BEYOND CONTEXT: EXECUTABLE MEMORY FOR EMBODIED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon robotic tasks require memory because the same current observation can call for different actions depending on earlier interactions. Existing memory-augmented systems improve how history is stored and retrieved, but often pass it to a learned model whose interpretation determines the next decision—an interface we call *Context-Memory*. Yet recalling the relevant past does not by itself determine its role in execution: identifying a previously used target and deciding whether to repeat an interaction require different operations on history. Our insight is that these operations can be made explicit, allowing remembered information to update task state rather than merely provide context for inferring it. We introduce *Execute-Memory*, which couples persistent entity and event memory with compositional task programs that resolve historical references, maintain progress, and determine the next execution requirement. By keeping uncertain world beliefs separate from accepted events, it links progress updates to observed interactions without assuming perfect perception. The resulting requirement is grounded in the current scene and passed as a local subgoal to a frozen vision-language-action policy. Experiments on RoboMME and challenging stress subsets derived from it show substantial gains over memory-augmented and policy baselines, supporting a shift from remembered context to executable memory.

1 INTRODUCTION

Recent vision-language-action (VLA) models have made rapid progress in robotic manipulation (Brohan et al., 2023b; Driess et al., 2023; Brohan et al., 2023a; O'Neill et al., 2024; Ghosh et al., 2024; Physical Intelligence et al., 2025), but much of their strength is realized when the current observation already contains the information needed for the next action. Long-horizon embodied tasks break this assumption. The same visible scene can require different actions depending on what happened earlier: a robot may need to remember which object was previously manipulated, how many times an interaction has been completed, or which step of a procedure should come next. In these settings, acting correctly requires more than perception of the present—it requires access to task-relevant history. Memory is therefore a natural next step for extending reactive VLA policies toward long-horizon execution.

Existing embodied memory methods mainly differ along three axes: how history is stored, how relevant information is retrieved, and how the retrieved memory is injected into a planner or policy. Prior work has explored raw observations, compressed features, recurrent states, structured scene representations, key events, and experience traces as memory substrates, together with attention, similarity-based retrieval, keyframe selection, and learned memory modules for accessing them (Fang et al., 2019; Shi et al., 2025; Koo et al., 2025; Sridhar et al., 2025; Yang et al., 2026; Zeng et al., 2026). These designs provide effective ways to preserve and expose task-relevant history. However, despite their differences in representation and retrieval, many systems ultimately share a similar execution interface: retrieved memory is provided as additional context to a learned model, which must interpret what that history implies for the current decision. We refer to this paradigm as *Context-Memory*. The key limitation is not that memory must be unstructured or insufficiently informative, but that its consequences for execution remain implicit.

This implicit interface becomes increasingly limiting as memory requirements become more semantically diverse. RoboMME categorizes embodied memory demands into temporal, spatial, ob-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

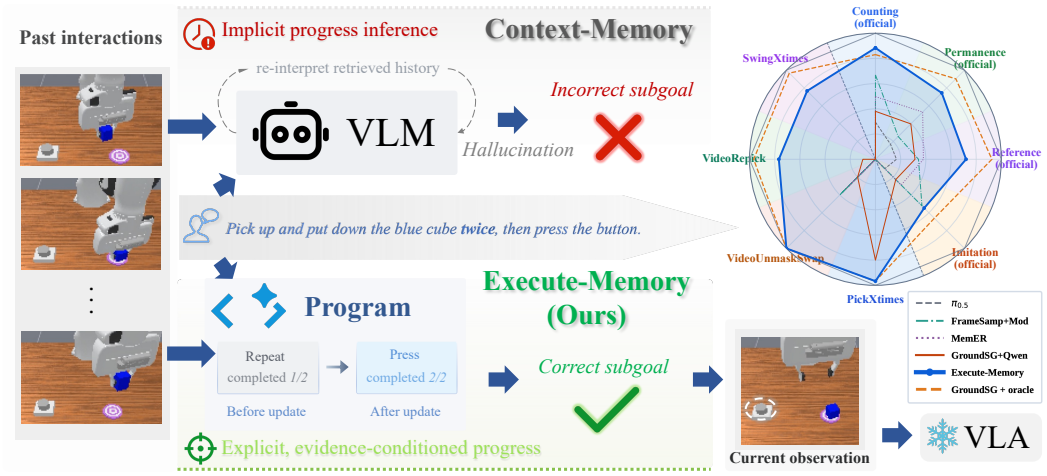

Figure 1: **Execute-Memory: from remembered context to executable state.** Context-Memory leaves history interpretation to the policy, whereas Execute-Memory converts interaction evidence into explicit state and a resolved local subgoal before invoking the same frozen VLA. The capability profile summarizes the method's performance across diverse tasks.

ject/reference, and procedural types (Dai et al., 2026), each of which affects execution differently: temporal memory may determine how many interactions remain or which event occurred first; spatial memory may preserve relations that are no longer directly visible; object/reference memory may identify which previously encountered entity is being referred to; and procedural memory may constrain what step should happen next. RMBench complements this view by characterizing how much task-relevant history may be required for an optimal decision (Chen et al., 2026). These benchmarks suggest that the next step for embodied memory is not simply better recall, but a different execution interface. We therefore ask: *can task-relevant history be transformed into explicit execution state and constraints, rather than being left as context for a learned model to interpret?*

Realizing executable memory introduces two challenges. First, continuous, partially observed robot experience must be converted online into persistent entities and meaningful events. Occlusion, identity fragmentation, transient contact, and repeated observations make it difficult to determine whether a new observation corresponds to the same entity or a genuinely new occurrence (Gu et al., 2023; Peng et al., 2024). Second, these structured memories must be turned into execution state. The system must determine which past entities and events matter for the current task, how they update progress or bind a historical referent, when execution should advance, and how the resulting requirement should be re-grounded in the current scene. This process must remain reliable even when entity correspondence or event interpretation is uncertain, rather than requiring every ambiguity to be collapsed into a single hard interpretation.

To address these challenges, we propose *Execute-Memory*, a framework that turns robot experience into structured, executable memory for long-horizon control. As illustrated in Figure 1, Context-Memory leaves the execution consequences of retrieved history to a learned model, whereas Execute-Memory makes them explicit through a compositional program over persistent entities and events. The program maintains historical bindings and task progress, then re-grounds the resulting execution requirement in the current observation before passing a local subgoal to a frozen VLA policy. In this way, memory no longer merely informs the policy; it explicitly participates in deciding *what should be executed next*.

Our contributions are threefold:

- **A memory-to-execution perspective.** We distinguish *Context-Memory*, where retrieved history is interpreted implicitly by a learned model, from *Execute-Memory*, where remembered information is explicitly transformed into target bindings, task progress, and procedural constraints.

- **The Execute-Memory framework.** We introduce a structured memory framework that maintains persistent entities and events, composes them through memory operators, tracks execution state explicitly, and re-grounds the resulting local requirement before invoking a frozen VLA policy.
- **Evaluation on history-dependent manipulation.** On RoboMME and challenging stress subsets derived from it, Execute-Memory achieves substantial gains over existing memory and policy baselines and remains robust as task-relevant history becomes longer and more complex.

2 Related Work

Representing and retrieving embodied history. Scene Memory Transformer and SAM2Act+ use attention to access stored observation embeddings and spatial features, respectively (Fang et al., 2019; 2025). EventVLA predicts future keyframe probabilities to retain sparse visual evidence (Yang et al., 2026), whereas KEMO combines kinematic cues with visual filtering to select event keyframes (Zeng et al., 2026). Complementary structured approaches include ConceptGraphs, which builds open-vocabulary 3D scene graphs for language-guided planning (Gu et al., 2023), and Object Permanence Filter, which maintains object tracks under occlusion (Peng et al., 2024). These approaches address how embodied history is represented and made accessible for subsequent decisions.

Memory-conditioned policies. MemoryVLA and HAMLET condition action prediction on retrieved or aggregated memory features (Shi et al., 2025; Koo et al., 2025). MemER learns to select historical keyframes and combines them with recent observations to generate textual instructions for a low-level policy (Sridhar et al., 2025). RoboMME compares context-, modulation-, and expert-based memory integration (Dai et al., 2026). We use *Context-Memory* to describe interfaces in which historical information conditions a learned decision module that interprets its implications for the next action or subgoal. This definition includes feature-level conditioning and is broader than RoboMME's memory-as-context mechanism. The distinction concerns where memory is translated into execution requirements, not whether it is structured or accompanied by feedback.

Structured planning and guarded execution. Code as Policies, ProgPrompt, SayCan, SayPlan, and SkiROS2 externalize control through policy code, situated program generation, affordance grounding, scene-graph planning, and guarded skill execution, respectively (Liang et al., 2022; Singh et al., 2023; Ahn et al., 2022; Rana et al., 2023; Mayr et al., 2023). Inner Monologue further couples language-model planning with execution feedback (Huang et al., 2022). Recent systems connect memory with learned local control. RMBench's Mem-0 uses completed-subtask memory and a subtask-end classifier (Chen et al., 2026). Cortex aligns a VLM planner and a VLA controller through a standardized skill interface and event-balanced training for completion, memory updates, and subtask transitions (Peng et al., 2026). Harness VLA combines a frozen VLA with analytic primitives, using task-specific execution traces and reusable success/failure knowledge to guide grounding, composition, and retries (Zhang et al., 2026). Execute-Memory focuses on how current-episode memory is queried and evaluated to govern execution. Within supported task families, compositional programs use persistent entities and events to resolve historical references, maintain repetition counts, and determine phase transitions. Event admission and phase-specific consumption regulate these updates under imperfect perception, while current-scene grounding produces a local subgoal for a frozen VLA. The contribution is therefore not programmatic control alone, but an explicit connection between embodied memory demands and execution state.

3 Method

Context-Memory leaves the effect of history on the next decision to a learned model. Execute-Memory externalizes this mapping: structured entities and events are queried by a task-defined program, accepted results update explicit execution state, and the current requirement is re-grounded before the frozen VLA acts. Figure 2 presents the four corresponding components.

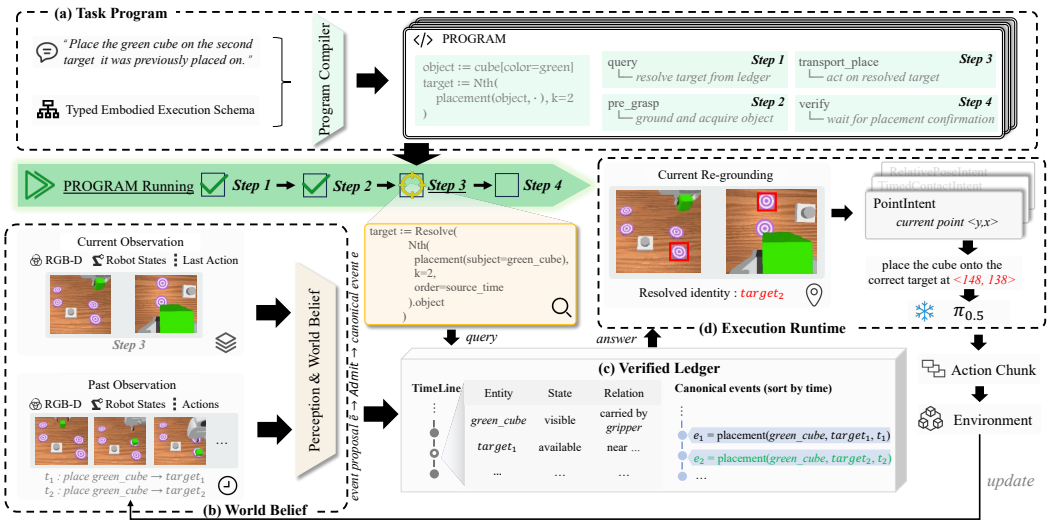

Figure 2: **From structured memory to local execution.** Task Program specifies memory operations; World Belief maintains entities; Verified Ledger stores accepted events; and Execution Runtime updates state and emits a grounded local subgoal through GroundSG (Dai et al., 2026) to frozen $\pi_{0.5}$ (Physical Intelligence et al., 2025). The example resolves the second historical placement and grounds its target in the current scene.

3.1 Task Program

Different memory demands require different operations: counting and ordering, historical reference resolution, set iteration, or procedural sequencing. The program makes these operations explicit rather than asking a learned model to infer them from retrieved context. Given instruction u and registered event vocabulary E, the compiler constructs a query specification and task program, $(Q, P) = \mathsf{Compile}(u, \mathsf{E})$. Q describes the required predicate, roles, temporal scope, and answer type; P describes operator composition, phase order, and completion requirements. The program specifies *how* to obtain an answer; the episode history supplies the answer itself.

At execution state s_{t-1}, let $\mathcal{L}_t$ denote the current ledger, including versioned entities, relations, and canonical events. The active query and its value/support pair are

$$q_t = \mathsf{PhaseQuery}(P, Q, s_{t-1}), \qquad (v_t, W_t) = \mathsf{Eval}(q_t, \mathcal{L}_t), \qquad v_t = [\![q_t]\!]_{\mathcal{L}_t}. \tag{1}$$

Here v_t is a typed answer and W_t its supporting ledger records; the bracket notation denotes only the value component. Unresolved identity, ordering, or evidence returns an unresolved result rather than a forced answer. Resolve, Count, Nth, Before/After, Repeat, and Set provide the main reusable operations; their full typing and semantics are given in Appendix B.

For the historical-target example, the program selects an event and then its role binding:

$$e^\star = \mathsf{Nth}(\mathsf{placement}(\mathtt{cube}, \cdot), 2), \qquad \theta_{\text{target}} = \mathsf{Resolve}(e^\star.\mathrm{object}). \tag{2}$$

The ordinal indexes an event, not a permanently named target; changing the episode history can therefore change the binding without changing the program. The current compiler uses registered templates and family compositions rather than unrestricted program synthesis. Task-level compositions are summarized in Table 9, and representative event contracts are listed in Table 6.

3.2 Perception and World Belief

World Belief converts continuous legal observations into persistent entity hypotheses. Inputs comprise RGB/RGB–D, robot state, and action feedback with source frame and timestamp. The belief retains current geometry, semantic state, relations, visibility, uncertainty, and provenance. A missing object in one view does not erase its historical hypothesis, and ambiguous correspondence can remain unresolved rather than being collapsed prematurely.

The current provider uses a compact YOLO11n-seg detector for object regions from Ultralytics (Ultralytics, 2024), calibrated RGB–D projection for 3D measurements, and temporal association over appearance, geometry, motion, visibility, and scene epoch. These are replaceable measurement providers, not the memory abstraction; their legal-input boundary and interface are detailed in Appendices D and E. Belief changes and observed interactions produce typed event proposals. A proposal records predicate, roles, source-time interval, and supporting observations, but has no phase authority and cannot by itself increment a counter.

3.3 Verified Ledger

World Belief describes what is plausible; the Verified Ledger records which occurrences may support program reasoning. Admission applies registered identity, temporal, geometric, and consistency checks to a proposal. Accepted occurrences are inserted idempotently:

$$\mathcal{E}_t^{\mathrm{can}} = \mathcal{E}_{t-1}^{\mathrm{can}} \oplus_{\mathrm{key}} \{\mathsf{Canon}(\widetilde{e}) : \mathsf{Accept}(\widetilde{e}) = 1\}. \tag{3}$$

The key prevents replay of the same occurrence from creating another event. Verified means accepted under the observation-based contract, not physical truth. Admission/deduplication is distinct from phase-scoped consumption: admission decides whether an occurrence is stored, while consumption decides whether an already stored occurrence may satisfy the current obligation. Evidence envelopes, EventSpec fields, provenance, and admission checks are specified in Appendices C–F.

3.4 Execution Runtime

The runtime turns query results into explicit task progress. With s_t denoting phase, role bindings, loop state, and consumed-event records, it applies

$$s_t = \begin{cases} \mathsf{Update}_P(s_{t-1}, v_t, W_t), & \mathsf{Ready}_P(s_{t-1}, v_t, W_t) = 1, \\ s_{t-1}, & \text{otherwise.} \end{cases} \tag{4}$$

For repeat obligation r, let $W_t^{(r)}$ be the current iteration's completion witness and $C_t^{(r)}$ its consumed event identifiers. The compact counted-loop update is

$$\begin{aligned} \delta_t^{(r)} &= \mathbf{1}[\mathsf{Complete}_P(W_t^{(r)}, s_{t-1}) \wedge \mathrm{ids}(W_t^{(r)}) \cap C_{t-1}^{(r)} = \varnothing], \\ n_t^{(r)} &= n_{t-1}^{(r)} + \delta_t^{(r)}, \qquad C_t^{(r)} = C_{t-1}^{(r)} \cup \begin{cases} \mathrm{ids}(W_t^{(r)}), & \delta_t^{(r)} = 1, \\ \varnothing, & \delta_t^{(r)} = 0. \end{cases} \end{aligned} \tag{5}$$

Thus a command request is not physical completion, and replaying an event cannot create a new repetition. The witness is selected under the active roles, source-time scope, causal order, and consumption scope; it is not the full set of historical query support records.

Finally, history determines what requirement is active while the current scene determines where it can be realized. Here θ_t denotes the resolved role binding, χ_t the active intent and phase constraints, and $\bar{o}_t$ the current legal sensory input:

$$g_t = \mathsf{Ground}(\theta_t, \mathcal{B}_t, \bar{o}_t), \quad \iota_t = \mathsf{Intent}(\theta_t, g_t, \chi_t), \quad g_t^{\pi} = \mathsf{Adapter}(\iota_t), \quad \mathbf{a}_t \sim \pi_{\mathrm{frozen}}(\bar{o}_t, g_t^{\pi}). \tag{6}$$

The principal path uses PointIntent through GroundSG. If the required binding, geometry, or supported intent remains unresolved, the runtime defers subgoal emission or abstains. Action-chunk lifecycle and intent details appear in Appendices G and I.

Reusable history and consumable evidence. Historical events may be queried repeatedly to resolve an entity, an ordinal relation, or a predecessor condition, but an interaction occurrence should not become a new completion merely because it is read again. Execute-Memory therefore separates admission, phase-scoped consumption, and current grounding: admission determines whether an observed occurrence enters the canonical ledger; consumption determines whether an accepted occurrence is eligible to satisfy the active obligation and has not already been consumed for that obligation; and grounding determines where the resulting requirement can be executed. Thus, historical evidence remains reusable for read-only reference queries without allowing repeated reads of the same occurrence to spuriously advance execution progress.

Table 1: **Results on the official RoboMME suite.** Entries are mean success rates (%) from 30 episodes per task and seed across three evaluation seeds (90 episodes per task in total); higher is better. Avg. is the unweighted mean over all 16 tasks. † marks privileged oracle references, which are capacity references rather than Execute-Memory inputs. All settings follow the evaluation protocol in Section 4.1.

Method	Counting				Permanence				Reference				Imitation				
	Bin Fill	Pick Xtimes	Swing Xtimes	Stop Cube	Video Unmask	Button Unmask	Video Unmask Swap	Button Unmask Swap	Pick Highlight	Video Repick	Video Place Button	Video Place Order	Move Cube	Insert Peg	Pattern Lock	Route Stick	Avg.
Policy and subgoal controls																	
$\pi_{0.5}$ (baseline)	36.7	37.8	40.0	3.3	17.8	22.2	14.4	13.3	12.2	2.2	26.7	25.6	28.9	2.2	6.7	4.4	18.4
SimpleSG + Qwen	**82.2**	**97.8**	4.4	3.3	34.4	15.6	12.2	4.4	21.1	31.1	27.8	22.2	83.3	**8.9**	6.7	12.2	29.2
SimpleSG + oracle†	83.3	100.0	100.0	43.3	35.6	18.9	16.7	15.6	42.2	31.1	30.0	27.8	84.4	11.1	95.6	57.8	49.6
GroundSG + Qwen	52.2	88.9	10.0	1.1	86.7	27.8	30.0	14.4	17.8	22.2	55.6	30.0	74.4	3.3	5.6	7.8	33.0
GroundSG + oracle†	84.4	100.0	98.9	48.9	96.7	92.2	97.8	74.4	83.3	93.3	98.9	94.4	85.6	18.9	97.8	62.2	83.0
Memory methods																	
FrameSamp + Context	36.7	70.0	76.7	15.6	28.9	23.3	26.7	17.8	21.1	15.6	30.0	16.7	82.2	0.0	18.9	22.2	31.4
FrameSamp + Modulator	43.3	92.2	93.3	38.9	33.3	25.6	20.0	23.3	24.4	27.8	57.8	26.7	83.3	3.3	56.7	**65.6**	44.7
TTT + Context	31.1	63.3	43.3	0.0	28.9	22.2	21.1	10.0	26.7	11.1	31.1	18.9	25.6	1.1	2.2	4.4	21.3
MemER	58.9	77.8	62.2	0.0	81.1	73.3	34.4	23.3	**68.9**	28.9	28.9	25.6	**84.4**	3.3	17.8	10.0	42.4
MemoryVLA	10.0	14.4	2.2	0.0	21.1	1.1	3.3	2.2	7.8	1.1	8.9	6.7	13.3	1.1	6.7	3.3	6.5
Execute-Memory (ours)	71.1	96.7	**95.6**	**90.0**	**87.8**	**76.7**	**88.9**	**44.4**	46.7	**75.6**	**72.2**	**93.3**	73.3	7.8	**77.8**	60.0	**72.4**

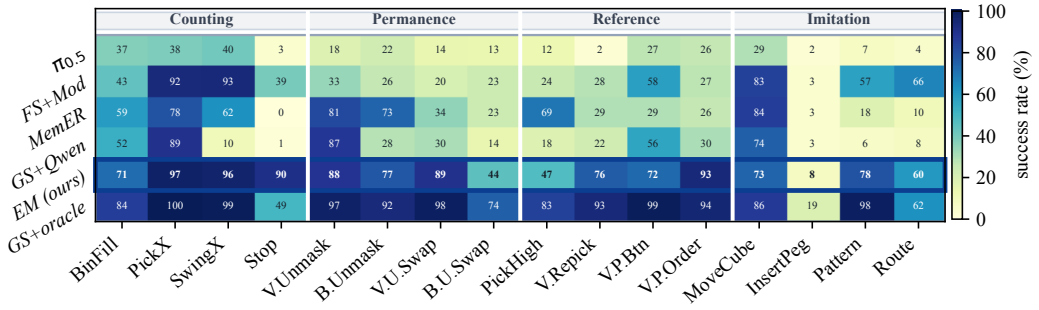

Figure 3: **Task-wise success profile on the official RoboMME suite.** The heatmap groups all 16 tasks by the four RoboMME memory-demand families; the Execute-Memory row is outlined for direct comparison. Here FS+Mod denotes FrameSamp+Modulator, GS denotes GroundSG, and EM denotes Execute-Memory.

4 Experiments

We evaluate whether explicit memory-to-execution semantics improve history-dependent manipulation. The experiments examine performance on RoboMME, robustness to longer histories, and the contribution of the individual execution-memory mechanisms.

4.1 Experimental Setup

Benchmark and protocol. We use the official 16-task RoboMME suite, grouped into Counting, Permanence, Reference, and Imitation (Dai et al., 2026). Custom stress subsets are reported separately. Each official task is evaluated for 30 episodes under each of three random seeds, for 90 episodes per task in total. Within each comparison, methods share the initial task instances, seeds, camera configuration, source videos, simulator version, and action-budget rule. Execute-Memory uses RGB or RGB–D observations, robot state, and available action feedback. Simulator-only identities and poses, truth segmentation, oracle subgoals, and evaluator-side progress or success signals are excluded from its inputs; success is used only for evaluation.

Compared methods. The policy baseline is $\pi_{0.5}$ without an additional history interface (Physical Intelligence et al., 2025). SimpleSG + Qwen and GroundSG + Qwen are explicit-subgoal controls, while SimpleSG + oracle and GroundSG + oracle provide privileged capacity references

Table 2: **Extended stress tests derived from RoboMME.** Entries report success rates (%) over a fixed batch of 30 episodes per task and stress level. All methods use the same episode set within each setting. N counts cycles or target arrivals, and M counts hidden-cover swaps. † marks the privileged oracle reference. Custom results are excluded from the official RoboMME average.

Method	PickXtimes				VideoUnmaskSwap				VideoRepick				SwingXtimes			
	N=2	N=5	N=8	N=10	M=1	M=3	M=5	M=7	N=2	N=4	N=6	N=8	N=3	N=4	N=5	N=6
$\pi_{0.5}$ (baseline)	16.7	0.0	0.0	0.0	20.0	36.7	30.0	36.7	0.0	0.0	0.0	0.0	0.0	0.0	6.7	0.0
GroundSG + oracle†	100.0	100.0	100.0	96.7	100.0	100.0	100.0	100.0	96.7	96.7	96.7	96.7	100.0	100.0	96.7	96.7
GroundSG + Qwen	**100.0**	90.0	**100.0**	80.0	20.0	20.0	10.0	20.0	20.0	20.0	30.0	10.0	0.0	0.0	0.0	0.0
FrameSamp + Context	70.0	0.0	0.0	0.0	30.0	13.3	33.3	33.3	16.7	6.7	0.0	0.0	86.7	0.0	0.0	0.0
FrameSamp + Modulator	90.0	66.7	0.0	0.0	20.0	13.3	20.0	40.0	30.0	0.0	0.0	0.0	**93.3**	0.0	0.0	0.0
TTT + Context	30.0	10.0	6.7	0.0	33.3	33.3	26.7	46.7	6.7	6.7	0.0	0.0	6.7	10.0	0.0	0.0
MemER	80.0	70.0	0.0	0.0	20.0	30.0	10.0	30.0	10.0	10.0	0.0	0.0	90.0	0.0	0.0	0.0
Execute-Memory (ours)	**100.0**	**100.0**	**100.0**	**96.7**	**96.7**	**96.7**	**100.0**	**100.0**	**80.0**	**80.0**	**76.7**	**76.7**	80.0	**86.7**	**86.7**	**76.7**

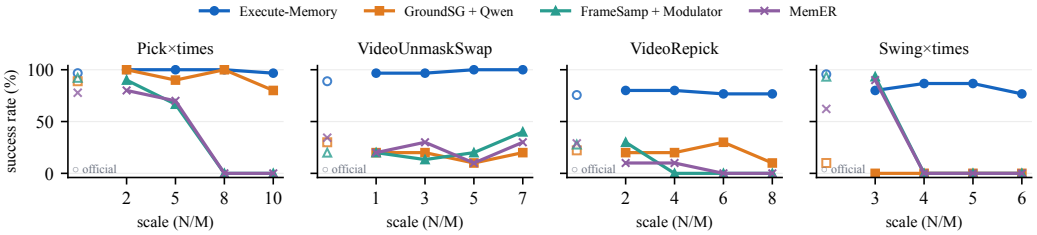

Figure 4: **Success rate as N or M increases.** Four task-specific panels compare Execute-Memory, GroundSG+Qwen, FrameSamp+Modulator, and MemER. Marker shapes identify methods. Filled markers are custom stress levels; open markers are the corresponding official-task reference and are not fitted as an additional stress level.

for the corresponding RoboMME subgoal interfaces (Dai et al., 2026). Memory baselines include FrameSamp + Context, FrameSamp + Modulator, and TTT + Context from the RoboMME memory suite (Dai et al., 2026), together with MemER (Sridhar et al., 2025) and MemoryVLA (Shi et al., 2025). Execute-Memory preserves the GroundSG interface and frozen $\pi_{0.5}$ checkpoint (Dai et al., 2026; Physical Intelligence et al., 2025). The Qwen subgoal controls use the Qwen3-VL-4B-Instruct model (Bai et al., 2025). Execute-Memory's comparisons with GroundSG + Qwen and GroundSG + oracle hold this local policy interface and checkpoint fixed. Both oracle conditions use privileged subgoals and are reported separately as policy-capacity references, not as observation-only competitors or inputs to Execute-Memory.

Metrics and reporting. All result tables report success rates (%). For the official RoboMME results, we compute the success rate for each seed and average across the three seeds. Overall scores are unweighted means across the 16 tasks; category scores average the four tasks in each group. Differences are in percentage points. The recorded implementation components and official evaluation fields are summarized in Table 4; the stress-subset settings are summarized in Table 5.

4.2 Results on RoboMME

Table 1 reports per-task performance and the macro-average; Figure 3 visualizes the corresponding task-wise profile, and Appendix Figure 7 shows the paired gain and oracle-reference gap. Execute-Memory achieves 72.4% overall success, compared with 18.4% for the policy baseline and 33.0% for GroundSG + Qwen. The gains are 54.0 and 39.4 percentage points, respectively; the difference from the strongest evaluated memory baseline is 27.6 percentage points. The GroundSG + Qwen comparison tests alternative memory-to-subgoal designs under the same local policy interface. The SimpleSG + oracle and GroundSG + oracle rows report 49.6% and 83.0%, respectively, as privileged capacity references for their corresponding interfaces; neither oracle condition is an input to Execute-Memory.

Table 3: **Minimal mechanism ablations.** Entries are success rates (%) from 10 episodes per task. Overall and category columns are unweighted means over the corresponding task sets.

Variant	Overall	Counting	Permanence	Reference	Imitation
Execute-Memory (full)	**72.5**	**87.5**	**75.0**	**72.5**	**55.0**
Structured Memory + Qwen	31.3	50.0	40.0	25.0	10.0
w/o event admission	6.3	0.0	10.0	5.0	10.0
w/o phase-scoped consumption	6.3	0.0	5.0	7.5	12.5

4.3 Scaling on Longer Histories

Scaling to longer histories. We construct four stress families from RoboMME. PickXtimes varies the number of pick–put-down cycles, $N \in \{2, 5, 8, 10\}$. VideoUnmaskSwap varies the number of hidden-cover swaps, $M \in \{1, 3, 5, 7\}$. VideoRepick combines a historical object reference with repeated pick–put-down cycles, $N \in \{2, 4, 6, 8\}$. SwingXtimes varies alternating right/left target arrivals, $N \in \{3, 4, 5, 6\}$. Task mechanics and evaluation rules are retained as the required count changes, and all methods share the episode set within each setting. Each level uses 30 episodes in the fixed stress batch; the reported values are not averaged over a separate seed set.

All custom stress results reuse the checkpoints from the official-task evaluation and use no stress-subset or level-specific fine-tuning. GroundSG+ oracle uses privileged evaluation-time subgoals and is reported only as a capacity reference.

Table 2 reports all eight methods, while Figure 4 visualizes Execute-Memory, GroundSG+Qwen, FrameSamp+Modulator, and MemER as N or M increases. At the largest settings, Execute-Memory achieves 96.7%, 100.0%, 76.7%, and 76.7% on PickXtimes, VideoUnmaskSwap, VideoRepick, and SwingXtimes, respectively. Increasing N in the counted-interaction settings lengthens the required execution sequence. Increasing M in VideoUnmaskSwap lengthens the hidden-cover history and increases swap/occlusion complexity; the oracle reference helps separate memory-related degradation from local execution difficulty.

These stress settings increase the amount of task-relevant history while preserving the underlying task family and execution interface. They test whether the same registered memory-to-execution semantics remain usable as more events must be distinguished, counted, or referenced, rather than whether the system can synthesize arbitrary unseen programs.

4.4 Ablation Study

We use the official RoboMME tasks to isolate two execution contracts and one memory-to-action interface. The ablations keep the observation interface, checkpoint, action budget, and task instances fixed within each comparison. Each variant is evaluated for 10 episodes per task. Overall and category scores are unweighted means over the corresponding task sets.

Variants. **Structured Memory + Qwen** receives the same episode-level structured entity/event memory used by Execute-Memory, but delegates production of the next local subgoal to Qwen rather than evaluating a typed execution program. Event admission and phase-scoped consumption remain enabled, while the downstream GroundSG/current-grounding interface and frozen $\pi_{0.5}$ execution path are kept fixed. **w/o event admission** bypasses event admission while preserving the downstream program and phase runtime. **w/o phase-scoped consumption** disables the obligation-scoped reuse restriction while retaining the other registered evidence checks.

Table 3 summarizes these comparisons.

Replacing the program-based subgoal decision with Qwen reduces overall success from 72.5% to 31.3%. Bypassing event admission or removing the obligation-scoped reuse restriction further reduces success to 6.3%. Under the evaluated configuration, these results indicate that both contracts are important for reliable execution.

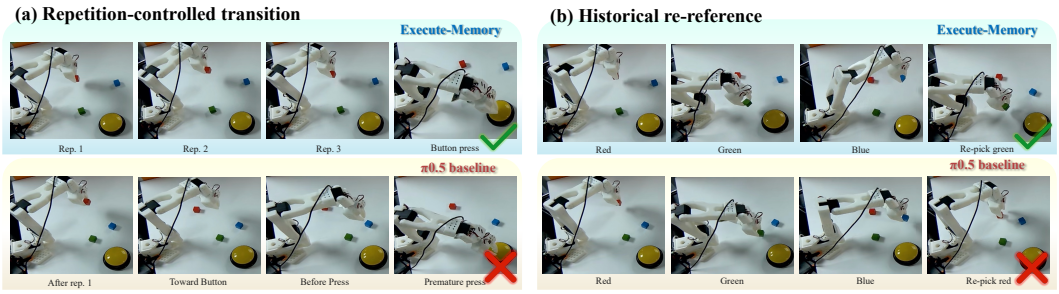

Figure 5: **Qualitative SO101 case studies.** Top: Execute-Memory. Bottom: $\pi_{0.5}$ baseline. **(a) Repetition-controlled transition.** The task requires three red-cube repetitions before a button press. Execute-Memory completes the required repetitions and then presses the button, whereas the baseline transitions to button pressing prematurely after the first pick. **(b) Historical re-reference.** We first construct a red–green–blue interaction history through staged sequential commands, and then issue a follow-up query asking the robot to pick the cube from the second interaction. Execute-Memory re-picks the green cube, whereas the baseline re-picks red.

4.5 Real-robot case study

Figure 5 shows two qualitative SO101 cases. The demonstrations illustrate deployment of Execute-Memory on a physical robot and are not included in the RoboMME aggregate results or used to estimate real-robot success rates. Both methods use the same fine-tuned $\pi_{0.5}$ policy weights, which remain frozen during evaluation.

5 Conclusion

We introduced *Execute-Memory*, which turns embodied history into explicit execution state rather than leaving its consequences implicit in a learned model. Persistent entities and accepted events are queried by compositional programs, re-grounded in the current scene, and realized by a frozen VLA. Across RoboMME and longer-history stress settings, this interface improves task-level reliability over context- and subgoal-based baselines. These results highlight the importance of translating remembered history into explicit execution requirements for reliable long-horizon control.

Our current scope is deliberately narrower than unrestricted task-program synthesis. Execute-Memory relies on registered task-family compositions and observation-based event contracts, and the stress tests evaluate longer histories within these supported families rather than arbitrary unseen task semantics. The SO101 study is likewise qualitative rather than a large-scale real-robot evaluation. Extending these explicit execution semantics to learned event proposals, broader task compositions, and more diverse physical systems is an important direction, provided that evidence provenance and occurrence-level consumption remain explicit.

AI Use Statement

Generative AI tools assisted with manuscript drafting, revision, organization, and formatting, and provided feedback on conceptual framing, experimental design, and the interpretation of results. The authors take responsibility for the final text, claims, and reported results.

Ethics Statement

This work studies robotic manipulation in simulation and includes qualitative demonstrations on a physical SO101 arm. The study does not involve human subjects, personal data, or sensitive personal information. We follow the benchmark and platform usage guidelines.

Reproducibility Statement

The paper, appendix, configuration tables, and supplementary materials describe the method, evaluation protocol, task settings, and recorded execution receipts used to obtain the reported results. These materials are intended to support independent reproduction of the experiments.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, et al. Qwen3-VL technical report, 2025. URL https://arxiv.org/abs/2511.21631.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-language-action models transfer web knowledge to robotic control, 2023a. URL https://arxiv.org/abs/2307.15818.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale. In *Robotics: Science and Systems XIX*, RSS2023. Robotics: Science and Systems Foundation, July 2023b. doi: 10.15607/rss.2023.xix.025. URL http://dx.doi.org/10.15607/RSS.2023.XIX.025.

Tianxing Chen, Yuran Wang, Mingleyang Li, Yan Qin, Hao Shi, Zixuan Li, Yifan Hu, Yingsheng Zhang, Kaixuan Wang, Yue Chen, Hongcheng Wang, Junjie Wang, Tianhang Yang, Renjing Xu, Ruihai Wu, Yao Mu, Yaodong Yang, Hao Dong, and Ping Luo. RMBench: Memory-dependent robotic manipulation benchmark with insights into policy design, 2026. URL https://arxiv.org/abs/2603.01229.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. RoboMME: Benchmarking and understanding memory for robotic generalist policies, 2026. URL https://arxiv.org/abs/2603.04639.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An embodied multimodal language model, 2023. URL https://arxiv.org/abs/2303.03378.

Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. SAM2Act: Integrating visual foundation model with a memory architecture for robotic manipulation, 2025. URL https://arxiv.org/abs/2501.18564.

Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks, 2019. URL https://arxiv.org/abs/1903.03878.

Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Tan, Lawrence Chen, Quan Vuong, Ted Xiao, Pannag Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems XX*, RSS2024. Robotics: Science and Systems Foundation, July 2024. doi: 10.15607/rss.2024.xx.090. URL http://dx.doi.org/10.15607/RSS.2024.XX.090.

Qiao Gu, Alihusein Kuwajerwala, Sacha Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aditya Agarwal, Corban Rivera, William Paul, Kirsty Ellis, Rama Chellappa, Chuang Gan, Celso Miguel de Melo, Joshua B. Tenenbaum, Antonio Torralba, Florian Shkurti, and Liam Paull. ConceptGraphs: Open-vocabulary 3d scene graphs for perception and planning, 2023. URL https://arxiv.org/abs/2309.16650.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022. URL https://arxiv.org/abs/2207.05608.

Myungkyu Koo, Daewon Choi, Taeyoung Kim, Kyungmin Lee, Changyeon Kim, Younggyo Seo, and Jinwoo Shin. Hamlet: Switch your vision-language-action model into a history-aware policy, 2025. URL https://arxiv.org/abs/2510.00695.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control, 2022. URL https://arxiv.org/abs/2209.07753.

Matthias Mayr, Francesco Rovida, and Volker Krueger. Skiros2: A skill-based robot control platform for ros. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6273–6280. IEEE, Oct 2023. doi: 10.1109/iros55552.2023.10342216. URL http://dx.doi.org/10.1109/IROS55552.2023.10342216.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schoelkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Buchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, Joao Silverio, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle

Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi Jim Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R Sanketi, Patrick Tree Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Martin-Martin, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shuran Song, Sichun Xu, Siddhant Haldar, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, and Zipeng Lin. Open x-embodiment: Robotic learning datasets and rt-x models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, May 2024. doi: 10.1109/icra57147.2024.10611477. URL http://dx.doi.org/10.1109/ICRA57147.2024.10611477.

Jiaqi Peng, Xiqian Yu, Delin Feng, Yuqiang Yang, Wenzhe Cai, Jing Xiong, Ganlin Yang, Jinliang Zheng, Jiafei Cao, Xueyuan Wei, Jiangmiao Pang, Yuan Shen, and Tai Wang. Cortex: A bidirectionally aligned embodied agent framework for long-horizon manipulation, 2026. URL https://arxiv.org/abs/2607.05377.

Shaoting Peng, Margaret X. Wang, Julie A. Shah, and Nadia Figueroa. Object permanence filter for robust tracking with interactive robots, 2024. URL https://arxiv.org/abs/2403.08231.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning, 2023. URL https://arxiv.org/abs/2307.06135.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. MemoryVLA: Perceptual-cognitive memory in vision-language-action models for robotic manipulation, 2025. URL https://arxiv.org/abs/2508.19236.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, May 2023. doi: 10.1109/icra48891.2023.10161317. URL http://dx.doi.org/10.1109/ICRA48891.2023.10161317.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. MemER: Scaling up memory for robot control via experience retrieval, 2025. URL https://arxiv.org/abs/2510.20328.

Ultralytics. Ultralytics yolo11, 2024. URL https://docs.ultralytics.com/models/yolo11/.

Ganlin Yang, Zhangzheng Tu, Yuqiang Yang, Sitong Mao, Junyi Dong, Tianxing Chen, Jiaqi Peng, Jing Xiong, Jiafei Cao, Jifeng Dai, Wengang Zhou, Yao Mu, and Tai Wang. EventVLA: Event-driven visual evidence memory for long-horizon vision-language-action policies, 2026. URL https://arxiv.org/abs/2606.20092.

Yihan Zeng, Minghao Ye, Yiyuan Chen, Yide Shentu, Philipp Wu, Zike Yan, and Zhongyu Li. KEMO: Event-driven keyframe memory for long-horizon robot manipulation with vla policies, 2026. URL https://arxiv.org/abs/2606.23589.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Yi Nie, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Jiaji Rao, Zhengru Fang, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering frozen vlas into reliable manipulation primitives via memory-guided agents, 2026. URL https://arxiv.org/abs/2607.08448v5.

APPENDIX

A ADDITIONAL EXPERIMENTAL ANALYSIS

The main text reports the official task-wise success profile and the scaling curves. This appendix provides a paired diagnostic view of the same fixed main results: the gain over the best non-oracle comparison and the remaining capacity-reference difference.

A.1 QUALITATIVE PICKXTIMES CASE STUDY

Figure 6 shows selected observations from an independent successful Execute-Memory rollout of the two-cycle PickXtimes setting. The instruction requires repeated pick–place interactions with the same cube, followed by a button press. This illustrates why object identity and execution progress play different roles: the cube remains the action subject across repetitions, while completing a repetition changes the outstanding obligation. The runtime uses interaction evidence to track progress and current grounding to supply the next local subgoal. The rollout receipt records terminal success; the selected images illustrate the scene progression rather than identify exact event-admission boundaries. This qualitative run is separate from the aggregate results in the main tables.

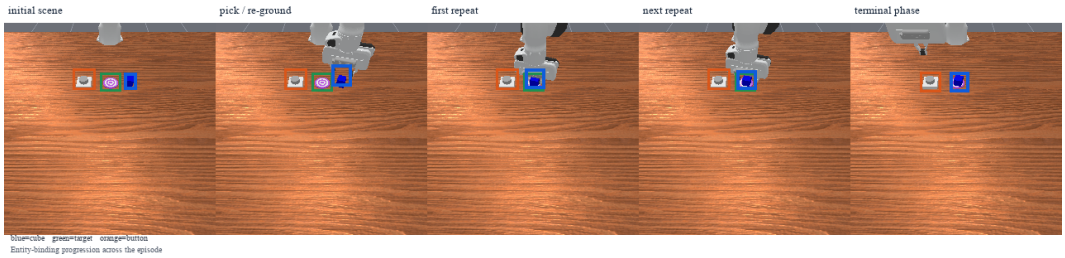

Figure 6: **Repeated interaction with a persistent object role.** Selected frames from a successful two-cycle PickXtimes rollout. Illustrative boxes identify the cube (blue), target (green), and button (orange); they are not raw detector outputs. The final panel shows a late observation, not necessarily the instant of button contact.

A.2 EVALUATION AND DEPLOYMENT CONFIGURATION

The official evaluation uses the following frozen configuration. Execute-Memory does not train a new low-level policy: it reuses the frozen $\pi_{0.5}$ policy and GroundSG execution interface.

A.3 STRESS-SUBSET CONFIGURATION

The custom stress evaluation changes only event/history scale. Camera, robot, checkpoint, evaluator, physical rules, and action-budget policy are shared.

A.4 EVENTSPEC AND EVIDENCE CONFIGURATION

The event contract separates observation-based admission from later phase-scoped consumption. Admission determines whether an occurrence can enter the canonical ledger; consumption determines whether that accepted occurrence can satisfy the active obligation again.

A.5 RUNTIME RECEIPT SCHEMA

Each rollout stores the following auditable fields. These records describe the execution trace and do not provide simulator truth as an Execute-Memory input.

Training boundary. Execute-Memory does not train a new low-level policy. The evaluation reuses the frozen $\pi_{0.5}$/GroundSG execution interface. The deployed components are listed in Table 4.

Table 4: Simulation evaluation and deployment configuration.

Item	Configuration
Benchmark	RoboMME validation split
Official tasks	16
Official evaluation	30 episodes per task and seed; three seeds (90 per task)
Execution budget	1300 steps
Robot	Panda with wrist camera
Observation	legal RGB/RGB–D, robot state, action feedback
Frozen policy	public frozen $\pi_{0.5}$ checkpoint (Physical Intelligence et al., 2025)
Grounding interface	GroundSG coordinate subgoal
Observation processing	fixed legal observation stride
Detector	YOLO11n-seg object segmentation provider (Ultralytics, 2024)
Identity/reappearance	public identity-association provider
Temporal provider	public temporal-geometry provider
Pair-temporal provider	public paired-boundary provider
Qwen model	Qwen3-VL-4B-Instruct
Qwen adapter	grounded-subgoal adapter from the RoboMME evaluation setup
Oracle	privileged evaluation-only channel; never passed to Execute-Memory
Stress evaluation	30 episodes per level in the fixed stress batch; zero-shot

Table 5: Custom zero-shot stress-subset configuration.

Subset	Scale settings
PickXtimes	$N \in \{2, 5, 8, 10\}$
VideoUnmaskSwap	$M \in \{1, 3, 5, 7\}$
VideoRepick	$N \in \{2, 4, 6, 8\}$
SwingXtimes	$N \in \{3, 4, 5, 6\}$
Shared conditions	same camera, robot, checkpoint, evaluator, physical rules
Changed factor	event/history scale only
Evaluation	official-task checkpoints reused zero-shot; no stress-level fine-tuning
Fairness	baseline, oracle, and Execute-Memory share instance manifests and action budgets
Oracle role	privileged capacity reference, not a learned baseline

B Program IR and Operator Specification

B.1 Query and program objects

The compiler consumes an instruction u and the registered event vocabulary E. The following notation is a semantic projection of the query specification, not a literal enumeration of constructor fields:

$$Q = (F, \mathsf{subj}, p, \mathsf{scope}, \mathsf{op}, k, \mathsf{anchor}, \mathsf{answer}), \tag{7}$$

where F is a family tag, subj and anchor are typed role descriptors, p is a registered predicate, scope is the temporal scope, op is a typed operator, k is an optional count or ordinal, and answer is the expected result type. The implementation additionally retains the source prompt, unsupported reason, and optional count/termination fields. The program's roles and constraints can be organized schematically as

$$P = (\mathcal{R}, \mathcal{C}, \mathcal{O}, \mathcal{H}, \mathcal{Z}), \tag{8}$$

with role declarations $\mathcal{R}$, semantic/geometric constraints $\mathcal{C}$, an operator graph $\mathcal{O}$, a phase graph $\mathcal{H}$, and evidence obligations $\mathcal{Z}$. A phase node stores its required inputs, completion predicate, failure predicate, and successor relation; it does not store hidden simulator progress.

B.2 Operators and typing

The semantic vocabulary below is realized by query evaluators and registered family constraints. It is not an unrestricted operator grammar accepted by a single general-purpose parser:

Table 6: Representative EventSpec and evidence contracts.

Event	Roles	Evidence	Admission condition	Runtime use
Attachment	tool/object	TCP proximity, gripper closure, relative motion	causal post-grasp consistency	pick-to-transport transition
Release	object/support	gripper opening, separation, support/height change	no stale or contradictory evidence	place completion
Stable arrival	held object/target	RGB–D 3D relation, dwell, low speed	hysteresis and source-time consistency	waypoint/target transition
Button completion	button/tool	contact geometry, press motion, departure	contact/transition evidence with valid source timing	phase eligibility and reuse are checked by the runtime; terminal interaction

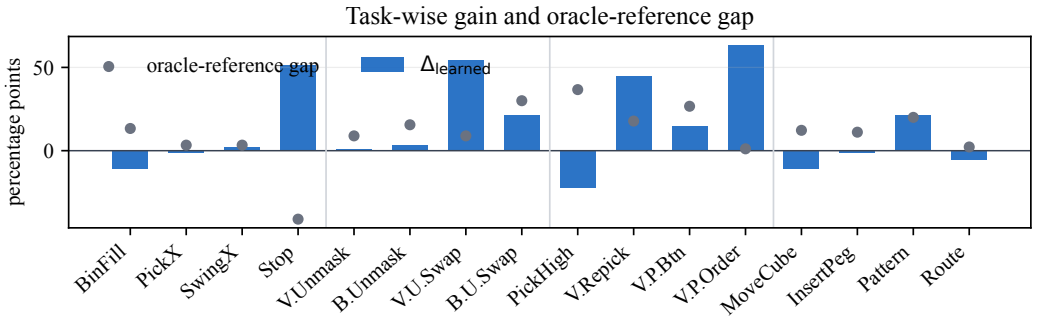

Figure 7: **Paired task-wise gain and oracle-reference gap.** Blue bars show the Execute-Memory gain over the best non-oracle comparison row in the main table; gray points show the difference from the GroundSG+oracle reference.

`Resolve`	entity or role reference	entity/role → binding
`Count`	cardinality of matching events	event set → integer
`Nth`	ordinal event selection	ordered event set × integer → event
`Before/After`	temporal relation	event pair → predicate
`Repeat`	repeated lifecycle execution	predicate × integer → sequence
`Set`	finite member iteration	entity set → iterator
`Sequence`	ordered subprograms	program list → program
`Timing`	temporal opportunity constraint	interval → constraint
`Topology`	ordered spatial relation	graph/path → constraint
`Wait/Verify`	observation-gated completion	predicate → phase guard

Queries over canonical events return typed bindings, counts, constraints, or unresolved status; control operators use those answers and runtime state. For the overview example, `Nth(placement(green_cube,target),2)` is evaluated after ordering the matching canonical placement events by source time. A repeated interaction can compose `Count` and `Repeat`; neither operator itself performs an action.

B.3 Compilation and Evaluation

The compiler constructs roles and phases around the query, for example

$$(Q, P) = \mathsf{Compile}(u, \mathsf{E}), \qquad q_t = \mathsf{PhaseQuery}(P, Q, s_t). \tag{9}$$

The phase query is specialized at runtime and evaluated read-only:

$$(\theta_t, \chi_t) = [\![P, q_t]\!]\,(\mathcal{L}_t). \tag{10}$$

The same composition pattern can be instantiated with different registered predicates and measurement providers. Instruction parsing is template-constrained, and family compilers and adapters still supply task-dependent realization rules. The instruction-only recipe does not supply an episode's target answer or hidden progress. For procedural tasks, the current waypoint and topology compilers bind confirmed demonstration constraints to an episode-specific phase graph. Their ordered roles

Table 7: Per-rollout receipt fields.

Receipt field	Purpose
Episode/task identifier	joins the observation, action, and terminal records
Source frame/timestamp	preserves observation causality
Entity/identity binding	records the active role hypothesis
Event proposal	candidate predicate, roles, and evidence window
Admission outcome	accepted, rejected, unresolved, or abstained
Canonical event key	identifies an idempotent accepted occurrence
Active phase	records the program counter at the decision
Consumed event ids	records phase-scoped use
Grounded subgoal	current-scene action requirement
Policy request	emitted request and action-chunk metadata
Terminal status	benchmark result after rollout
Failure attribution	semantic, phase, grounding, policy, or terminal diagnosis

Illustrative Execute-Memory Query and Grounding Record

```
{
  "instruction": "Place the cube on the target used in its second recorded placement.",
  "program": {"operator": "Nth",
              "predicate": "placement(subject=cube)",
              "ordinal": 2,
              "return": "object",
              "answer_type": "entity"},
  "accepted_memory": [
    {"event": "placement", "subject": "<cube>",
     "object": "<target_1>", "source_time": "<t_1>",
     "event_key": "<key_1>"},
    {"event": "placement", "subject": "<cube>",
     "object": "<target_2>", "source_time": "<t_2>",
     "event_key": "<key_2>"}],
  "active_phase": "resolve_second_historical_target",
  "query_result": {"resolved_entity": "<target_2>",
                   "supporting_event": "<key_2>",
                   "status": "resolved"},
  "current_observation": {"source_frame": "<current_frame>",
                          "rgbd_available": true,
                          "robot_state_available": true},
  "grounding": {"current_role": "<target_2>",
                "current_geometry": "<current_scene_geometry>",
                "intent": "PointIntent"},
  "groundsg_subgoal": "<resolved local subgoal>",
  "policy": "frozen pi0.5"
}
```

Figure 8: **Representative Execute-Memory query and grounding record.** The interface illustration shows how a typed program reads accepted entity/event memory, resolves an episode-dependent answer, and re-grounds the requirement in the current scene before passing a local subgoal to the frozen policy. It is an interface contract, not a free-form VLM prompt.

and edge constraints are compiled uses of memory, not facts inferred from the instruction or an independent authority about the world. The abstract $\mathsf{Compile}(u, \mathsf{E})$ notation denotes the recipe/interface stage; family initialization can defer this materialization until its legal evidence is available. Once instantiated, execution position changes through runtime state rather than by treating the program as a replacement for history.

B.4 Illustrative Execute-Memory Query and Grounding Record

The following interface illustration shows the semantic chain from an instruction and structured ledger to a current-scene grounded subgoal. It is a representative record, not a free-form VLM prompt used by the official evaluation.

The record illustrates the Execute-Memory interface rather than a free-form prompt used by the official evaluation. Historical records determine the typed answer and active requirement, while the current legal observation determines executable geometry. GroundSG and the frozen policy realize the resulting local subgoal. Here, `accepted_memory` contains canonical-ledger events; `event_key` identifies an interaction occurrence rather than an action request or consumption phase; and `supporting_event` remains readable by historical queries while phase-

scoped consumption prevents the same occurrence from completing the active obligation twice. `current_geometry` comes from the current legal RGB/RGB–D observation rather than a historical coordinate. An unresolved query defers emission instead of producing a forced subgoal.

C Event Registry and Evidence Contracts

C.1 Registered EventSpec

The registration contract for a predicate available to compilation, proposal, or admission can be summarized as

$$\mathsf{EventSpec} = (p, R_{\mathrm{subj}}, R_{\mathrm{obj}}, \Phi, \mathcal{C}_e, \mathcal{T}_e, \mathcal{A}_e), \tag{11}$$

where p is the predicate name and type, R_{subj} and R_{obj} are admissible role types, Φ lists legal channels and required evidence roles, $\mathcal{C}_e$ contains identity, geometry, confirmation, hysteresis, and contradiction rules, $\mathcal{T}_e$ contains source-time and causality rules, and $\mathcal{A}_e$ contains abstain conditions. The registry also records a schema version and the expected event key.

Registered schemas are validated for predicate type, role binding, legal observation channels, evidence requirements, and temporal rules before provider outputs can enter admission.

C.2 Evidence contract

For an event requiring a state change, the evidence envelope records source frame f, timestamp τ, and the three logical roles

$$\Gamma = (o^{\mathrm{before}}, o^{\mathrm{boundary}}, o^{\mathrm{after}}). \tag{12}$$

The roles mean that the precondition is observed before the candidate boundary, the transition is supported at the boundary, and the postcondition is observed afterwards. Events whose schema does not require all three roles may mark a role as not applicable, but cannot replace a missing required role with a command request or an oracle field.

D Persistent Belief Representation

D.1 Belief state

The information retained by the object-centric providers can be written as

$$\mathcal{B}_t = \{b_i^t\}_{i=1}^{N_t}, \qquad b_i^t = (\hat{e}_i, \hat{x}_i^t, \hat{v}_i^t, \hat{z}_i^t, \Sigma_i^t, \eta_i^t), \tag{13}$$

where $\hat{e}_i$ is a persistent entity hypothesis, $\hat{x}_i^t$ is current geometry, $\hat{v}_i^t$ is motion when observable, $\hat{z}_i^t$ contains semantic state and relations, Σ_i^t captures uncertainty, and η_i^t stores identity, scene-epoch, and source provenance. Belief update uses legal RGB/RGB–D, robot state, and action observations in the source record:

$$\mathcal{B}_t = \mathsf{UpdateBelief}(\mathcal{B}_{t-1}, o_t^{\mathrm{legal}}). \tag{14}$$

D.2 Identity and scene epochs

Identity candidates and ambiguity status are represented explicitly; selected trace resolvers and shared contracts support reasoning over alternatives. These interfaces do not imply that every online provider enumerates a full hypothesis set. Agreement of actionable signatures permits a common current decision, not an irreversible merger of histories that may matter at a later phase. Historical demonstrations and execution observations retain scene/source provenance, so a historical role can be linked without treating its old coordinate as current geometry. Static anchors and propagated state may remain valid within their scene assumptions. Missing current depth must not be filled with an unrelated old pose. Visibility, motion, and uncertainty estimates are provider-dependent; their confidence fields are not assumed to be calibrated probabilities.

D.3 PROVIDER INTERFACES

For a phase query q_t, a family provider extracts task-family measurements:

$$x_t = \mathsf{FamilyProvider}_F(\mathcal{B}_t, o_t^{\text{legal}}, q_t). \tag{15}$$

A deterministic or learned event provider then returns one or more proposals

$$\widetilde{e}_t = (p, \widetilde{R}, [\tau_b, \tau_a], \widetilde{u}, \widetilde{\Gamma}), \tag{16}$$

with proposed predicate and roles, source-time interval, uncertainty, and evidence window. Proposals have no phase authority; every proposal type uses the same registered evidence and admission contract.

E PERCEPTION PROVIDER INSTANTIATION

The object-centric provider instantiates the belief interface with replaceable visual and geometric modules. A compact RGB/RGB–D segmentation detector (the current evaluation uses the public YOLO11n-seg family) proposes regions; depth projection and the calibrated camera–robot transform convert valid pixels into world-frame measurements. Temporal association combines appearance, geometry, motion, visibility, and scene-epoch consistency to update entity hypotheses. Robust geometric estimators are applied to the resulting point sets before a family provider proposes an event. These modules produce beliefs and proposals only: they cannot write canonical events, advance a phase, or read simulator identities, truth masks, oracle subgoals, reward, or success. The registry and evidence gate therefore remain unchanged if the detector, tracker, or geometric measurement provider is replaced.

The Qwen comparison path uses Qwen3-VL-4B-Instruct with the grounded-subgoal adapter from the RoboMME evaluation setup.

F CANONICAL LEDGER AND ADMISSION

F.1 CANONICAL EVENT

The semantic contents of an admitted canonical event are

$$e = (\mathrm{id}, p, R, [\tau_b, \tau_a], \Gamma, \rho, \kappa) \in \mathcal{E}_t^{\text{can}}, \tag{17}$$

where id is the event identifier, R are canonical role bindings, Γ is the source-linked evidence envelope, ρ contains provenance including scene and identity epochs, and κ contains confidence and consistency diagnostics. Canonical events are immutable and exactly-once with respect to their registered event key. The key identifies the interaction occurrence established by the evidence contract, not the phase that later consumes it. Repeated observations of one occurrence and later phase reads therefore retain the same key; a new key is created only when the evidence contract identifies a new interaction occurrence.

F.2 LEDGER STATE

Separating its semantic contents, the ledger is

$$\mathcal{L}_t = (\mathcal{V}_t, \mathcal{R}_t, \mathcal{E}_t^{\text{can}}, \mathcal{P}_t), \tag{18}$$

where $\mathcal{V}_t$ stores versioned entity states, $\mathcal{R}_t$ stores typed relations, $\mathcal{E}_t^{\text{can}}$ stores canonical events, and $\mathcal{P}_t$ stores provenance, schema-version, and receipt metadata. The ledger is append only for canonical events; state versions and relations may be superseded while their provenance remains available. State/relation components are views of the observation-maintained belief store; the admission pseudocode updates the canonical-event component. These state and relation versions retain belief/validity status; only the canonical-event projection provides accepted occurrences for completion and count queries. Physical truth is not implied by membership in the ledger.

F.3 Admission semantics

Admission is a conjunction of registered checks:

$$\mathsf{Gate}(\widetilde{e}, \mathsf{E}, \mathcal{B}_t, \mathcal{L}_t) = 1 \tag{19}$$

only if the predicate and roles are registered, source-time and evidence obligations hold, identity and geometry meet the registered criteria, and contradiction, staleness, confidence, and abstention checks pass. An existing matching event key is an idempotent replay, not a new occurrence; conflicting reuse is rejected. Otherwise the record is unchanged and admission returns no canonical event:

$$\mathsf{Admit}(\widetilde{e}; \mathsf{E}, \mathcal{B}_t, \mathcal{L}_t) = (\mathcal{L}_t, \bot). \tag{20}$$

A successful first admission canonicalizes the proposal and writes it once. Consumption is checked separately against the active program obligation, its role bindings and source-time scope, and the runtime's consumed-event record. The same event remains readable for historical queries; it cannot satisfy a new repetition merely because it is read again. Queries do not write world facts.

F.4 Receipts and replay

A receipt links proposal and event identifiers to source frames/timestamps, registry versions, checks, and admission outcomes. Runtime receipts separately record the consuming phase and hold/abstain reason, so recording consumption does not rewrite an immutable event. For a fixed record stream, provider outputs, configuration, and initial state, the deterministic query and consumption rules can be replayed. This is not a claim that simulator trajectories or learned perception are deterministic. Request indices can identify log entries but cannot replace physical evidence; oracle/evaluator data remain evaluation-only.

G Runtime and Intent Interfaces

G.1 Evidence eligibility and phase consumption

Physical completion uses accepted evidence eligible for the active obligation. Writing $\mathcal{U}(P, s, \mathcal{L})$ for this eligible subset,

$$\mathcal{U}(P, s, \mathcal{L}) \subseteq \mathcal{E}^{\mathrm{can}}, \qquad e \in \mathcal{U} \Rightarrow \mathsf{RoleTimeValid}_P(e, s) \wedge \neg \mathsf{Consumed}_P(e, s). \tag{21}$$

Eligibility depends on the phase and occurrence being certified. A physical event preceding the relevant action cannot complete that action. An event already in the ledger may be used when it is causally valid and not consumed for that obligation; it need not be re-admitted. Read-only facts also support role resolution, predecessor checks, and count-threshold logic. These logical steps follow the declared program, while physical completion cannot be inferred from requests, fixed frame indices, or policy-internal flags.

G.2 Action-chunk lifecycle

Execution state s and the policy's active chunk c are separate. The chunk retains its issuing phase so that a later phase change can trigger the declared handoff rule. The current registry distinguishes completing the issued chunk from interrupting it after an evidence-supported transition. In particular, task-specific StopCube and SwingXtimes paths retain chunk-completion behavior; this does not change their event thresholds or make elapsed action count into completion evidence. A new policy request waits for a permitted handoff. Program completion and benchmark terminal success are recorded separately.

G.3 Decision intent

The semantic requirements of an intent can be summarized as

$$\iota_t = (z_t, C_t, \Omega_t, E_t, G_t^+, G_t^-, K_t), \tag{22}$$

where z_t are action variables, C_t are spatial, temporal, and contact constraints, Ω_t is a validity or opportunity window, E_t are evidence references, G_t^+ and G_t^- are completion/failure predicates, and

K_t is the required executor capability profile. These requirements are distributed across the typed intent and program contracts, rather than being fields of one literal dataclass. The common intent object records semantic action, bound roles, source/scene identity, evidence references, confidence, status, and intent kind; the subclasses add point, timing, or relative-pose variables.

PointIntent. PointIntent represents a current actionable point or region and is the principal implemented path. Its adapter preserves the existing GroundSG coordinate semantics and emits the unchanged subgoal format consumed by frozen $\pi_{0.5}$.

TimedContactIntent. TimedContactIntent represents an action aligned to an opportunity window, such as the crossing/contact relation in StopCube. Its configured realization combines prediction, response calibration, readiness checks, and GroundSG dispatch under the shared intent contract.

RelativePoseIntent. RelativePoseIntent represents a relative pose and contact relation, as required by InsertPeg. It records the axis, endpoint, approach relation, tolerances, scene epoch, and evidence references through the shared adapter contract.

G.4 Capability and adapter semantics

For a resolved role θ_t, current observations produce

$$g_t = \mathsf{Ground}(\theta_t, \mathcal{B}_t, \bar{o}_t), \qquad g_t^{\pi} = \mathsf{IntentAdapter}_{K_t}(\iota_t), \qquad \mathbf{a}_t \sim \pi_{\mathrm{frozen}}(\bar{o}_t, g_t^{\pi}). \tag{23}$$

If the required capability is absent, the adapter returns `unsupported`; if evidence, identity, or geometry is unresolved, it returns $\perp$. Neither outcome certifies physical completion. Logical progress already justified by ledger facts is not rolled back merely because the next intent cannot yet be emitted.

For an executor K, evidence is decision-sufficient only when every trusted world hypothesis under consideration produces a resolved and feasible executable signature. Writing $\Sigma_{K,s}(\iota)$ for the fields required by executor K at the active phase, the shared deterministic contract checks

$$\mathsf{Sufficient}(\mathcal{B}_t, P, s_t, K) \iff \left[\begin{array}{l} \Sigma_{K,s_t}(\iota_t(\omega)) \text{ is resolved for every retained } \omega, \\ \Sigma_{K,s_t}(\iota_t(\omega_1)) = \Sigma_{K,s_t}(\iota_t(\omega_2)) \text{ for all retained } \omega_1, \omega_2, \\ \iota_t(\omega) \text{ is feasible under } K \text{ for every retained } \omega \end{array}\right]. \tag{24}$$

The retained set must be nonempty. Missing variables are unresolved, not wildcards. The contract is evaluated for the active phase before intent emission; agreement at one phase cannot justify erasing distinctions needed at a later phase. This is an emission-sufficiency gate, not a proof of physical correctness.

TimedContactIntent keeps separate crossing, contact, and dispatch intervals:

$$T_{\mathrm{cross}} \in [\ell_c, u_c], \qquad L_{\mathrm{contact}} \in [\ell_l, u_l], \qquad T_{\mathrm{dispatch}} \in [\ell_c - u_l,\ u_c - \ell_l]. \tag{25}$$

This interval is an arithmetic compatibility envelope, not a probability of successful contact or a robust timing guarantee. Task-specific calibration and observed timing errors must be evaluated separately. RelativePoseIntent for InsertPeg records an unsigned peg axis, insertion endpoint, approach side and axis, position and axis tolerances, insertion-depth interval, scene epoch, and evidence references. Endpoint colour is used only for legal correspondence; near/far and approach side are computed from the registered frame graph.

H Task-Family Composition

A task instance is represented as

$$\mathsf{TaskInstance} = (\mathsf{memory\ demand}, \mathsf{event\ bindings}, \mathsf{program\ composition}, \mathsf{intent\ sequence}). \tag{26}$$

RoboMME's T, S, O, and P descriptors denote temporal, spatial, object/reference, and procedural memory demands. They are not independent methods or intent types. The table summarizes family-level compositions; it is not a per-task implementation or success matrix for all sixteen benchmark tasks.

Table 8: Family-level semantic compositions. Intent types describe required information, not measured success or complete implementation coverage.

Family	Demand	Shared composition	Intent
Point reference	$O + T + S$	Resolve; Nth; Before/After	Point
Hidden relation	$O + T + S + P$	Resolve hidden relation; Sequence; reveal events	Point
Counted interaction	$T + O + S$	Count; Repeat; lifecycle events	Point
Set iteration	$O + T + S$	Set membership; per-member completion	Point
Ordered/topological	$T + P + S$	Sequence; Topology; waypoint events	Point
Interaction	$O + S + P$	mode; attachment; arrival guards	Point
Timed interaction	$T + S + P$	Timing; crossing/contact evidence	Timed
Rigid-body interaction	$O + S + P$	correspondence; relative geometry	Relative pose

I Full Runtime Algorithms

These algorithms summarize the execution semantics shared by the registered task-family implementations. Step_P applies the enabled-transition rule in Equation 4, including causal eligibility, phase-scoped consumption, and declared logical transitions. The action buffer c stores its issuing phase and lifecycle profile. Observation admission runs even while a chunk is active; a new intent is requested at a permitted handoff.

Algorithm A1 Detailed Execute-Memory runtime

Require: instruction u, registry E, family provider, event provider, intent adapter, frozen policy π, legal stream

$(Q, P) \leftarrow \mathsf{Compile}(u, \mathsf{E})$
$(\mathcal{B}_0, \mathcal{L}_0, s_0, c) \leftarrow (\emptyset, \emptyset, \mathsf{Initial}(P), \emptyset)$
for each legal observation o_t^{legal} **do**
  $\mathcal{B}_t \leftarrow \mathsf{UpdateBelief}(\mathcal{B}_{t-1}, o_t^{\text{legal}})$
  $\mathcal{L}_t \leftarrow \mathcal{L}_{t-1}$
  $q_t \leftarrow \mathsf{PhaseQuery}(P, Q, s_{t-1})$
  $x_t \leftarrow \mathsf{FamilyProvider}(\mathcal{B}_t, o_t^{\text{legal}}, q_t)$
  $\widetilde{\mathcal{E}}_t \leftarrow \mathsf{EventProvider}(\mathcal{B}_t, o_t^{\text{legal}}, x_t, q_t, \mathsf{E})$
  for each $\widetilde{e} \in \widetilde{\mathcal{E}}_t$ **do**
    $(\mathcal{L}_t, e) \leftarrow \mathsf{Admit}(\widetilde{e}; \mathsf{E}, \mathcal{B}_t, \mathcal{L}_t)$ ▷ Algorithm A2
  end for
  $s_t \leftarrow \mathsf{Step}_P(s_{t-1}, \mathcal{L}_t)$
  $c \leftarrow \mathsf{ReconcileChunk}(c, s_t, o_t^{\text{legal}})$
  if $\mathsf{Ready}(c)$ and s_t is active **then**
    $(\iota_t, g_t^\pi) \leftarrow \mathsf{ReGroundAndEmit}(P, \mathcal{L}_t, s_t, \mathcal{B}_t, o_t^{\text{legal}}, \mathsf{IntentAdapter})$
    if $\iota_t \notin \{\perp, \mathsf{unsupported}\}$ and $\mathsf{Compatible}(\iota_t, \pi)$ **then**
      $c \leftarrow \pi(\bar{o}_t, g_t^\pi)$; attach issuing phase and lifecycle profile
    else
      hold, abstain, or request evidence; leave justified program state intact
    end if
  end if
  execute the next permitted action in c, or hold
end for

Algorithm A2 Evidence-gated canonical admission

Require: proposal $\widetilde{e}$, registry E, belief $\mathcal{B}$, ledger $\mathcal{L}$

if schema, role, or legal-channel validation fails **then**
  return $(\mathcal{L}, \bot)$
end if
$k \leftarrow \mathsf{EventKey}(\widetilde{e}, \mathsf{E})$
if k is already in the canonical ledger **then**
  if the proposal matches the stored occurrence and source/scene identity **then**
    return $(\mathcal{L}, \mathcal{L}[k])$ ▷ idempotent replay, no new occurrence
  else
    return $(\mathcal{L}, \bot)$ ▷ conflicting key reuse
  end if
end if
if source-time causality or required evidence roles fail **then**
  return $(\mathcal{L}, \bot)$
end if
if identity or geometry consistency fails **then**
  return $(\mathcal{L}, \bot)$
end if
if a registered contradiction, staleness, or abstain condition holds **then**
  return $(\mathcal{L}, \bot)$
end if
if a confidence requirement of the active event schema fails **then**
  return $(\mathcal{L}, \bot)$
end if
$e \leftarrow \mathsf{Canonicalize}(\widetilde{e}, \mathsf{E}, \mathcal{B})$
$\mathcal{L} \leftarrow \mathcal{L} \oplus e$ ▷ the canonical-event commit boundary
return $(\mathcal{L}, e)$ ▷ phase consumption is checked by the runtime

Algorithm A3 Current re-grounding and intent emission

Require: program P, ledger $\mathcal{L}_t$, phase s_t, belief $\mathcal{B}_t$, legal observation o_t^{legal}, intent adapter

$Q \leftarrow \mathsf{StaticQuery}(P)$
$q_t \leftarrow \mathsf{PhaseQuery}(P, Q, s_t)$
$(\theta_t, \chi_t) \leftarrow [\![P, q_t]\!](\mathcal{L}_t)$
if a required binding/constraint is unsupported or emission evidence is insufficient **then**
    return $(\bot, \bot)$
end if
$g_t \leftarrow \mathsf{Ground}(\theta_t, \mathcal{B}_t, \bar{o}_t)$
if scene/identity consistency or current-geometry validity fails **then**
    return $(\bot, \bot)$
end if
$k \leftarrow \mathsf{RequiredIntentKind}(P, s_t, \chi_t)$
if the adapter lacks capabilities required for k **then**
    return $(\mathsf{unsupported}, \bot)$
end if
if $k = \mathsf{Point}$ **then**
    $\iota_t \leftarrow \mathsf{PointIntent}(g_t, \chi_t, \mathcal{E}_t^{\text{can}})$
else if $k = \mathsf{TimedContact}$ **then**
    $\iota_t \leftarrow \mathsf{TimedContactIntent}(g_t, \chi_t, \mathcal{E}_t^{\text{can}})$
else if $k = \mathsf{RelativePose}$ **then**
    $\iota_t \leftarrow \mathsf{RelativePoseIntent}(g_t, \chi_t, \mathcal{E}_t^{\text{can}})$
else
    return $(\mathsf{unsupported}, \bot)$
end if
if ι_t is unconfirmed or fails the configured decision-sufficiency checks **then**
    return $(\bot, \bot)$
end if
$g_t^\pi \leftarrow \mathsf{IntentAdapter}(\iota_t)$
if $g_t^\pi \in \{\bot, \mathsf{unsupported}\}$ **then**
    return $(g_t^\pi, \bot)$
end if
return (ι_t, g_t^π)

J Task and Event Composition Examples

The sixteen RoboMME tasks instantiate a small set of reusable compositions. The table makes explicit which historical operation, canonical event family, and intent interface are used by each task.

K Implementation Boundaries

The implementation instantiates the formal IR, event registry, evidence contracts, family runtimes, truth-isolation checks, and GroundSG adapters through provider backends. Providers may use different measurement implementations, but expose the same typed roles, evidence receipts, admission outcomes, phase guards, and intent interface. This separation lets perception and event providers evolve without changing the compositional program semantics.

Point, timed-contact, and relative-pose intents share the same adapter contract; the configured family provider supplies the measurements required by each intent. The frozen policy therefore receives a stable grounded subgoal format, while event registration, admission, phase consumption, and current grounding remain inspectable and replayable.

Truth isolation. All runtime inputs are drawn from the legal observation contract. Simulator actor identifiers and poses, truth segmentation, oracle subgoals, and evaluator reward, success, or scripted progress labels are excluded from inference. The program's own execution state is legitimate internal

Table 9: Task-level composition map used by the official RoboMME evaluation.

Task	Family	Memory/program operators	Canonical event / intent
BinFill	set iteration	Set; Repeat; Verify	release/containment; Point
PickXtimes	counted loop	Count; Repeat; Sequence	pick/place; Point
SwingXtimes	ordered loop	Sequence; Repeat; Verify	stable arrival; Point
StopCube	timed interaction	Count; Timing; Wait	crossing/contact; Timed
VideoUnmask	hidden relation	Resolve; Sequence	reveal/release; Point
ButtonUnmask	hidden relation	Resolve; Sequence	button completion; Point
VideoUnmaskSwap	hidden relation	Resolve; Before/After	swap/reappearance; Point
ButtonUnmaskSwap	hidden relation	Resolve; Sequence	button/container events; Point
PickHighlight	set iteration	Set; Sequence; Verify	member completion; Point
VideoRepick	reference loop	Resolve; Count; Repeat	identity-preserving pick/place; Point
VideoPlaceButton	reference query	Resolve; Nth; Sequence	placement/button; Point
VideoPlaceOrder	ordered reference	Resolve; Nth; Before/After	ordered placement; Point
MoveCube	interaction	Resolve; Sequence; Verify	attachment/release; Point
InsertPeg	rigid-body relation	Resolve; relative geometry; Verify	endpoint/pose; Relative pose
PatternLock	ordered topology	Sequence; Topology; Verify	waypoint arrival; Point
RouteStick	ordered topology	Sequence; Topology; Verify	waypoint arrival; Point

state. Request indices and chunk counters may be logged and used to manage a buffer, but do not certify physical events. Evaluation-only terminal handling is recorded separately.

K.1 Execution Diagnostics

Receipts record admission outcomes, duplicate-count checks, observed physical-completion transitions, decision coverage, evidence references, grounding status, and evidence-to-dispatch delay. These diagnostics support post-hoc attribution across event recognition, phase eligibility, current grounding, and action-chunk handoff; they do not by themselves prove physical correctness.

L Reproducibility and Extended Protocol

The evaluation records contain the frozen episode manifest, task and stress-subset definitions, checkpoint hashes, benchmark commit, camera and action-budget configuration, legal-observation contract, source-video/cache hashes, per-step receipts, and failure-attribution records. The manifest also identifies the evaluation condition and paired instance mapping. Oracle traces are stored in a separate evaluation channel and are never mounted in the Execute-Memory inference process.

For each episode we retain the terminal evaluator output only after the rollout, the Execute-Memory ledger and phase trace, event keys and evidence references, grounding record, emitted intent, policy action chunks, and timestamps. This separation supports separate inspection and post-hoc diagnosis of event recognition, phase transitions, grounding, and physical execution. Paired deltas are computed from the same manifest.

The custom stress subset keeps the official-task checkpoints and evaluation interfaces fixed while increasing event or history scale. Each level uses 30 episodes in the fixed stress batch; the reported values are not averaged over a separate seed set. Privileged oracle traces remain in a separate evaluation channel.

M Extended Problem Formulation

M.1 History-Dependent Embodied Execution

We consider a partially observable control process with instruction u. Let $\bar{o}_t$ denote the current sensory input: RGB or RGB–D, robot state, and legally available action feedback. Its source record is

$$o_t^{\text{legal}} = (\bar{o}_t, f_t, \tau_t), \tag{27}$$

where (f_t, τ_t) identifies the source frame and timestamp, rather than the arrival order of a processing request. With executed actions a_i, the legal history is

$$h_t^{\text{legal}} = (o_1^{\text{legal}}, a_1, \ldots, a_{t-1}, o_t^{\text{legal}}). \tag{28}$$

Truth isolation excludes simulator-only state, oracle answers, and evaluator-side progress signals from inference. Source metadata supports temporal reasoning and evidence alignment; a timestamp or frame index is not itself an observation that an interaction has completed.

Let $\mathcal{D}^*(h, u)$ be the nonempty set of next decisions that are optimal for the task return given history h. A direct witness of history dependence is a pair of histories with identical current decision inputs but incompatible optimal choices:

$$\exists h_t, h_t' : \quad \bar{o}_t = \bar{o}_t', \qquad \mathcal{D}^*(h_t, u) \cap \mathcal{D}^*(h_t', u) = \varnothing. \tag{29}$$

The instruction is fixed, as is any clock value exposed to a reactive controller. Equality concerns the current perceptual state, not the identities of records in the two histories. Disjoint optimal-decision sets rule out a common optimal choice based only on $(\bar{o}_t, u)$; merely choosing different distributions over otherwise equivalent actions would not establish this claim. The relevant historical difference may specify a different target or whether to repeat an operation instead of terminating.

M.2 Memory-Assisted Execution Objective

Let μ_t denote any summary computed from the legal history and instruction,

$$\mu_t = \mathsf{Mem}(h_t^{\text{legal}}, u). \tag{30}$$

For a downstream executor K, a memory-assisted system emits a consumable decision or abstains:

$$\begin{aligned} &d_t = \mathsf{Decide}(u, \bar{o}_t, \mu_t) \in \mathcal{D}_K \cup \{\perp\}, \\ &a_t \sim \pi_K(\bar{o}_t, \mathsf{Adapter}_K(d_t)) \quad \text{if } d_t \neq \perp. \end{aligned} \tag{31}$$

The objective is to retain historical distinctions that change the executable decision, such as referents, completed interactions, or ordering constraints, without requiring the executor to reconstruct the full history. A summary that aliases histories requiring incompatible decisions loses information needed by the executor. Uncertainty that leaves the current decision unchanged need not prevent action; insufficient evidence or incompatible supported decisions permits $\perp$ and further observation. However, always abstaining is not a useful solution. Evaluation must jointly consider task completion, errors among emitted decisions, decision coverage, and the delay incurred while obtaining evidence. RoboMME provides the task-level characterization used in this work (Dai et al., 2026).