission matters most on the family splits and at 72B and 7B. Third, we remove grounding. We keep the library in the prompt but ask the LLM to write the primitive actions itself, as every baseline does (Appendix A.3). The same eight skills then reach 0.0% on CG w/ Image and 3.0% on CG w/o Image, no better than G-Memory, against 79.8% and 86.2% with grounding. Grounded use solves 237 and 247 rows that this control misses and loses none, under both the official scorer and the tolerant parser. Every primitive-plan method fails at this step on VIKI-L2. The ToM funnel in Table 5 shows that the bottleneck is writing one executable joint plan in a single pass, not naming objects or reasoning about the partner. Grounding by binding removes this step. Fourth, we replace the proposer. A library mined by replaying reference plans without any LLM, certified by the same gate, solves the same CG rows and more ID rows (Table 3). The gate, not the proposer, decides what the library contains. Three temperature-zero repeats of the 72B and 7B runs are identical row by row, and scoring with either the strict flag or the tolerant parser flips the direction of none of the 60 paired comparisons with baselines (Appendix A.1). In short, the library transfers when its content comes from traces, passes admission, and its demos are grounded by role binding.

Comparison with existing memory methods on VIKI-L2. Figure 2 compares our method with memory methods that write primitive plans from their own memory. This comparison changes both the library and how it is used, so the gap is larger than the effect of memory alone, which Figure 5, Table 3, and the PARTNR results below isolate. On the CG splits our method reaches 79.8% (w/ Image) and 86.2% (w/o Image) at 72B. G-Memory, the strongest baseline, reaches 4.7% and 3.4%. Our method reaches 65.0% and 72.4% at 30B and 15.2% and 25.6% at 7B. The CG split contains one composition pattern, so this result covers that pattern only. On ID and single-family OOD at 72B our method reaches 78.0% and 54.2%, G-Memory 51.0% and 20.8%, and ToM 2.5% and 2.7%. ToM matches zero-shot in all 12 paired cells. It raises the share of plans whose objects all exist in the

Table 1: PARTNR R_S performance on open-weight backbones, $n = 76$ episodes, mean$_{\pm\text{std}}$ over $R = 3$ runs. Succ./Comp. denote success/completion (↑). Memory methods use R-only memory. Bold marks the best mean in each column.

	Llama-3.1-8B		**Llama-3.3-70B**		**Qwen2.5-7B**		**Qwen2.5-72B**	
Method	Succ	Comp	Succ	Comp	Succ	Comp	Succ	Comp
Zero-shot	$.500_{\pm.020}$	$.692_{\pm.033}$	$.601_{\pm.030}$	$.720_{\pm.048}$	$.386_{\pm.077}$	$.593_{\pm.103}$	$.610_{\pm.033}$	$.700_{\pm.055}$
ToM	$.689_{\pm.038}$	$.727_{\pm.058}$	$.855_{\pm.013}$	$.902_{\pm.014}$	$.412_{\pm.046}$	$.568_{\pm.075}$	$.702_{\pm.015}$	$.750_{\pm.021}$
RAG	$.711_{\pm.020}$	$\mathbf{.840}_{\pm.043}$	$.794_{\pm.013}$	$.862_{\pm.019}$	$.404_{\pm.033}$	$.698_{\pm.068}$	$.750_{\pm.013}$	$.798_{\pm.017}$
MEMENTO	$.715_{\pm.026}$	$.790_{\pm.033}$	$.820_{\pm.008}$	$.875_{\pm.015}$	$.421_{\pm.030}$	$.732_{\pm.057}$	$.781_{\pm.020}$	$.815_{\pm.024}$
G-Memory	$.732_{\pm.013}$	$.785_{\pm.022}$	$.846_{\pm.013}$	$.866_{\pm.034}$	$.500_{\pm.013}$	$.695_{\pm.019}$	$.820_{\pm.023}$	$.855_{\pm.035}$
Ours	$\mathbf{.757}_{\pm.099}$	$.802_{\pm.036}$	$\mathbf{.882}_{\pm.008}$	$\mathbf{.912}_{\pm.015}$	$\mathbf{.575}_{\pm.026}$	$\mathbf{.746}_{\pm.044}$	$\mathbf{.842}_{\pm.013}$	$\mathbf{.870}_{\pm.022}$

Table 2: PARTNR transfer with Llama-3.1-8B, success rate as mean$_{\pm\text{std}}$ over $R = 3$ runs (↑), on 76 R_S, 11 H_R_T, and 5 H_R_S_T episodes. The R-only row repeats Table 1. Ours w/o adm. keeps every proposal and skips the execution check (450 and 470 skills against 29 and 31 admitted on the R_S rows). Zero-shot and ToM use no memory, so their R_S rows repeat one value. Bold marks the best mean per row.

Target	**Source memory**	**Ours**	**Ours w/o adm.**	**RAG**	**MEMENTO**	**G-Memory**	**Zero-shot**	**ToM**
R_S	R-only	$\mathbf{.757}_{\pm.099}$	$.672_{\pm.035}$	$.711_{\pm.020}$	$.715_{\pm.026}$	$.732_{\pm.013}$	$.500_{\pm.020}$	$.689_{\pm.038}$
R_S	R+S	$\mathbf{.910}_{\pm.022}$	$.711_{\pm.034}$	$.763_{\pm.020}$	$.789_{\pm.026}$	$.807_{\pm.020}$	$.500_{\pm.020}$	$.689_{\pm.038}$
H_R_T	H_R+H_T	$\mathbf{.273}_{\pm.091}$	$.182_{\pm.052}$	$.121_{\pm.052}$	$.091_{\pm.000}$	$.182_{\pm.091}$	$.000_{\pm.000}$	$.091_{\pm.091}$
H_R_S_T	R+S+H_R+H_T	$\mathbf{.400}_{\pm.000}$	$.067_{\pm.000}$	$.067_{\pm.115}$	$.200_{\pm.000}$	$.333_{\pm.115}$	$.067_{\pm.115}$	$.067_{\pm.115}$

scene from 52.7% to 64.4%, but of its 178 repaired object references only one becomes a solved plan, because the bottleneck is writing an executable joint plan (Table 5, Figure 7). Under strict OOD our method stays above G-Memory at 72B (31.9% against 14.4%) and 30B (15.9% against 11.4%) but not at 7B (3.7% against 3.3%). When G-Memory receives entries retrieved for other instructions, its single-family OOD score halves, so about half of that score comes from retrieval (Table 4).

Decentralized cooperation on PARTNR. On PARTNR only the memory changes. Every method retrieves from its own memory in every round and outputs one action, so the differences in Tables 1 and 2 come from the content of the memory. Rerunning RAG, MEMENTO, and G-Memory under our prompt with only the retrieved block replaced changes no baseline mean by more than 0.02, so the gap does not come from the prompt (Appendix A.1). Table 1 reports R_S success and completion. Our method has the highest mean success on all four backbones, Llama-3.1-8B (75.7% against 73.2% for G-Memory), Llama-3.3-70B (88.2% against 85.5% for ToM), Qwen2.5-7B (57.5% against 50.0% for G-Memory), and Qwen2.5-72B (84.2% against 82.0% for G-Memory). Except on Qwen2.5-7B, the margins are two to three points, and on Llama-3.1-8B the margin lies within one standard deviation, so we do not claim that these margins are statistically significant. On Llama-3.1-8B RAG has the higher completion (84.0% against 80.2%). On closed backbones, our method leads on Claude-3.5-Sonnet (80.0% against 78.0% for MEMENTO), while on GPT-4o every memory method scores below zero-shot (70.2% for ours against 78.7%), which shows that memory does not help every strong backbone (Appendix A.9). On R_S, success stays between 73.6% and 91.0% for k from 3 to 20, so the retrieval count matters little when the memory matches the task (Appendix A.7). Each step takes about 2.5 times as long as under MEMENTO, almost entirely because the prompt is longer, and no retrieval call in the main configurations fell back to zero-shot generation (Appendix A.7).

Constraint composition on PARTNR. Table 2 reports transfer to constraint compositions. Adding S memory to R memory raises our R_S success from 75.7% to 91.0%. G-Memory reaches 80.7%, MEMENTO 79.1%, and RAG 76.3%. Targets with three and four constraints remain hard. Our method reaches 27.3% on H_R_T, against 18.2% for G-Memory, 12.1% for RAG, and 9.1% for MEMENTO, and 40.0% on H_R_S_T, against 33.3% and 20.0%. These two rows have only 11 and 5 episodes, so we report them without a significance claim. We also repeat the admission comparison of Figure 5 on PARTNR. Keeping every proposal and skipping the execution check gives 67.2% (R-only) and 71.1% (R+S) against 75.7% and 91.0% for the admitted libraries, and 18.2% and 6.7% on H_R_T and H_R_S_T against 27.3% and 40.0%. The R-only gap lies within one standard deviation and the other three do not. Admission therefore matters on PARTNR as well.

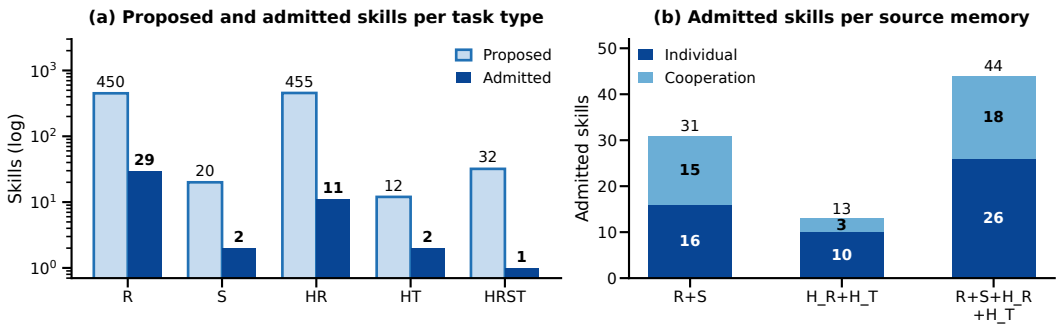

Figure 3: PARTNR library construction. (a) For each task type, the number of skills the LLM proposes and the number the admission gate keeps, on a log axis. (b) For the three combined source memories of Table 2, the admitted skills split into individual and cooperation skills, with the total above each bar.

5.3 Skill Induction under Execution-Based Admission (RQ2)

On VIKI-L2 the proposer reads 3,598 traces across 896 family and effect opportunities and submits 1,668 candidates. 23 pass the execution check, and eight remain after deduplication, with support counts from 46 down to 3 (Figure 5a, Appendix A.3). Two skill bodies are found independently by several families, one by nine. Without traces the proposer submits 16,182 candidates and the gate keeps two. Without the execution check 123 candidates are kept. The gate therefore removes most of what the proposer writes, and Figure 5b to d shows how each retained library performs. To test whether the LLM proposer matters, we also mine skills by replaying the reference plans of the same induction episodes without any LLM, certify them with the same gate, and use them under the same prompt. The gate certifies 9 of the 17 replayed candidates and rejects, among others, an Interact skill that omits the Move before it and fails every holdout execution. The certified library solves the same CG rows at 72B as the eight admitted skills (Table 3). On CG, the admitted library therefore does not depend on LLM proposal. LLM proposal is needed only when no reference plan exists to replay. Both libraries come from the induction traces without human authoring, so the gate, not the source of the candidates, determines what is admitted. We do not run a hand-written library, because it would test a human prior rather than a memory method, and we do not run replay mining on PARTNR. On PARTNR the same gate admits 45 of 969 proposals across the five task types (Figure 3a). The admission rate is higher than on VIKI-L2 because a PARTNR proposal is one short segment of an alternating-round trace, which recurs across many traces, whereas a VIKI-L2 proposal must reproduce a plan-level effect and one skill covers a whole family (Appendix A.8). The gate can also add a missing capability. Of six proposals from 239 targeted traces, one passes a 60-episode gate and raises completion from 23.3% to 90.5% on a disjoint confirmation pool (Appendix A.4).

5.4 Attribution of the Transfer (RQ3)

Ordering rules and name grounding on VIKI-L2. Figure 6 (Appendix A.3) removes two parts of the prompt that every baseline prompt also contains. w/o Order removes the ordering rules, so the LLM has to order the effects on its own. w/o Grounding removes the asset-name list, so a line whose entity names differ from the asset names binds to no admitted skill. On Qwen2.5-VL-72B, removing the ordering rules lowers ID/OOD/CG w/ Image/CG w/o Image success from 78.0/54.2/79.8/86.2% to 58.8/35.0/27.6/32.7%. Removing the asset-name list gives 41.0/17.2/0.0/86.2%. The ordering rules are worth 19 to 54 points. The asset-name list is worth 37 to 80 points on the three splits where the LLM writes names that differ from the asset names, and nothing on CG w/o Image, where the LLM already writes the asset names exactly. The 30B and 7B runs show the same pattern. On CG w/ Image at 30B, success falls from 65.0% to 55.2% without ordering rules and to 2.4% without the asset-name list (Appendix A.3).

Cooperation skills and the hierarchy on PARTNR. Figure 4 shows two views of the PARTNR library on Llama-3.1-8B. The left panel removes one part of the library at a time and keeps everything else fixed. Removing the cooperation skills (w/o Coop) causes the largest losses and drops success to zero on the hardest transfers. Flattening the hierarchy (w/o Hier) also drops four-constraint transfer to zero. Individual skills reach 44.8 grounded instances per skill, whereas cooperation skills mostly

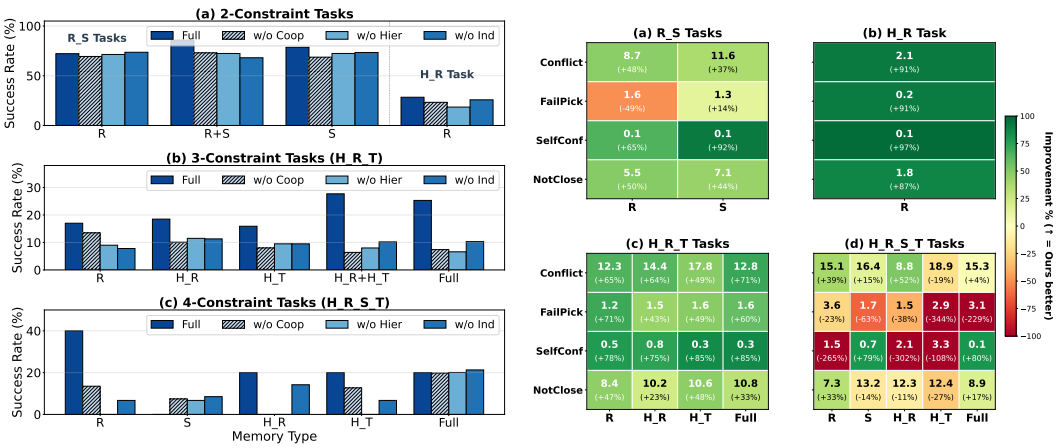

Figure 4: PARTNR library structure and failures (Llama-3.1-8B). Left, component ablation removing cooperation skills, the hierarchy, or individual skills on two-constraint, three-constraint (H_R_T), and four-constraint (H_R_S_T) tasks, with the source memory on the horizontal axis. Right, failure counts per episode (↓) for ours against MEMENTO. Cells show our count and the relative reduction (green fewer, red more), with source memories as columns (Appendix A.6).

hold 3 to 10, which is why we keep the two at separate levels (Appendix A.8). The right panel counts the four coordination failures per episode (Appendix A.6) for our library and for MEMENTO under each source memory. On R_S, H_R, and H_R_T our library has fewer failures in every cell but one, with reductions of 87% to 97% on H_R. On H_R_S_T, conflicts still fall under four of the five source memories, but failed picks and self-confusions rise when the source memory lacks the skill type the target needs. Without rearrangement skills (H_T memory) the agents attempt picks for which no instance exists. Without spatial skills (H_R memory) they move the same object back and forth between contradictory placements. The reductions are largest exactly where the source memory holds the skill type the target needs, which ties the coordination gains to the content of the library rather than to the prompt.

6 Conclusion

Memory-as-Skill turns the interaction traces of embodied agents into a memory that is worth keeping. Because each skill is written over roles and paired with grounded instances, what an agent learned in one episode carries over to new objects, rooms, and partners, and to combinations of subtasks and constraints that never appeared together in training. Because a skill enters the library only after it executes on separate episodes, the library is small, free of the proposer's mistakes, and independent of which model or procedure proposed it. Because the library holds cooperation skills alongside individual ones, decentralized agents coordinate through what they retrieve, with no messages and no partner-belief module, and they make fewer conflicting, failed, and misplaced actions than agents that retrieve raw episodes. The same library serves both an agent that acts one step at a time and a planner that writes a complete plan, and it improves every open-weight backbone we tried without changing the backbone. It is also cheap to use, costing one embedding lookup per round, and it grows incrementally, since a single admitted skill can add a capability the library was missing. Taken together, these properties make an execution-verified skill library a practical memory for multi-agent cooperation. Extending it to concurrent actions and comparing it directly with communication are the natural next steps.

Limitations. The library is built offline and frozen before evaluation, so how it grows during deployment is not measured, and all experiments run in simulation. The VIKI-L2 CG split has one composition pattern, and the three- and four-constraint PARTNR targets have few episodes, so those results are indicative rather than conclusive. The execution check on PARTNR relies on simulator state restoration, which a physical deployment would have to replace.

AI USE STATEMENT

In this work, we used generative AI tools to edit the text of the paper for readability, to condense and restructure sections to fit the page limit, and to check the manuscript for consistency of terminology and cross-references. We did not use generative AI tools to generate data, to develop the method or its conceptual framework, to formulate or prove claims, to propose hypotheses, to design the experiments, to implement the method, to clean or reformat data, or to analyze or interpret results. Large language models are components of the method under study, as the skill proposer in Section 4.1 and as the agent backbones in Section 5.1, and their role there is part of the experimental design rather than a use of AI in preparing the paper. We have reviewed all AI-assisted edits against the original text and the experimental records, and every number and claim in the paper was checked by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses two publicly released simulated benchmarks, VIKI-R and PARTNR, and open-weight or commercially available language models. It involves no human subjects, no personal data, and no data collection beyond the traces generated in simulation. The method builds a memory of executable skills for embodied agents. Its potential misuse is the same as that of any multi-robot or household planning system, and we see no risk specific to this work beyond those of the underlying benchmarks. The authors declare no conflicts of interest.

REPRODUCIBILITY STATEMENT

The method is specified in Section 4, including the admission criterion in Equation 1 and the retrieval rule in Equation 2. Section 5.1 lists the benchmarks, splits, backbones, and baselines. Appendix A.1 gives the induction budget, the proposing models, the episode pools used for induction, the ordering rules, the pairing of runs, and the prompt control. Appendix A.3 reports every VIKI-L2 control, namely the replay-mined library, the CG split audit, the primitive-output control, strict OOD, and the ToM analysis. Appendix A.4 describes the PARTNR gate calibration and its episode pools, Appendix A.5 validates the task-type classifier, Appendix A.6 defines the coordination failure metrics, and Appendix A.7 gives the retrieval threshold, the fallback rate, the cost measurements, and the sensitivity to k. All prompts are printed verbatim in Appendix A.10. Hardware is listed in Appendix A.1.

REFERENCES

Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, Siddharth Patki, Ishita Prasad, Xavier Puig, Akshara Rai, Ram Ramrakhya, Daniel Tran, Joanne Truong, John M. Turner, Eric Undersander, and Tsung-Yen Yang. Partnr: A benchmark for planning and reasoning in embodied multi-agent tasks. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=3qX2qV9Zf5.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4311–4317. IEEE, 2024.

Marco Dorigo, Guy Theraulaz, and Vito Trianni. Swarm robotics: Past, present, and future [point of view]. *Proceedings of the IEEE*, 109(7):1152–1165, 2021.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

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.

Li Kang, Xiufeng Song, Heng Zhou, Yiran Qin, Jie Yang, Xiaohong Liu, Philip Torr, Lei Bai, and Zhenfei Yin. Viki-r: Coordinating embodied multi-agent cooperation via reinforcement learning. *Advances in Neural Information Processing Systems*, 38, 2026.

Taeyoon Kwon, Dongwook Choi, Hyojun Kim, Sunghwan Kim, Seungjun Moon, Beong-woo Kwak, Kuan-Hao Huang, and Jinyoung Yeo. Embodied agents meet personalization: Investigating challenges and solutions through the lens of memory utilization. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2505.16348.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pp. 2873–2882. PMLR, 2018.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Huao Li, Yu Quan Chong, Simon Stepputtis, Joseph Campbell, Dana Hughes, Michael Lewis, and Katia Sycara. Theory of mind for multi-agent collaboration via large language models. *arXiv preprint arXiv:2310.10701*, 2023.

Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In Alice H. Oh, Brendan O'Connor, Michael Collins, Amir Globerson, Constantinos Daskalakis, Blair Adlam, Amr Ahmed, and Alekh Agarwal (eds.), *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/3941c435860b3b75b94f316212c462e0-Paper-Conference.pdf.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, et al. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. *URL https://arxiv. org/abs/2305.16291*, 2(11), 2023.

Yu Wang and Xi Chen. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S Siegel, Jiahai Feng, Noa Korneev, Joshua B Tenenbaum, and Jacob Andreas. Learning adaptive planning representations with natural language guidance. *arXiv preprint arXiv:2312.08566*, 2023.

Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

Hongkang Yang, Zehao Lin, Wenjin Wang, Hao Wu, Zhiyu Li, Bo Tang, Wenqiang Wei, Jinbo Wang, Zeyun Tang, Shichao Song, Chenyang Xi, Yu Yu, Kai Chen, Feiyu Xiong, Linpeng Tang, and Weinan E. Memory^3: Language modeling with explicit memory. *arXiv preprint arXiv:2407.01178*, 2024. URL https://arxiv.org/abs/2407.01178.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *arXiv preprint arXiv:2504.00587*, 2025.

Ruihong Zeng, Jinyuan Fang, Siwei Liu, and Zaiqiao Meng. On the structural memory of llm agents. *arXiv preprint arXiv:2412.15266*, 2024.

Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, et al. Proagent: building proactive cooperative agents with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17591–17599, 2024a.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*, 2025a.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. *arXiv preprint arXiv:2307.02485*, 2023.

Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In *International conference on machine learning*, pp. 5872–5881. PMLR, 2018.

Weinan Zhang, Junwei Liao, Ning Li, Kounianhua Du, and Jianghao Lin. Agentic information retrieval. *arXiv preprint arXiv:2410.09713*, 2024b.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 2025b. doi: 10.1145/3748302.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*, 2023. URL https://arxiv.org/pdf/2305.10250.pdf.

A APPENDIX

The appendix records the controls behind the main-text claims. Appendix A.1 gives setup details and the induction budget. Appendix A.2 describes plan-level use of the library on VIKI-L2. Appendix A.3 holds the VIKI-L2 controls, namely the replay-mined library, the CG split audit, the primitive-output control, strict OOD, and the ToM failure analysis. Appendix A.4 reports the PARTNR gate calibration, Appendix A.5 validates the PARTNR task-type classifier, Appendix A.6 defines the coordination failure metrics, Appendix A.7 reports retrieval threshold, fallback, cost, and sensitivity to k, Appendix A.8 describes the PARTNR library, Appendix A.9 reports closed backbones, and Appendix A.10 gives the prompts.

A.1 SETUP DETAILS AND INDUCTION BUDGET

Setup details. VIKI-L2 has 14 training families, and induction sees only the 3,598 even-indexed episodes of its 7,196-episode training pool. The 297 CG rows hold 295 distinct tasks. The PARTNR task-type classifier reaches mean F1 of 0.97 on elementary and 0.90 on composite types (Appendix A.5). Every PARTNR memory, including those of RAG and MEMENTO, is built offline from heuristic-agent traces of its source task type, retrieval does not exclude the current episode, and no target episode of Tables 1 and 2 occurs in its source memory. The library without admission corresponds to skill libraries whose proposing model decides what is kept (Wang et al., 2023; 2024). Memory is frozen during evaluation, and each PARTNR cell is the mean and sample standard deviation over three sampling seeds on one episode set. Scoring every VIKI-L2 method with either the strict flag or the JSON-tolerant parser alone flips the direction of none of the 60 paired comparisons between our method and a baseline. All reported VIKI-L2 cells are computed from per-episode outputs. They comprise 72 main-comparison cells, 36 cells of the library comparison, and 36 RQ3 cells, 144 in total, plus 18 strict-OOD comparison cells. Every grounded-use condition, including the RQ3 ablations, uses simulator `SOLVED`. Baselines use the JSON-tolerant parser because the official scorer rejects some otherwise valid percentage-form outputs.

Induction budget and support. Only the 3,598 even-indexed episodes from the 7,196-episode VIKI training pool are exposed to induction, and the other half is unavailable to the proposing agent. Across 14 families and 896 family and effect opportunities, the full arm, with Qwen2.5-VL-72B as the proposer, submits 1,668 candidates. Of these candidates, 1,342 have a valid type, 23 pass the execution check, and eight remain after deduplication. Their recomputed support counts are 46, 23, 23, 20, 20, 20, 7, and 3. Two skill bodies are independently induced by multiple families, including one recovered in nine families. The other six have one donor family and disappear when that family is withheld. The library induced without traces comes from 16,182 candidates proposed from the instruction alone, of which nine are admitted and two remain after deduplication, and both produce zero strict successes in every downstream cell. The library without admission retains 123 skills, so the admission gate filters quality rather than merely reducing library size.

Pairing, repeats, and condition controls. The main VIKI comparison contains all 72 expected combinations of three models, four splits, and six methods. The strict-OOD split adds 18 cells for the same three models. ToM is not run on that split, and its sixth arm is an earlier skill-memory variant of our method. Our method exceeds ToM in all 12 model and split comparisons, and ToM and zero-shot solve nearly the same rows in all 12 paired cells. Three temperature-zero repeats of the 72B and 7B full arms are identical row by row. This confirms that decoding is deterministic and does not estimate the variance from library construction, since each configuration uses one library. The 30B full arm has one run. The 30B and 7B baselines use the think condition. No matched no-think variant exists for our method, and baseline-only no-think results vary by method, so we draw no cross-condition conclusion.

Ordering rules. The three ordering rules that enter every VIKI-L2 prompt are not written by hand. The proposing LLM, Qwen2.5-VL-72B, reads the reference plans of the 3,598 induction episodes and summarizes the action order that recurs across them into three plain-text statements. The rules are therefore induced from the same pool as the skills and frozen with the library before evaluation; no CG or held-out episode contributes to them (Appendix A.3). The same three statements appear verbatim in the prompt of every baseline in Figure 2, so the 19 to 54 points they account for in

Figure 6 are shared by all methods on this benchmark and are not an advantage of ours over the baselines; the RQ3 ablation measures how much of the induced content lies in the rules rather than in the skills.

Prompt control. The memory baselines of Tables 1 and 2 use the prompt of Listing 1 with their retrieved memory in `{rag_examples}`, whereas our method uses Listing 3, which adds four structured reasoning steps and a rule block. To test whether those steps rather than the memory drive the gap, we rerun RAG, MEMENTO, and G-Memory under Listing 3 with only the retrieved block replaced, in the R_S setting of Table 1 over three seeds. No baseline mean changes by more than 0.02 in success or completion, the seed standard deviation rises slightly, and the ordering of methods is unchanged. Tables 1 and 2 therefore report the baselines under their own prompt, and the structured reasoning steps of Listing 3 do not account for the gap.

Hardware. Open-weight LLM evaluation (Llama-3.1-8B/70B, Llama-3.3-70B, Qwen-2.5-7B/72B) is performed on compute nodes equipped with $4\times$ NVIDIA GH200 Grace Hopper Superchips, each providing an H100 Tensor Core GPU with 96 GB of HBM memory, a 72-core Grace CPU with 120 GB of LPDDR memory, and NVIDIA NVLink-C2C interconnect between CPU and GPU. The 70B-class models are loaded in BF16 and sharded across multiple GPUs within a node. GPT-4o is accessed through the OpenAI API. The wall-clock measurements of Appendix A.7 were collected on this hardware.

A.2 Plan-Level Use of the Library on VIKI-L2

Some benchmarks ask for one complete plan from the instruction and the initial observation instead of one action per round. Stage 2 then runs once per episode with the same steps. The admitted library is small enough to enter the prompt whole, rendered by `name`, `effect`, and `pre`, together with the names of the entities in the scene and three ordering rules in plain text. The rules are not written by hand. The proposing LLM summarizes them from the recurring action order of the induction episodes, so they come from the same pool as the skills, and every baseline prompt on this benchmark contains the same three rules verbatim (Appendix A.1). The LLM performs skill prediction and matching by writing, for each subgoal, the `effect` of the skill it follows over named entities, for example `holding(robot_1, box_3)`. Each `effect` belongs to exactly one admitted skill, so this line identifies the skill and its binding. It is the plan-level counterpart of the `[Skill]` and `[Ref]` fields of the round-level prompt (Appendix A.10). Instance retrieval and grounding are then deterministic. The roles of the identified skill are bound to the named entities and its `demo` is written out under that binding as primitive actions. Because the plan is produced once from a complete initial observation with known entity names, the grounded demo needs no round-by-round adaptation and is emitted as written rather than rewritten by the LLM. A line that names no admitted skill or an entity outside the scene fails the episode. As on the round-level benchmark, the LLM decides which skills to apply, to what, and in what order, and the library supplies how each is carried out.

A.3 VIKI-L2 Controls

Replay-mined library. As a control for LLM proposal, we mine skills without any LLM. Replaying the reference plans of the 3,598 induction episodes and cutting them at the same subgoal boundaries yields 17 candidate skills, 13 for `pos.name` effects and 4 for `is_activated`. The same execution check and the same three criteria certify 9 of them. The rejected ones are one `Interact` skill that omits a preceding `Move` and fails every holdout execution, and seven coordination variants with no marginal gain. The certified skills enter the same prompt and grounding step. Table 3 compares this library with the admitted library on every split and backbone under strict `SOLVED`. On CG the two libraries solve the same rows at 72B and choose the same skill sequence on 297 of 297 rows. On ID the replay-mined library solves more rows, 738 against 721 at 72B, and the difference lies in one family, `dog_push_box_for_two_panda_transport` (0 of 46 against 17 of 46), where the replay-mined library holds a three-role relay skill that the admitted library lacks. The 10 rows the admitted library wins at 30B ID all fall in `clear_table` and differ only in the second answer after a re-ask. This library is a control for the proposal step, not an upper bound.

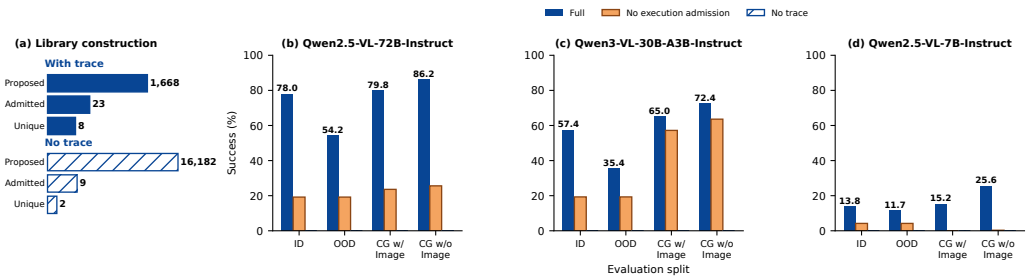

Figure 5: VIKI-L2 library construction. (a) Candidate skills proposed, admitted by the execution check, and unique after deduplication, for proposal from traces (With trace) and without traces (No trace), both with Qwen2.5-VL-72B as the proposer. Bar length is on a log scale. (b) to (d) Strict SOLVED per split on Qwen2.5-VL-72B, Qwen3-VL-30B, and Qwen2.5-VL-7B for the admitted library (Full) and the library without admission (No execution admission, 123 skills). The No trace library is zero on every split and is not drawn. CG-I and CG-T are CG with and without images.

Table 3: Admitted library against the replay-mined library under the same prompt and grounding step, strict SOLVED, rows solved out of n. Only-ours and only-replay count rows solved by one library alone.

Backbone	Split	n	Admitted	Replay-mined	Only ours	Only replay
Qwen2.5-VL-72B	ID	924	721	738	0	17
Qwen2.5-VL-72B	OOD	924	501	501	0	0
Qwen2.5-VL-72B	CG w/ Image	297	237	237	0	0
Qwen2.5-VL-72B	CG w/o Image	297	256	256	0	0
Qwen3-VL-30B	ID	924	530	568	10	48
Qwen3-VL-30B	OOD	924	327	328	0	1
Qwen3-VL-30B	CG w/ Image	297	193	196	0	3
Qwen3-VL-30B	CG w/o Image	297	215	218	0	3
Qwen2.5-VL-7B	ID	924	128	165	0	37
Qwen2.5-VL-7B	OOD	924	108	124	0	16
Qwen2.5-VL-7B	CG w/ Image	297	45	47	0	2
Qwen2.5-VL-7B	CG w/o Image	297	76	78	0	2

CG split audit. A read-only audit compares the CG tasks with every episode pool used to build the library evaluated in Figure 2, namely the seeds, holdouts, and tool calls of its 896 induction runs, the support probe, and the 3,598-episode pool from which the ordering rules are mined. All 1,242 training episodes that the proposing agent addresses through tools lie in the induction half, and none of the 896 transcripts contains a CG task identifier. Four of the 295 CG task identifiers also label training episodes, two of them in the induction half, because VIKI-L2 reuses identifiers across its training and test files. Each paired episode differs from its CG task in instruction, initial positions, goals, and reference plan, and none contains the held-out combination. The 297 CG rows hold 295 distinct tasks because two tasks occur twice. The unseen property is the combination of a two-robot cutting subtask with an independent delivery subtask. A weaker structure, a cutting subtask without the two-robot requirement combined with an independent delivery, does occur in training (1,033 rows), so we do not claim that this structure is unseen.

Primitive-output control. This arm keeps the admitted library in the prompt of Qwen2.5-VL-72B-Instruct and asks the LLM to write primitive actions directly, as every baseline does, so it differs from the main arm only in that the LLM transcribes the demos into the plan itself instead of the grounding step writing them out under a binding. It keeps the G-Memory prompt, model, decoding, and scoring of Figure 2 and replaces only the memory block with one fixed rendering of the eight skills and the three ordering rules, identical for every row. The serving host differs from the one behind Figure 2. The G-Memory and zero-shot responses are those of Figure 2. On CG w/o Image the arm succeeds on 9 of 297 rows under both the JSON-tolerant parser and the official scorer, against 10 for G-Memory (9 and 10 rows solved by only one arm) and 1 for zero-shot (8 and 0). On CG w/ Image it succeeds on no row under either criterion, against 14 for G-Memory (0 and 14) and none for zero-shot. Grounded use of the same library solves 247 and 237 rows that this arm misses and loses none. Dropping the second occurrence of the two repeated tasks changes no direction. Rerunning the

first 20 rows of CG w/o Image changes 19 of the 20 responses and their success count from 0 to 3, so single-run differences of a few rows are within run-to-run variation. The ID and single-family OOD splits are not run for this arm.

Strict OOD. Single-family withholding can leave a near-duplicate family in memory. We therefore group families whose instructions look alike before evaluation, using the cosine similarity between MiniLM means of 40 training instructions per family, with threshold 0.85 chosen at the visible gap between 0.8588 and 0.8182. No evaluated method's retrievals or scores enter grouping. All memory arms are rerun after removing the complete group. Table 4 shows that ours remains ahead of G-Memory at 72B and 30B and within four rows of it at 7B, but it drops more sharply from the single-family split. This is evidence against a favorable leakage-only interpretation, while also showing that broad cross-family support is still incomplete. The shuffled-retrieval columns are a placebo for G-Memory. Its pipeline, prompt, model, and scoring are unchanged and only the retrieved entries are permuted across instructions. Its single-family OOD success drops from 0.208 to 0.095 at 72B, from 0.149 to 0.099 at 30B, and from 0.046 to 0.020 at 7B, so retrieval accounts for roughly half of G-Memory's score on that split, and strict OOD removes most of the rest.

Table 4: VIKI-L2 success under single-family OOD and strict OOD. G-Mem. shuf. keeps the G-Memory pipeline and shuffles the retrieved entries across instructions, so each query receives the entries retrieved for another instruction.

	Single-family OOD			Strict OOD		
Model	**Ours**	**G-Mem.**	**G-Mem. shuf.**	**Ours**	**G-Mem.**	**G-Mem. shuf.**
Qwen2.5-VL-72B	0.542	0.208	0.095	**0.319**	0.144	0.080
Qwen3-VL-30B	0.354	0.149	0.099	**0.159**	0.114	0.091
Qwen2.5-VL-7B	0.117	0.046	0.020	0.037	0.033	0.026

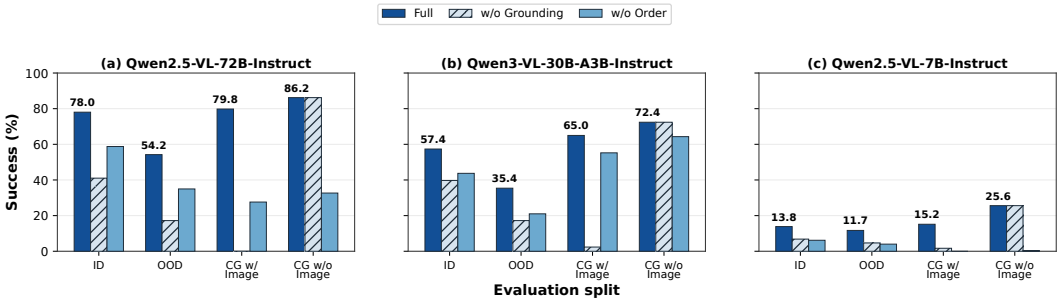

Figure 6: VIKI-L2 representation ablation on the three backbones (Section 5.4). Full is the admitted library, w/o Grounding removes the asset-name list from the prompt, and w/o Order removes the ordering rules. OOD is single-family held-out and CG denotes compositional generalization with or without images.

Why ToM does not improve VIKI-L2. The ToM and zero-shot arms share the same primitive-plan output space and differ only in the adapter prompt, so this is the valid contrast for isolating ToM. Table 5 examines the highest-scoring ToM cell. ToM improves intermediate grounding, but the advantage vanishes before final execution. Pairwise mediation gives the same diagnosis. ToM repairs 178 object-hallucination cases and breaks 56, but only one repair becomes a success. It repairs 115 length violations and breaks 165, with only six repairs becoming successes. Explicit partner-reasoning markers already occur in 99.9% of ToM and 100% of zero-shot rows. The limiting step is therefore executable joint-plan composition, not parsing, object mention, or failure to consider the partner. Because ours grounds skill demos by role binding rather than having the LLM write primitive actions, its large gap over ToM should not be read as a pure prompting effect. The library comparison of Figure 5 and the primitive-output control above hold the use fixed and are the comparisons that isolate the library.

A.4 PARTNR Gate Calibration

PARTNR uses mutually disjoint induction, 60-episode gate, 60-episode confirmation, and report pools, all selected before proposal. A 40-episode bidirectional calibration ranks a correct skill above

Table 5: Cumulative ToM/zero-shot plan funnel on Qwen2.5-VL-72B single-family OOD ($n = 924$), the highest-scoring ToM cell.

Cumulative requirement	ToM	Zero-shot
Parsed	924 (100.0%)	924 (100.0%)
All verbs in reference vocabulary	838 (90.7%)	822 (89.0%)
All objects present in scene	595 (64.4%)	487 (52.7%)
No longer than reference plan	418 (45.2%)	418 (45.2%)
Final scored success	25 (2.7%)	32 (3.5%)

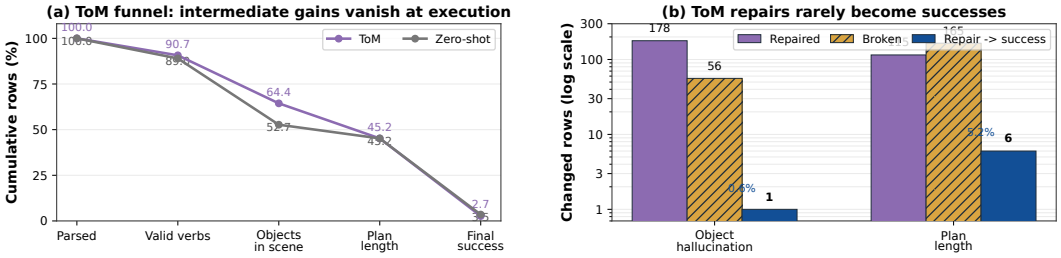

Figure 7: ToM failure analysis on Qwen2.5-VL-72B single-family OOD. (a) ToM improves intermediate vocabulary and object-validity filters, but both arms converge before final success. (b) Most ToM repairs do not become successes. Annotations show the repaired-to-success fraction.

a partially correct redundant variant. Wrong-effect, reversed, and incomplete variants yield zero or negative gain. Of six candidates, only the skill whose gate gain is 0.601 completion, with 41 episodes improved and 7 degraded, is admitted. On the disjoint confirmation pool its gain is 0.672. On 366 scored episodes from the 369-episode report pool its gain is 0.109, almost entirely on the 68 `is_in_room` episodes, where it is 0.604, and it is 0.004 below zero on the other 298 (Figure 8). An identical-configuration repeat gives the same numbers. These measurements use privileged simulator state restoration. Two attempted 7B/8B end-to-end model-agent reruns were incomplete because of endpoint and timeout failures. We exclude their partial rows and do not extrapolate the reported effect to those agents.

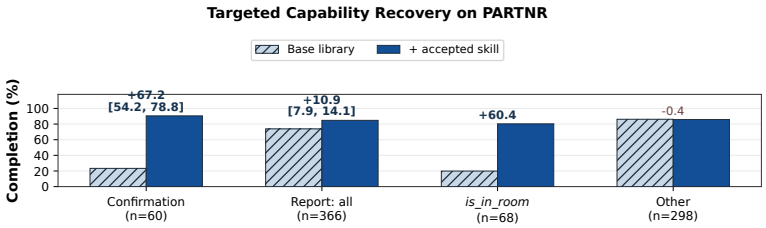

Figure 8: PARTNR targeted recovery after admitting one induced skill. Completion before and after admission on the confirmation pool, the full report pool, the `is_in_room` episodes, and the remaining episodes. The bracket above each pair marks the gain.

A.5 Task Type Classification Validation

We compare proposition-based task-type predictions with ground-truth labels on 2,000 episodes from the validated PARTNR training set (Chang et al., 2025). Figure 9 reports per-type F1. Elementary types (R, S, H, T) average 0.97, and composite types (R_S, H_R, H_R_T, H_R_S_T) average 0.90. H_R_S_T is lower and contains only five episodes. Overall exact-match accuracy is 90.9%. The predicted type decides which run configuration, and therefore which source memory, the episode is evaluated with (Section 4.2). A misclassified episode is therefore evaluated with the source memory of its predicted type. To measure the cost of such errors, we regroup the same episodes by their ground-truth labels and rerun Llama-3.1-8B. Table 6 reports completion across 22 matched episodes. Ground-truth grouping improves aggregate completion by 0.039. It leaves R_S unchanged, raises H_R by 0.095 and H_R_T by 0.133, and lowers H_R_S_T by 0.111. Given the small per-task samples, these values characterize headroom rather than establish a systematic effect. Ground-truth grouping is not deployable because the type must be inferred from the instruction, but this comparison suggests that classification is not the dominant bottleneck relative to the ablations in Figure 4.

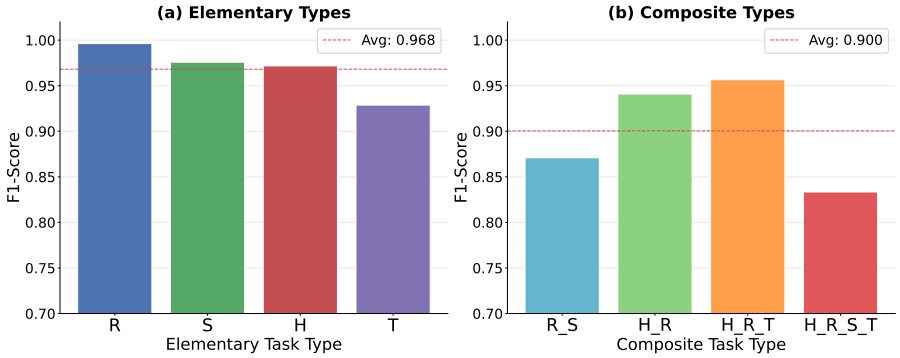

Figure 9: Task-type classification on PARTNR. (a) Elementary types. (b) Composite types. H_R_S_T has only five episodes.

Table 6: Classifier and ground-truth (GT) task-type grouping on Llama-3.1-8B. Aggregate completion differs by 0.039. Gains are computed from unrounded values and can differ by 0.001 from subtraction of the displayed entries.

Task	Classifier Comp.	GT Comp.	Gain
R_S	0.875	0.875	+0.000
H_R	0.786	0.881	+0.095
H_R_T	0.267	0.400	+0.133
H_R_S_T	0.437	0.325	−0.111
All	0.676	0.716	+0.039

A.6 Coordination Failure Metrics

We define four coordination failure metrics to quantify the quality of multi-agent interaction during cooperative task execution. All metrics are computed per episode and averaged across evaluation episodes. Lower values indicate better coordination.

A.6.1 Inter-Agent Conflict (Conflict/Ep)

Inter-agent conflicts occur when multiple agents target the same object within one coordination round under PARTNR's sequential execution scheme, causing resource contention (e.g., one agent attempts to pick an object just claimed by another). Let a_t^i be agent i's action in round t and $\text{target}(a_t^i)$ its target object. We detect a conflict under the following condition.

$$\mathbf{1}_{\text{conflict}}(t) = \mathbf{1}\left[\exists\, i \neq j \text{ such that } \text{target}(a_t^i) = \text{target}(a_t^j) \wedge a_t^i, a_t^j \in \mathcal{A}_{\text{manip}}\right] \tag{3}$$

where $\mathcal{A}_{\text{manip}} = \{\texttt{pick}, \texttt{place}, \texttt{open}, \texttt{close}\}$ denotes the set of manipulation actions. The conflict rate per episode is defined below.

$$\text{Conflict/Ep} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \sum_{t=1}^{T_e} \mathbf{1}_{\text{conflict}}(t) \tag{4}$$

where $\mathcal{E}$ is the set of evaluation episodes and T_e is the episode length.

A.6.2 Failed Pick Attempts (FailPick/Ep)

Failed pick attempts measure unsuccessful object grasping due to incorrect positioning, unexpected object state changes, or interference from other agents. A failed pick is recorded when an agent executes a `pick` action but the object is not successfully added to the agent's inventory.

$$\mathbf{1}^i_{\text{failpick}}(t) = \mathbf{1}\left[a_t^i = \texttt{pick}(o) \wedge o \notin \text{inventory}^i(t+1)\right] \tag{5}$$

The metric aggregates across all agents as follows.

$$\text{FailPick/Ep} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \sum_{t=1}^{T_e} \sum_{i=1}^{N} \mathbf{1}^i_{\text{failpick}}(t) \tag{6}$$

where N is the number of agents.

A.6.3 SELF-CONFUSION ERRORS (SELFCONF/EP)

Self-confusion errors capture inconsistent decision-making where a single agent issues contradictory actions on the same target object within a short time window τ. This is distinct from normal task progression. In successful transport, the pick and the subsequent place act on different effective targets within the world model because the object is picked from a source location and placed at a destination. A self-confusion event instead flags repeated reversals on the same object-location pair (e.g., picking an object and placing it back at its original source within τ steps), indicating conflicting skill guidance rather than productive manipulation. We define contradictory action pairs below.

$$\mathcal{C} = \{(\texttt{pick}(o), \texttt{place}(o)), (\texttt{open}(o), \texttt{close}(o))\} \tag{7}$$

A self-confusion event is detected under the following condition.

$$\mathbf{1}^i_{\text{selfconf}}(t) = \mathbf{1}\left[\exists\, t' \in [t-\tau, t) \text{ such that } (a^i_{t'}, a^i_t) \in \mathcal{C} \vee (a^i_t, a^i_{t'}) \in \mathcal{C}\right] \tag{8}$$

where $\tau = 5$ timesteps in our experiments. The metric is computed as follows.

$$\text{SelfConf/Ep} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \sum_{t=1}^{T_e} \sum_{i=1}^{N} \mathbf{1}^i_{\text{selfconf}}(t) \tag{9}$$

A.6.4 NAVIGATION FAILURES (NOTCLOSE/EP)

Navigation failures occur when an agent attempts a manipulation action without being sufficiently close to the target object, reflecting poor spatial awareness or coordination timing. Let $\text{pos}^i(t)$ denote the position of agent i at timestep t, and $\text{pos}(o)$ denote the position of object o. A navigation failure is recorded under the following condition.

$$\mathbf{1}^i_{\text{notclose}}(t) = \mathbf{1}\left[a^i_t \in \mathcal{A}_{\text{manip}} \wedge \|\text{pos}^i(t) - \text{pos}(\text{target}(a^i_t))\|_2 > d_{\text{thresh}}\right] \tag{10}$$

where $d_{\text{thresh}} = 1.5$ meters is the interaction distance threshold. The metric is defined below.

$$\text{NotClose/Ep} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \sum_{t=1}^{T_e} \sum_{i=1}^{N} \mathbf{1}^i_{\text{notclose}}(t) \tag{11}$$

A.6.5 SUMMARY

Table 7 summarizes the four coordination failure metrics and their interpretations.

Table 7: Summary of coordination failure metrics.

Metric	Failure Type	Indicates
Conflict/Ep	Object contention	Poor task allocation
FailPick/Ep	Grasping failure	State estimation errors
SelfConf/Ep	Action contradiction	Inconsistent guidance
NotClose/Ep	Distance violation	Spatial unawareness

A.7 RETRIEVAL THRESHOLD, FALLBACK, COST, AND SENSITIVITY

Threshold. The main results use one fixed setting, $\theta = 0.3$ and $k = 5$, across all backbones, memory configurations, and splits. As a sensitivity check around this operating point, we embed task instructions and skill descriptions with `all-mpnet-base-v2`, retrieve the top 20 candidates by cosine similarity, ask four LLM judges (Llama-3.1-8B, Llama-3.1-70B, Qwen-2.5-7B, and Qwen-2.5-72B) to select the five most relevant, and sweep θ from 0.2 to 0.6 against these labels. Judge-optimal thresholds lie at 0.2 for R_S and H_R and between 0.4 and 0.45 for H_R_T and H_R_S_T, and the judges agree with one another. Using the judge-specific thresholds instead of $\theta = 0.3$ would change per-task fallback rates by at most a few tenths of a percentage point, because the retrieval scores logged in the main configurations have medians above 0.8 and minima above 0.3. This analysis reuses the main evaluation episodes and is a sensitivity analysis, not held-out tuning.

Fallback activation rate. We instrument the retrieval pipeline and log every retrieval call in the main experiments. Across $8{,}855$ retrieval calls under the matched-memory configurations used for the main results, the fallback to retrieval-free generation triggered in 0 cases. The minimum logged scores were 0.488 on R_S, 0.343 on H_R, 0.845 on H_R_T, and 0.580 on H_R_S_T, and the medians on R_S and H_R were 0.836 and 0.813. On a broader evaluation set spanning more diverse memory configurations ($13{,}795$ retrieval calls), the overall fallback rate is 0.55%, concentrated on H_R_S_T (2.65%, $27/1{,}020$) where library coverage is most stretched. To test that fallback activates when memory is mismatched, we use S-only memory on H_R_S_T, where it triggers on 27 of 30 retrieval calls, and on R_S, where it rises to 3.8% (49/1,283). The near-zero rate in the main configurations reflects close alignment between the library and the PARTNR task distribution.

Cost. Per agent, each step costs one retrieval of $\mathcal{O}\big((|\mathcal{L}| + K) \cdot d + k\big)$ and one LLM call C_{LLM}, where $|\mathcal{L}|$ is the abstract-skill library size, K is the number of grounded instances within the matched skill, d is the embedding dimension, and k is the retrieval count. No term depends on agent count N, and per-round wall-clock time scales as $\mathcal{O}(N \cdot C_{\text{LLM}})$ under sequential execution, in contrast to the $\mathcal{O}(N^2 \cdot C_{\text{LLM}})$ overhead of communication-based approaches. Measured on Llama-3.1-8B (H_R tasks, $1{,}270$ steps for our method and 845 for MEMENTO), one retrieval call averages 54 ms, under 1% of step latency. Total per-step latency averages 9.4 ± 9.0 s for our method against 3.7 ± 1.8 s for MEMENTO. Action generation accounts for over 99% of step time in both methods, so the roughly $2.5\times$ increase comes from the longer prompt that the $k = 5$ demonstrations produce, not from retrieval. Against the strongest baseline this cost buys two to three points of success on R_S and ten points on the R+S composition (Tables 1 and 2), so whether it is worth paying depends on whether a deployment needs the composition transfer. Reducing k lowers prompt cost and may reduce success (Figure 10).

Sensitivity to k. Figure 10 reports retrieval-count sensitivity over $k \in \{3, 5, 10, 20\}$ on Llama-3.1-8B. For aligned R_S tasks, success stays between 73.6% and 91.0% across all k, so when memory content matches the task the retrieved count matters little. Sensitivity increases under harder transfer and memory misalignment. With R-only memory, H_R_S_T reaches 40.0% at k=5 and 15.4% at k=10, consistent with a trade-off between coverage and interference. Completion is less sensitive to k than success, so retrieval count mainly affects whether all constraints are satisfied at once. The main experiments fix $k = 5$; this analysis shows how results move around that setting and is not used to select it.

A.8 PARTNR LIBRARY

The admission rate on PARTNR, 45 of 969, is higher than on VIKI-L2, 8 of 1,668. The gate is the same on both benchmarks. What differs is the unit of proposal. A PARTNR proposal is one segment of an alternating-round trace, so distinct short behavior units such as a navigate-and-pick or a wait-then-grasp recur across many traces and pass the recurrence criterion. A VIKI-L2 proposal must reproduce a plan-level effect after rebinding, and one skill covers a whole task family, so few distinct skills exist to admit. Decentralized alternating execution also draws on a wider set of behavior units per episode, since each agent plans one action at a time from its own observation, so the PARTNR library needs more admitted skills than the eight that suffice for complete VIKI-L2 plans. The hierarchy contains individual skills ($\mathcal{L}_{\text{ind}}$), which represent single-agent primitives, and cooperation skills ($\mathcal{L}_{\text{coop}}$), which represent multi-agent coordination patterns.

Reuse, measured as the mean number of grounded instances per abstract skill, separates the two levels (Figure 11). Individual skills reach 44.8 instances per skill, whereas cooperation skills range from 1.0 on data-sparse H_T and H_R_S_T to 24.5, with most between 3 and 10 instances on R and H_R. The lower reuse of cooperation skills is consistent with their dependence on agent roles and spatial context, and it motivates the hierarchy: individual skills act as broadly reused building blocks, whereas cooperation skills retain more task-specific coordination context. Of the stated preconditions, the instance filter checks only the holding state in `self_cond` (Section 4.2), and the others reach the LLM as text inside the demonstration. For a multi-object instruction such as "move jug, kettle, and teapot to kitchen," top-k retrieval (k=5) surfaces three complementary individual skills covering each object class plus cooperation skills capturing the handover patterns required for joint transport, rather than redundant variants of a single skill.

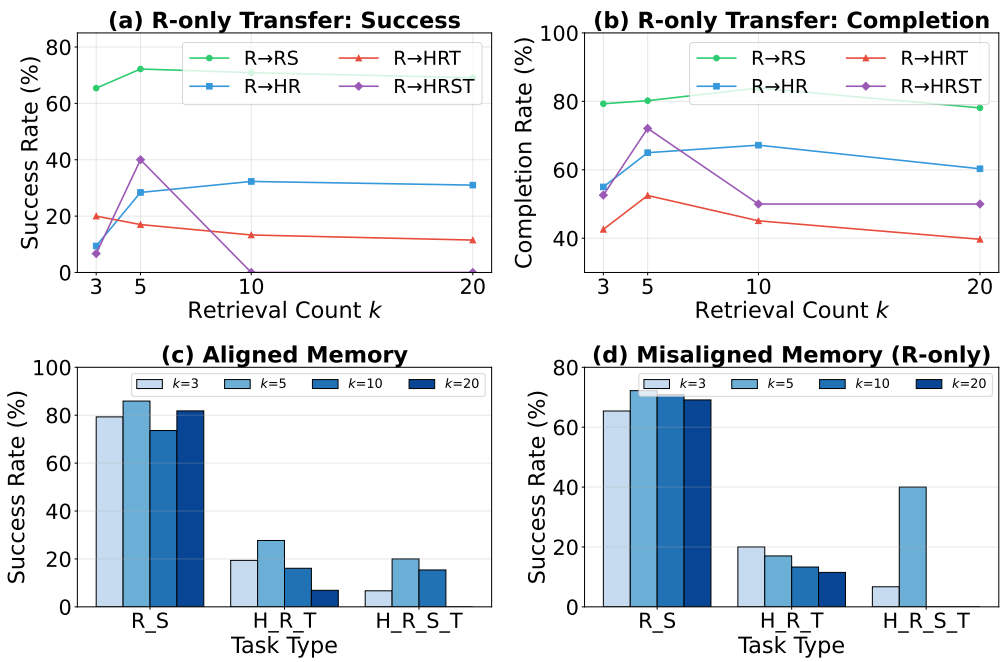

Figure 10: Effect of retrieval count k on transfer performance, evaluated on Llama-3.1-8B. Panels (a) and (b) show success and completion as functions of k for R-only memory transfer to four target task complexities. Panel (c) uses aligned memory whose types match task requirements. Panel (d) uses R-only memory for all task types. Across panels, k=5 is the most consistent operating point, and k=20 degrades complex transfers as excess retrieval introduces conflicting guidance.

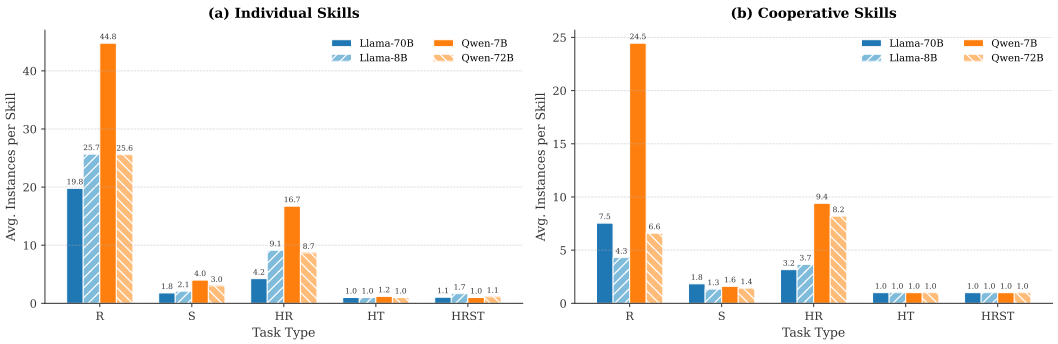

Figure 11: Mean instances per abstract individual skill (left) and cooperation skill (right), for four proposing backbones and five source task types. Cooperation-skill reuse ranges from 1.0 to 24.5 instances, with most R and H_R values between 3 and 10.

A.9 Closed Backbones

Table 8 reports Claude-3.5-Sonnet and GPT-4o in the setting of Table 1. On Claude-3.5-Sonnet our method leads the four available baselines by two points. On GPT-4o our method has the highest completion rate (86.7%, against 85.9% for ToM and 84.3% for zero-shot) but lower success than zero-shot (70.2% against 78.7%), and RAG and MEMENTO fall lower still. Completion counts the fraction of goal propositions satisfied, whereas success requires every constraint to be satisfied, so a method can improve partial progress while reducing strict success. GPT-4o has the strongest zero-shot success of all six backbones, and retrieved demonstrations may bias it toward an action that worked in a similar but nonidentical context. The current aggregates do not identify that mechanism causally, but they show that memory augmentation should be evaluated on both success and completion rather than assumed to help every strong backbone. G-Memory has no completed run on either backbone.

Table 8: PARTNR R_S performance on the two closed backbones, same setting as Table 1. N/A marks a cell without a completed run.

	Claude-3.5-Sonnet		GPT-4o	
Method	Succ↑	Comp↑	Succ↑	Comp↑
Zero-shot	0.620	0.700	<u>0.787</u>	0.843
ToM	0.710	0.740	**0.816**	<u>0.859</u>
RAG	0.755	0.815	0.658	0.848
MEMENTO	<u>0.780</u>	<u>0.840</u>	0.631	0.845
G-Memory	N/A	N/A	N/A	N/A
Ours	**0.800**	**0.850**	0.702	**0.867**

A.10 PROMPT TEMPLATE

Listings 1 to 3 give the PARTNR prompts. Listing 1 is used by zero-shot and, with `{rag_examples}` filled from their own memory, by RAG, MEMENTO, and G-Memory; Listing 2 is used by ToM; Listing 3 is used by our method and by the prompt control of Appendix A.1. In the prompt of our method, `{rag_examples}` holds the demonstrations retrieved in the current round, and `{state_comparison}` holds the observation changes since the agent's previous turn. Both are refreshed every round, and action receipts are appended. The `[Diff]`, `[Skill]`, and `[Ref]` fields are filled by the LLM in its own output and do not trigger retrieval.

Listing 1: Zero-Shot Prompt

```
prompt: |-
      {system_tag}You are an agent that solves multi-agent planning problems.
      The task assigned to you will be situated in a house
      and will generally involve navigating to objects,
      picking and placing them on different receptacles to achieve rearrangement.
      You strictly follow any format specifications
      and pay attention to the previous actions taken
      in order to avoid repeating mistakes.

      There will be another agent
      trying to solve the same task that you are at the same time.
      You may find that that agent has picked up relevant objects
      or is in the process of completing parts of the task.
      If that is the case you may want to
      move on to a different part of the task.

      Rooms do not need to be explored more than once.
      This means if you have explored the living room and have not found the object,
      then you should explore the kitchen,
      if a relevant object is still not found,
      you should explore the hallway etc...

      {agent_role_description}

      Many calls to the same action in a row are a sign
      that something has gone wrong and
      you should try a different action.{eot_tag}{rag_examples}{user_tag}Task: {input}

      {world_description}

      Possible Actions:
      {tool_descriptions}
      - Done: Used to indicate that the agent has finished the task. Example (Done[])

      What is the next action to make progress towards completing the task?
      Return your response in the following format

      Thought: <reasoning for why you are taking the next action>
      <next action call>
      Assigned!

      Here is an example:
      Thought: Since there are no objects found
      I should explore a room I have not explored yet
      Explore[<room name>]
      Assigned!
      {eot_tag}{assistant_tag}

stopword : "Assigned!"
```

```
end_expression : "Done[]"
```

Listing 2: ToM Prompt

```
prompt: |-
   {system_tag}You are an agent that solves multi-agent planning problems.
      The task assigned to you will be situated in a house
      and will generally involve navigating to objects,
      picking and placing them on different receptacles to achieve rearrangement.
      You strictly follow any format specifications
      and pay attention to the previous actions taken
      in order to avoid repeating mistakes.
      Rooms do not need to be explored more than once.

   **Collaboration with Another Agent:**
   There will be another agent trying to solve the same task as you.
   Your success depends on effective collaboration.
   You must observe their actions to infer their intentions, plans,
   and knowledge to avoid redundant work and efficiently divide the task.

   **Theory of Mind (ToM) for Collaboration:**
   To collaborate effectively, use the following structured reasoning process
   in your "Thought" block before deciding on an action.
   This mimics a dual-process reasoning system,
   combining long-term strategic thinking with immediate observation.

   * **Step 1: Belief Formation (What do they know and what are their habits?)**
      * Summarize the other agent's recent significant actions and their location.
      * Based on their location and path,
      what have they likely observed? What do they probably know or not know?
      * Are there any recurring patterns or tendencies in their behavior?

   * **Step 2: Hypothesis Generation (What is their current plan?)**
      * Based on your belief,
      form a primary hypothesis about
      the other agent's immediate subgoal or overall strategy.

   * **Step 3: Prediction & Planning (What will they do, and what should I do?)**
      * Based on your best hypothesis,
      predict the other agent's most likely next action(s).
      * Given your prediction, formulate your own plan.
      Your plan should complement their work, not conflict with it.
      Decide whether to tackle a different sub-task,
      prepare the next step for them, or explore a new area.

   **General Guidelines:**
   * Avoid repeating mistakes. If an action fails,
   analyze why and try a different approach.
   * Rooms do not need to be explored more than once.
   If you have explored the living room and have not found an object,
   move on to another unexplored room.
   * Many calls to the same action in a row
   are a sign that something has gone wrong and
   you should try a different action.

   {agent_role_description}

   {eot_tag}{user_tag}Task: {input}

   {world_description}

   Possible Actions:
   {tool_descriptions}
   - Done: Used to indicate that the agent has finished the task. Example (Done[])

   What is the next action to make progress towards completing the task?
   Return your response in the following format.
   Use the structured ToM reasoning in your thought process.

   Thought: <MUST follow the 3-step ToM reasoning process described above>
   <next action call>
   Assigned!

   Here is an example of the desired format:
   Thought:
   1. **Belief Formation:** The other agent was last seen moving into the kitchen.
   They have already explored the living room.
   They likely know what objects are (or are not) in the living room.
   2. **Hypothesis:** My primary hypothesis is that
   they are now searching for the apple in the kitchen.
```

```
   3. **Prediction & Planning:** They will likely pick up the apple if they find it.
   To collaborate efficiently and not duplicate effort,
   I will not go to the kitchen. Instead,
   I will search for the other required object,
   the television remote, in a room that neither of us has explored yet,
   which is the bedroom.
   Explore[bedroom]
   Assigned!
   {eot_tag}{assistant_tag}

stopword : "Assigned!"
end_expression : "Done[]"
```

Listing 3: Ours Prompt

```
prompt: |-
   {system_tag}You are an agent solving household tasks with another agent.
   Agents act sequentially:
   you observe the updated environment after your partner's action,
   then decide yours.

   {agent_role_description}

   ## Decision Process

   Before each action, follow these steps:

   **1. Observation Diff** What changed since last step?
   Identify object displacements,
   partner state changes, and property modifications.

   **2. Skill Prediction** What type of skill do I need?
   - **Individual skill**: act alone (navigate, pick, place, clean, fill, power on/off)
   - **Cooperation skill**: coordinate with partner
   (division of labor, handoff, complementary work, conflict avoidance)

   **3. Skill Matching** Which retrieved example applies?
   Match your situation to a retrieved skill below.
   Apply its strategy, adapting to your environment.

   **4. Action Selection** Choose an action complementary to partner's progress.

   ## Rules
   - Do NOT re-explore rooms. Move to unexplored rooms.
   - Repeating same action = something wrong. Try different action.
   - Temporal tasks ("then","after"): complete prerequisites first.
   - Done[] only when ALL requirements verified complete.

   ===== RETRIEVED SKILL EXAMPLES (Individual + Cooperation) =====
   {eot_tag}{rag_examples}{user_tag}
   ===== END =====

   Task: {input}

   {world_description}

   {state_comparison}

   Actions: {tool_descriptions}
   - Done[]

   Thought:
   [Diff] <what changed; what partner did>
   [Skill] <individual/cooperation>: <type>
   [Ref] <matched skill name and why>
   [Plan] <action rationale>
   <action>
   Assigned!
   {eot_tag}{assistant_tag}

stopword : "Assigned!"
end_expression : "Done[]"
```