

MemoGuard: An Adaptive Runtime for Guarding Against Memory Traps in Communication-Limited Robot Navigation

Rajat Bhattacharjya^{†*}, Hyeonjong Ju^{‡*}, Sing-Yao Wu[†], Eli Bozorgzadeh[†], Nikil Dutt[†]

[†]Department of Computer Science, University of California, Irvine

[‡] Department of Computer Science, Yonsei University, South Korea

Abstract—Communication-limited robots in mission-critical scenarios such as disaster inspection and search-and-rescue must make reliable onboard decisions without access to remote operators or high-capacity reasoning services. Episodic memory reuse is an attractive low-cost fallback, but retrieval similarity does not guarantee execution validity, i.e., a retrieved action may match the current context yet be unsafe due to changed topology, insufficient battery margin, or unreliable prior outcomes. We call such high-similarity but execution-invalid episodes *memory traps*. This creates a safety-efficiency design space where similarity only reuse minimizes fallback cost but can be unsafe, while always invoking local reasoning improves safety at high computational and energy cost. This paper presents MemoGuard, a lightweight adaptive runtime that validates episodic memories against topology, resource, and outcome contracts before reuse, invoking fallback only when validation fails. In a graph-based corridor-inspection simulator, MemoGuard reduces battery safety violations by 76.6% over similarity-only top-1 reuse while reducing fallback calls by 21.4% over always reasoning. On an NVIDIA Jetson AGX Xavier with local llama3.2:3b fallback reasoning, this corresponds to 3.67 s and 36.97 J of avoided fallback-reasoning overhead per trial. We open-source MemoGuard at <https://github.com/hetheiin/memoguard>.

Index Terms—Robot navigation, embodied AI, episodic memory, mission-critical systems, safety-efficiency tradeoff

I. INTRODUCTION

Mobile robots operating in mission-critical settings such as disaster inspection and search-and-rescue increasingly rely on operator-specified mission intent to guide autonomous behavior [1], [2]. In practice, however, these robots may experience degraded or intermittent communication with remote operators, cloud planners, or high-capacity reasoning services [1], [3]. During such intervals, the robot must make onboard decisions with limited compute, energy, and sensing reliability [1], [4]. A natural fallback is case-based episodic reuse: retrieving a past execution episode that resembles the current situation and reusing or adapting the associated action sequence [5], [6].

This strategy is appealing for embedded robots because memory lookup and lightweight validation can be much cheaper than invoking a local VLM [7] or large reasoning

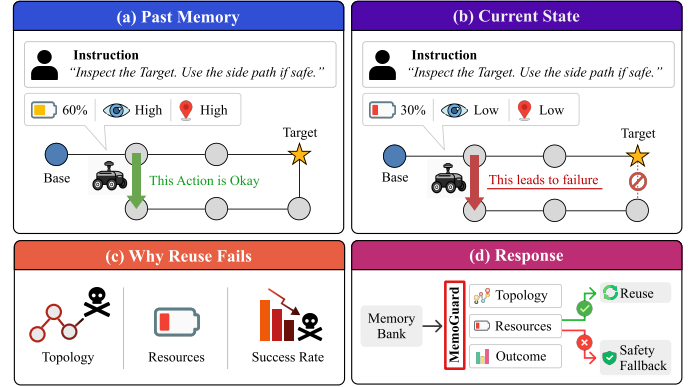


Fig. 1. Motivating memory trap under communication loss. (a) A past memory records a successful action for the mission intent: inspect target T using the side path if safe. (b) The current state has the same intent and similar local context, but changed topology, resource, and localization conditions make the remembered action unsafe. (c) Similarity-based reuse fails by executing the stale action. (d) MemoGuard rejects the unsafe memory by validating topology, resource feasibility, and outcome reliability before fallback.

module [8]. However, embodied memory reuse is not only a retrieval problem. Fig. 1 illustrates this problem in a corridor-inspection mission: a robot may retrieve a past episode matching the current intent, visibility, and blockage context, yet the remembered action may be invalid because the side route is blocked, the battery margin is insufficient, or localization confidence has degraded. A memory can be semantically relevant while its execution assumptions no longer hold.

Prior case-based and episodic-memory approaches show that past experiences can guide action selection [5], [6], but reuse in resource-constrained settings is dominated by retrieval similarity while execution assumptions remain implicit. This leaves a systems gap: before executing a retrieved action, the robot must verify that those assumptions still hold under the current topology, resource trajectory, and outcome history. We call high-similarity memories that fail this test *memory traps*.

This paper presents MemoGuard, a lightweight runtime for contract-validated episodic memory reuse in communication-limited embodied robots navigating the safety-efficiency design space. MemoGuard treats each memory as an *episodic action memory*: a compact record containing the mission context, pre-state, remembered action, execution contract, and outcome statistics. At runtime, MemoGuard validates candidates using hard feasibility gates over topology, resource feasibility, and outcome reliability, invoking a planner or local reasoning (LLM) fallback only when validation fails.

* Equal contribution. R. Bhattacharjya is the corresponding author (e-mail: rajatb1@uci.edu).

Paper accepted at ESWEK CODES 2026 in the Late Breaking Results (LBR) track. Authors' version posted for personal use and not for redistribution. The definitive version of the paper will appear in IEEE Embedded Systems Letters.

We make the following contributions:

- We identify **memory traps** and design MemoGuard, a lightweight contract-validation runtime that guards against unsafe episodic reuse via hard feasibility gates and outcome-reliability scoring over a bounded supervisory action set.
- We evaluate MemoGuard in a graph-based corridor-inspection simulator across three topologies with blocked routes, altered viewpoint affordances, and resource constraints.
- We calibrate per-call fallback-reasoning overhead on an NVIDIA Jetson AGX Xavier, showing that reduced fallback frequency translates to 3.67 s and 36.97 J of avoided LLM fallback overhead per trial in MODE_30W_ALL.

II. MEMORY TRAPS AND PROBLEM FORMULATION

We now formalize the failure mode introduced in Fig. 1. We model communication-limited robot navigation as a supervisory decision problem in which a robot may either reuse a retrieved episodic memory or invoke a higher-cost planner/reasoning fallback. The core challenge is that retrieval similarity and execution validity are different properties: a memory may appear relevant while its remembered action is no longer executable.

(A) Episodic Action Memories: We model the operating environment as a topological graph $G_t = (V_t, E_t)$, where nodes represent locations (base, corridor, target, safe waypoint, alternate viewpoint) and edges encode traversal cost, obstacle state, and local visibility. At decision step t , the robot state s_t includes the current node, target, battery, localization confidence, communication state, mission intent, and local observations. We focus on communication-limited intervals in which the robot cannot reliably access remote operators, cloud planners, or high-capacity reasoning services.

The robot acts through a bounded supervisory action set:

$$\mathcal{A} = \{\text{follow_planner}, \text{inspect_target}, \text{return_to_safe_waypoint}, \text{inspect_alternate_viewpoint}, \text{wait_for_recovery}, \text{abort_mission}\}. \quad (1)$$

These actions cover path following, target inspection, recovery, and mission termination. Reasoning is not itself an executable action; the planner/reasoning fallback selects one action from $\mathcal{A}$.

An episodic action memory is a compact record of a prior execution event:

$$m_i = (\iota_i, x_i, a_i, \kappa_i, \rho_i), \quad (2)$$

where ι_i is the intent context; x_i is the retrieval key (current node, target, visibility, obstacle state, edge cost, battery range, localization confidence); $a_i \in \mathcal{A}$ is the remembered action; κ_i is the execution contract (topological preconditions, viewpoint availability, expected action cost, path-risk profile, and safety constraints); and ρ_i stores success and failure counts for equivalent state-action memories.

(B) Top- k Retrieval and Memory Traps: Given current state s_t , the memory retriever first filters memories by intent context, current node, target, and mission phase, producing an

eligible set $\mathcal{M}_t$. It then ranks eligible memories by a retrieval similarity score $S_{\text{ret}}(s_t, m_i)$ and returns the top- k candidate set¹:

$$\mathcal{C}_t^k = \text{TopK}_{m_i \in \mathcal{M}_t} S_{\text{ret}}(s_t, m_i). \quad (3)$$

A memory trap occurs when a retrieved memory appears similar to the current state but is invalid for execution. Let $\Phi(s_t, m_i) \in \{0, 1\}$ denote whether memory m_i is valid for reuse at state s_t . We define the top- k memory-trap set as

$$\mathcal{T}_t^k = \{m_i \in \mathcal{C}_t^k : \Phi(s_t, m_i) = 0\}. \quad (4)$$

Validity depends on whether the remembered action remains compatible with the current mission, topology, resource state, and outcome history:

$$\Phi(s_t, m_i) = \Phi_{\text{intent}}(s_t, m_i) \wedge \Phi_{\text{topo}}(s_t, m_i) \wedge \Phi_{\text{res}}(s_t, m_i) \wedge \Phi_{\text{out}}(m_i). \quad (5)$$

Thus, a memory trap is not necessarily an irrelevant memory. It may match the current state semantically or locally while still being unsafe because its execution assumptions no longer hold, e.g., due to changes in the physical environment or context.

(C) Runtime Objective: The runtime objective is to select a supervisory action during communication-limited execution. Given current state s_t , memory bank $\mathcal{M}$, and retrieved candidate set $\mathcal{C}_t^k$, the robot should reuse a remembered action only if at least one candidate passes validation:

$$a_t = \begin{cases} a_i, & \exists m_i \in \mathcal{C}_t^k \text{ such that } \Phi(s_t, m_i) = 1, \\ f(s_t), & \text{otherwise,} \end{cases} \quad (6)$$

where $f(s_t)$ denotes the local planner, local reasoning fallback, or safe supervisory policy. Similarity-only reuse implicitly assumes that the highest-scoring memory is executable. MemoGuard instead validates candidate memories before reuse, rejecting memory traps whose execution contracts no longer hold.

III. MEMOGUARD RUNTIME

MemoGuard (Fig. 2) is a lightweight runtime layer between episodic memory retrieval and action execution. Fig. 2(b) shows the episodic memory format and Fig. 2(c) illustrates the runtime flow: given the current robot state and memory bank, MemoGuard retrieves candidate memories, validates their execution contracts, and either reuses a remembered supervisory action or invokes a planner/reasoning fallback.

(A) Candidate Retrieval: At each decision step, MemoGuard masks actions inconsistent with the current mission phase and retrieves memories matching the current intent context, node, target, and mission phase; if the robot is already at the target or a valid alternate viewpoint, it selects `inspect_target` directly.

Eligible memories are ranked using a retrieval score combining environmental and agent-state similarity (Fig. 2(c), left):

$$S_{\text{ret}}(s_t, m_i) = \lambda_{\text{env}} S_{\text{env}}(s_t, m_i) + \lambda_{\text{agent}} S_{\text{agent}}(s_t, m_i), \quad (7)$$

where S_{env} and S_{agent} are normalized to $[0, 1]$ and $\lambda_{\text{env}} + \lambda_{\text{agent}} = 1$ (we use $\lambda_{\text{env}} = 0.8$, $\lambda_{\text{agent}} = 0.2$ to emphasize local traversability). S_{env} compares local visibility, obstacle state, and edge-cost similarity; S_{agent} compares battery range

¹Our evaluated prototype uses the lightweight $k = 1$ instantiation.

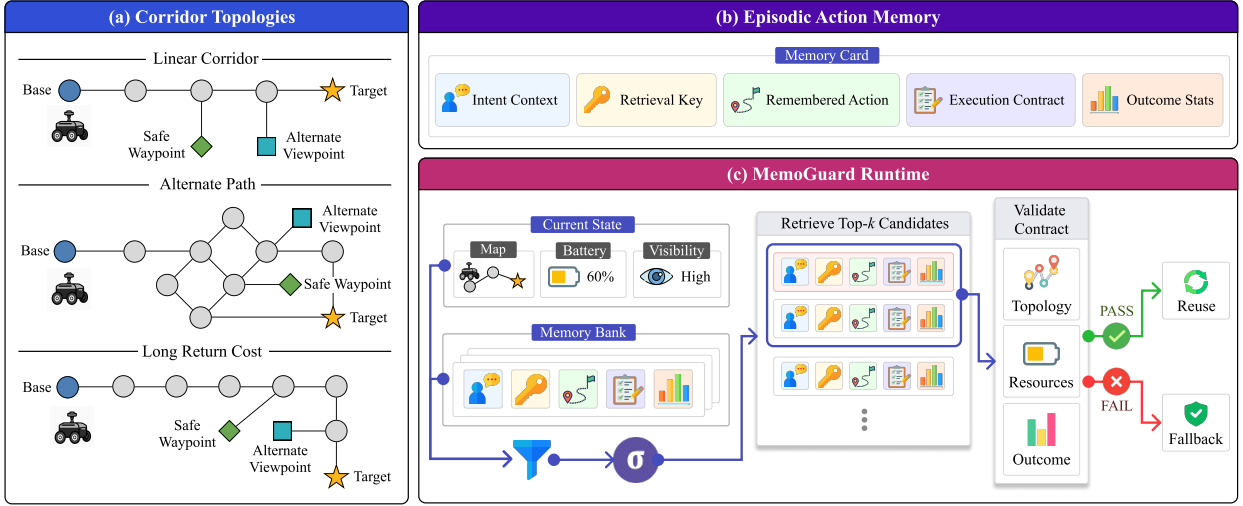


Fig. 2. MemoGuard framework. (a) Corridor topologies used in evaluation: Linear Corridor, Alternate Path, and Long Return Cost. (b) Episodic action memory format, including intent context, retrieval key, remembered action, execution contract, and outcome statistics. (c) Runtime flow: candidate memories are retrieved, validated using topology, resource, and outcome checks, and either reused or rejected in favor of a planner/reasoning fallback.

and localization-confidence level.

(B) Execution-Contract Validation: A retrieved memory is not reused solely because it is similar. As shown in Fig. 2(c), MemoGuard applies three hard feasibility gates as mentioned below (intent compatibility is enforced during eligibility filtering; validation covers topology, resources, and outcome reliability).

Topology check. It rejects memories whose structural assumptions no longer hold, including cases where the required edge is blocked or the remembered alternate viewpoint is unavailable.

Resource check. It rejects memories that would drive the robot below the configured battery safety floor. Let b_t be the current battery, $C_{\text{act}}(s_t, m_i)$ be the estimated cost of executing the remembered action, and B_{safe} be the safety floor. The resource condition is

$$b_t - C_{\text{act}}(s_t, m_i) \geq B_{\text{safe}}. \quad (8)$$

If the remembered action target is unavailable or unreachable, the action cost is treated as infeasible and the memory is rejected.

Outcome check. It rejects memories with unreliable prior outcomes. MemoGuard computes empirical reliability as

$$R(m_i) = \frac{n_i^s}{n_i^s + n_i^f}, \quad (9)$$

where n_i^s and n_i^f are accumulated success and failure counts for equivalent state-action memories, and reuses only if $R(m_i)$ exceeds the outcome threshold τ_{out} .

(C) Reuse or Fallback: A candidate memory is valid only if all checks pass:

$$\Phi(s_t, m_i) = \Phi_{\text{topo}}(s_t, m_i) \wedge \Phi_{\text{res}}(s_t, m_i) \wedge \Phi_{\text{out}}(m_i). \quad (10)$$

As shown in Fig. 2(c), if $\Phi(s_t, m_i) = 1$, MemoGuard reuses a_i ; otherwise it checks the next candidate. If no candidate passes, the runtime invokes fallback $f(s_t)$, which selects one action from $\mathcal{A}$ (Eq. 1). The reasoning fallback is instantiated as local llama3.2:3b inference constrained to a single supervisory action, following recent works on LLM-based high-level planning involving embodied agents [9]–[12].

IV. EXPERIMENTS AND EVALUATION

We evaluate whether MemoGuard reduces unsafe episodic memory reuse while avoiding the cost of always-reasoning, using a controlled graph-world stress test where retrieved memories may appear similar but become invalid due to topology, resource, or alternate-viewpoint changes.

(A) Setup: We implement a graph-based corridor inspection simulator for a ground robot operating over an initially available topological map. As shown in Fig. 2(a), the simulator uses three map families: *Linear Corridor*, *Alternate Path*, and *Long Return Cost*. Edges encode traversal cost and obstacle state; nodes encode local visibility.

Each scenario randomizes edge costs, obstacle states, visibility levels, target location, alternate-viewpoint availability, initial battery, and localization confidence, with a battery safety floor of $B_{\text{safe}} = 25$ and a maximum episode length of 50 steps.

Movement actions use a graph planner computing minimum-cost paths via Dijkstra’s algorithm. For `follow_planner`, the planner targets the inspection node before inspection and the safe waypoint or base after; for `return_to_safe_waypoint` and `inspect_alternate_viewpoint`, it targets the corresponding recovery or alternate-viewpoint node. The reasoning fallback returns one action from $\mathcal{A}$ (Eq. 1) and increments the fallback-call metric; graph planning only executes selected movement actions.

The memory bank is constructed offline from rollout data: successful transitions are stored as episodic action memories accumulating success counts, and failed rollouts contribute failure counts when they match an existing memory signature, yielding 11,558 memories. During evaluation, the retriever samples up to 2,000 memories using a fixed seed and ranks them using the retrieval score from Section III.

(B) Trap Generation, Baselines, and Metrics: To test memory traps, we generate trap scenarios from previously generated non-trap scenarios. Each trap applies one controlled modification while preserving the overall mission structure; only one trap type is applied per scenario.

TABLE I

AVERAGE PERFORMANCE ACROSS TRAP SCENARIOS. MISSION = MISSION SUCCESS; INSPECT = TARGET INSPECTED; DEplete = BATTERY DEPLETION; SAFETY VIOL. = BATTERY SAFETY VIOLATION; FALLBACK = AVERAGE FALLBACK CALLS PER TRIAL; COST = BATTERY-EQUIVALENT EXECUTION COST. ARROWS INDICATE PREFERRED DIRECTION.

Policy	Mission ↑	Inspect ↑	Deplete ↓	Safety Viol. ↓	Fallback ↓	Cost ↓
Top-1 Reuse	32.6	80.4	18.9	67.4	0.00	36.73
Threshold Reuse	32.7	76.6	18.9	67.1	0.53	37.09
Always Reasoning	83.5	98.6	2.2	16.5	18.58	63.03
MemoGuard	84.2	98.2	2.0	15.8	14.60	60.21

- **Blocked Edges:** one or more previously traversable edges are changed to blocked.
- **Reduced Battery:** the robot’s initial battery is reduced by 15%, subject to a minimum feasible battery level.
- **Removed Alternate Viewpoint:** the alternate viewpoint is removed and treated as a normal node with no alternate-inspection affordance.

Trap scenarios remain solvable after modification, so failures reflect invalid memory reuse rather than impossible missions.

We compare four policies. **Top-1 Reuse** directly executes the highest-scoring memory; **Threshold Reuse** adds a similarity threshold, invoking fallback if the score is too low. **Always Reasoning** invokes the local reasoning fallback at every decision step. **MemoGuard** retrieves a candidate memory and validates it before reuse.

We report six metrics. *Mission Success* is the percentage of trials in which the robot inspects the target, preserves the battery safety floor, and terminates at a recovery node. *Target Inspected* is the percentage of trials in which the target is inspected, regardless of final recovery status. *Battery Depletion* is the percentage of trials in which the battery reaches zero. *Battery Safety Violation* is the percentage of trials ending below the configured battery safety floor. *Fallback Calls* is the average number of planner/reasoning fallback invocations per trial. *Execution Cost* is the average battery-equivalent cost accumulated over movement, inspection, waiting, and fallback overhead.

(C) Memory-Trap Results: Table I summarizes average performance across trap scenarios. Similarity-only reuse is cheap but unsafe: Top-1 Reuse has zero fallback calls, but only 32.6% mission success and 67.4% battery safety violations. Threshold Reuse performs similarly, with 67.1% battery safety violations, showing that retrieval-score thresholding alone does not solve the memory-trap problem. Many traps are not low-confidence retrieval failures; they are high-similarity memories whose topology, resource, or alternate-viewpoint assumptions are invalid.

Compared with Top-1 Reuse, MemoGuard increases mission success from 32.6% to 84.2%. Overall, battery safety violations drop from 67.4% to 15.8%, corresponding to a 76.6% relative reduction. Compared with Always Reasoning, MemoGuard reduces fallback calls from 18.58 to 14.60 per trial, a 21.4% reduction, while maintaining comparable mission success: 84.2% vs. 83.5%. Thus, MemoGuard improves the safety-efficiency tradeoff: it is much safer than similarity-only reuse and less fallback-intensive than always reasoning.

Fig. 3 disaggregates the safety-efficiency tradeoff by trap

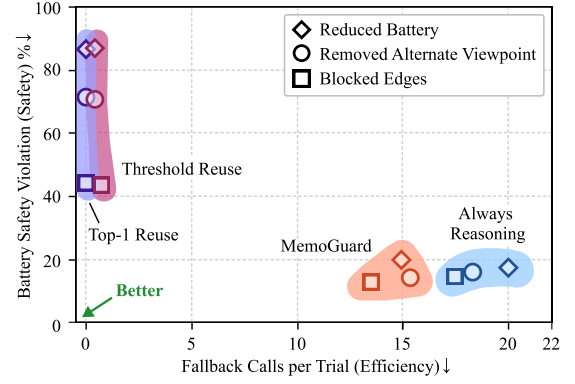


Fig. 3. Safety-efficiency tradeoff by trap type. Each point is one policy averaged over one trap mechanism; lower-left is better. Text labels identify trap mechanisms and policy operating regions. MemoGuard approaches Always Reasoning safety with fewer fallback calls, with the largest gains for blocked-edge and alternate-viewpoint traps.

mechanism. In Blocked Edges scenarios, MemoGuard reduces safety violations from 44.3% under Top-1 Reuse to 13.0%, slightly below Always Reasoning at 15.0%, while using fewer fallback calls: 13.51 vs. 17.47. In Removed Alternate Viewpoint scenarios, MemoGuard reduces violations from 71.2% to 14.4%, compared with 16.5% for Always Reasoning, while reducing fallback calls from 18.28 to 15.37. The Reduced Battery case is harder: MemoGuard reduces violations from 86.8% to 20.1%, but remains above Always Reasoning at 17.9%; however, it still uses fewer fallback calls, 14.94 vs. 19.99. These trends show that MemoGuard is strongest when memory traps arise from discrete topology or affordance changes, while resource-margin traps remain more sensitive to cost estimation.

On Jetson AGX Xavier in MODE_30W_ALL, local llama3.2:3b fallback reasoning costs 0.922 s and 9.288 J per call; therefore, MemoGuard’s 3.98-call reduction over Always Reasoning in Table I corresponds to 3.67 s and 36.97 J of avoided fallback-reasoning overhead per trial.

V. CONCLUSION AND FUTURE WORK

This paper presents MemoGuard, a lightweight runtime for contract-validated episodic memory reuse in communication-limited robot navigation. MemoGuard addresses memory traps by validating a retrieved memory’s topology, resource, and outcome assumptions before reuse. This validation shifts the safety-efficiency tradeoff away from unsafe similarity-only reuse and reduces reliance on costly always-reasoning. In corridor-inspection scenarios, MemoGuard reduced battery safety violations by 76.6% over similarity-only top-1 reuse and fallback calls by 21.4% over always reasoning. Future work will extend MemoGuard from graph-level contracts to cross-layer execution contracts that incorporate localization uncertainty, communication quality, and energy/resource trajectories for communication-constrained mobile robots [13]–[15], extending toward multi-robot collaboration [16]–[19].

REFERENCES

- [1] R. Bhattacharjya, S.-Y. Wu, H. Oh, C. Nam, S. Koo, M. Imani, E. Bozorgzadeh, and N. Dutt, “Avery: Adaptive vlm split computing through embodied self-awareness for efficient disaster response systems,” *arXiv preprint arXiv:2511.18151*, 2025.
- [2] F. Cladera, Z. Ravichandran, J. Hughes, V. Murali, C. Nieto-Granda, M. A. Hsieh, G. J. Pappas, C. J. Taylor, and V. Kumar, “Air-ground collaboration for language-specified missions in unknown environments,” *IEEE Transactions on Field Robotics*, 2025.

- [3] M. Nakanoya, S. S. Narasimhan, S. Bhat, A. Anemogiannis, A. Datta, S. Katti, S. Chinchali, and M. Pavone, “Co-design of communication and machine inference for cloud robotics,” *Autonomous Robots*, vol. 47, no. 5, pp. 579–594, 2023.
- [4] K. Cesare, R. Skeele, S.-H. Yoo, Y. Zhang, and G. Hollinger, “Multi-uav exploration with limited communication and battery,” in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 2230–2235.
- [5] A. Aamodt and E. Plaza, “Case-based reasoning: Foundational issues, methodological variations, and system approaches,” *AI Commun.*, 1994.
- [6] J. Rothfuss, F. Ferreira, E. E. Aksoy, Y. Zhou, and T. Asfour, “Deep episodic memory: Encoding, recalling, and predicting episodic experiences for robot action execution,” *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4007–4014, 2018.
- [7] Z. Zheng, X. Zhou, Z. Mao, S. Sun, L. Zhang, Y. Ao, Y. Feng, Q. Zhang, Y. Lin, and X. Chen, “Freqcache: Accelerating embodied vln models with adaptive frequency-guided token caching,” *arXiv preprint arXiv:2604.24391*, 2026.
- [8] S. Wang, Y. Luo, X. Chen, A. Luo, D. Li, C. Liu, S. Chen, Y. Zhang, and J. Yu, “Vlingnav: Embodied navigation with adaptive reasoning and visual-assisted linguistic memory,” *arXiv preprint arXiv:2601.08665*, 2026.
- [9] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian *et al.*, “Do as i can, not as i say: Grounding language in robotic affordances,” in *Conference on robot learning*. PMLR, 2023, pp. 287–318.
- [10] Z. Wan, Y. Du, M. Ibrahim, J. Qian, J. Jabbour, Y. Zhao, T. Krishna, A. Raychowdhury, and V. J. Reddi, “Reca: Integrated acceleration for real-time and efficient cooperative embodied autonomous agents,” in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2025, pp. 982–997.
- [11] H. Su, A. Walsman, D. Garces, S. Kakade, and S. Gil, “Data-efficient multi-agent spatial planning with llms,” *arXiv preprint arXiv:2502.18822*, 2025.
- [12] A. Tahmasbi, S. Majidi, K. Taram, and A. Bera, “From building blocks to planning: Multi-step spatial reasoning in llms with reinforcement learning,” *arXiv preprint arXiv:2512.24532*, 2025.
- [13] R. Bhattacharjya, A. Sarkar, I. Kool, S. Baidya, and N. Dutt, “Access-av: Adaptive communication-computation codesign for sustainable autonomous vehicle localization in smart factories,” *ACM Trans. Embed. Comput. Syst.*, vol. 25, no. 1, Jan. 2026. [Online]. Available: <https://doi.org/10.1145/3771770>
- [14] C. Wu, Z. Zhao, J. Wang, R. Xu, C. Zhu, Z. Yang, and Z. Zhang, “Joint communication and computation design for mobile embodied ai network (mean),” *arXiv preprint arXiv:2605.14300*, 2026.
- [15] A. Vashisth, M. Kulshrestha, D. Conover, and A. Bera, “Scalable multi-robot informative path planning for target mapping via deep reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 11, no. 3, pp. 3414–3421, 2026.
- [16] R. Karam, R. Lin, B. A. Butler, and M. Egerstedt, “Learning altruistic collaboration in heterogeneous multi-team systems,” *arXiv preprint arXiv:2605.21723*, 2026.
- [17] R. Karam, A. A. Nguyen, R. Lin, D. R. Martin, D. Morales, B. A. Butler, and M. Egerstedt, “Collaboration in multi-robot systems: Taxonomy and survey over frameworks for collaboration,” *arXiv preprint arXiv:2603.23898*, 2026.
- [18] X. Zhang, J. Chen, Y. Zhu, B. Luo, and M. Guo, “Cocoplan: Adaptive coordination and communication for multi-robot systems in dynamic and unknown environments,” *IEEE Robotics and Automation Letters*, vol. 11, no. 3, pp. 3270–3277, 2026.
- [19] Z. Ravichandran, F. Cladera, A. Prabhu, J. Hughes, V. Murali, C. Taylor, G. J. Pappas, and V. Kumar, “Heterogeneous robot collaboration in unstructured environments with grounded generative intelligence,” *arXiv preprint arXiv:2510.26915*, 2025.