EVOWEAVE: COMPOSING REUSABLE AGENT EXPERIENCE THROUGH OPERATION LEVEL GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Different tasks often require interactive agents to perform similar operations at intermediate steps. Retrieving an entire past workflow can introduce irrelevant steps, whereas retrieving isolated operations can discard the dependencies needed to reuse them. We introduce EvoWeave, a framework for experience memory that represents abstract states as nodes and reusable operations as directed edges with input and output contracts, provenance, and capability references. A planner decomposes the current task into a dependency graph; operation retrieval and bounded directed search then assemble relevant experience fragments. An execution agent can inspect these fragments and their referenced capabilities, while reflection separates concrete execution evidence from reusable abstractions. We study the framework across four benchmarks, five evaluation settings, and three language model backbones. Across backbones, AppWorld Normal and Challenge task goal completion average 51.91 and 27.99, respectively, exceeding the baseline without memory by 6.99 and 4.99 percentage points. An AppWorld ablation associates directed connections with an additional improvement of 2.08 points over independent operation cards. The results motivate composition of reusable operations while exposing tradeoffs in retrieval overhead and conditions for valid reuse. The anonymous code repository is available at https://anonymous.4open.science/r/EvoWrave-C634/.

1 INTRODUCTION

Language model agents solve interactive tasks by interleaving reasoning with actions and environmental feedback (Yao et al., 2022). Although task goals vary, many constituent operations may recur across tasks: locating a contact, filtering records, reading an attribute, modifying an object, or submitting a result. An agent that has already performed these operations should be able to reuse that experience when pursuing a different goal. Approaches based on experience therefore reuse insights or workflows that an agent has learned from prior interactions (Zhao et al., 2024a; Wang et al., 2024; Ouyang et al., 2026), making the organization of experience a central design choice.

One part of that choice is granularity: the unit at which experience is stored and retrieved, which decides how much of a past task is carried into a new one. Figure 1 illustrates this choice. A new request to contact a collaborator may require finding contact information, drafting an email, and sending a message. A historical workflow containing these operations, however, may also include unrelated research, profile inspection, or record keeping, so supplying that entire workflow couples useful experience to unnecessary context. We therefore propose retrieving the relevant operations separately, which allows partial reuse across historical tasks. We use three terms consistently: an *operation* is one abstract step of a task, such as sending a message; an *action* is the concrete environment call that realizes an operation during execution; and a *capability* is the external resource that such an action needs, such as an API definition or a library function. The same granularity principle then applies to capabilities: an operation can reference only the API definition or function required by its action, rather than an entire manual or code library. Longer context does not guarantee that a model uses the relevant parts of it (Liu et al., 2024), so we treat narrower retrieval as a hypothesis to be measured rather than a benefit to be assumed.

Fine grained reuse alone does not solve composition. By composition we mean assembling several separately retrieved operations into one executable sequence for the current task, such that every

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.

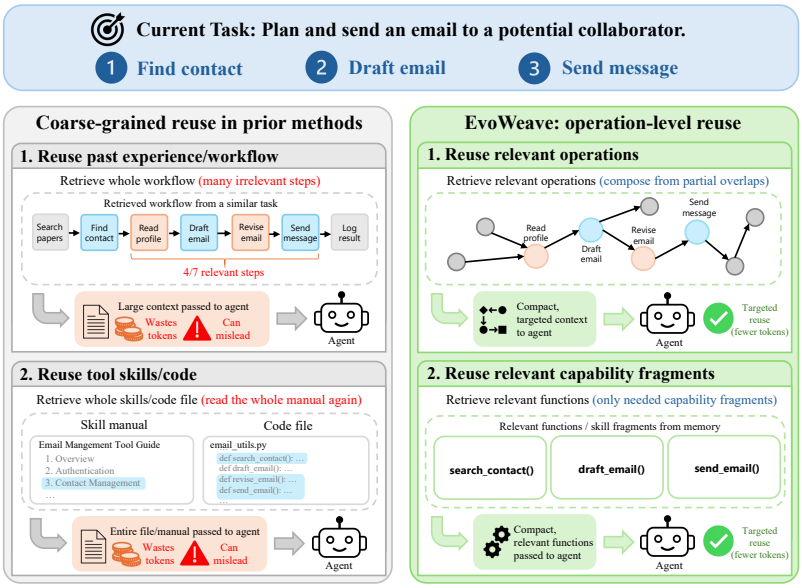

Figure 1: Motivation for reusing individual operations and capability fragments. The illustrative comparison contrasts retrieving an entire historical workflow or capability file with selecting relevant operations and functions for a new task. Selective reuse aims to reduce irrelevant context while supporting composition across partially overlapping experiences. The figure is conceptual; its step counts and anticipated benefits are not experimental measurements or claims about every prior method.

operation finds the inputs it requires either in the environment or in the outputs of the operations placed before it. Two individually relevant operations may disagree about the intermediate object they consume, or require an unobserved prerequisite. A path to a downstream operation does not establish that every input is available. Prior work has already explored reusable programs (Wang et al., 2023; Yuan et al., 2024), linked memories (Xu et al., 2026), and modular procedural memory (Han et al., 2025). What these representations do not retain is, for each reusable unit, the input conditions and the directed transitions that determine whether the unit can be executed after the units retrieved alongside it. The question we address is therefore more specific: how can operation level experience preserve its conditions and historical connections while remaining selectable for a new task?

We propose EvoWeave, which represents an operation as a directed transition between abstract states, annotated with its contract, evidence, and capability references. A current task plan specifies which steps directly depend on one another. These dependencies guide retrieval of historical operations and search for connecting fragments. The execution agent receives the assembled operation fragments as context, together with an explicit list of the plan dependencies that retrieval could not connect, and it can query the experience graph again during execution. Reflection retains concrete observations in task history while proposing abstract, versioned updates to shared experience.

Our contributions are threefold: (i) an operation level representation that separates reusable transitions from concrete execution evidence; (ii) a plan guided retrieval and connection procedure that retains input requirements, candidate consistency, and search failures; and (iii) an empirical analysis spanning three backbones and five evaluation settings, with ablations that distinguish operation decomposition, directed connections, and active retrieval. We report heterogeneous gains and resource tradeoffs rather than assuming that a graph representation improves every task.

2 Related Work

Experiential and procedural memory. ExpeL distills experiential knowledge (Zhao et al., 2024a), Reflexion uses verbal feedback (Shinn et al., 2023), and ReasoningBank organizes rea-

soning memories from previous interactions (Ouyang et al., 2026). CLIN maintains textual causal abstractions across trials, while Buffer of Thoughts retrieves reusable reasoning templates (Majumder et al., 2023; Yang et al., 2024). Synapse retrieves trajectories with abstracted states, Agent Workflow Memory induces workflows, and LEGOMem studies modular procedural memory (Zheng et al., 2024; Wang et al., 2024; Han et al., 2025). These approaches show that neither experience reuse nor decomposition originates with our work. EvoWeave focuses on operations as state transition edges and uses direct plan dependencies to retrieve and connect historical fragments. Our ablations examine these connections within the reported configurations.

Structured memory and retrieval. MemoryBank and Generative Agents connect persistent memories to later interactions and behavior (Zhong et al., 2024; Park et al., 2023). Mem0 and A-MEM investigate scalable storage and memory organization, while MemGPT manages context levels (Chhikara et al., 2025; Xu et al., 2026; Packer et al., 2023). Retrieval augmented generation provides external evidence; Self-RAG further learns when to retrieve and how to critique generations (Lewis et al., 2020; Asai et al., 2024). GraphRAG and HippoRAG organize retrieval through graphs (Edge et al., 2024; Gutiérrez et al., 2024). Closer to interactive agents, AriGraph combines semantic and episodic memory in a world model, and G-Memory traces collaborative experience through a graph hierarchy (Anokhin et al., 2024; Zhang et al., 2026). EvoWeave instead centers retrieval on operation contracts and historical transitions. Reachability supplies a candidate fragment, not a certificate that a live action is feasible.

Reusable capabilities and planning. Voyager, LATM, CRAFT, and Code as Policies explore reusable skills, tool creation, or executable programs (Wang et al., 2023; Cai et al., 2024; Yuan et al., 2024; Liang et al., 2023). LearnAct improves action functions using execution feedback, while Toolformer studies learning tool use (Zhao et al., 2024b; Schick et al., 2023). Skill Retrieval Augmentation distinguishes retrieving a skill, incorporating it, and completing the task (Su et al., 2026), supporting our separation of memory references from execution evidence. LLMCompiler and SwiftSage address planning and execution, and LATS searches action trajectories using environmental feedback (Kim et al., 2023; Lin et al., 2023; Zhou et al., 2023). Our bounded search connects stored operations rather than simulating future actions, and it requires no update to backbone parameters. AgentBoard's analysis of progress beyond final success further motivates examining category behavior and reuse alongside aggregate scores (Ma et al., 2024).

3 Method: EvoWeave

Figure 2 shows the flow from task planning to experience retrieval, capability access, execution, and reflection.

3.1 Experience as Operations with Explicit Contracts

Given a task q, an environment, and a concrete interaction history H_t, the agent must produce actions within an interaction budget. We distinguish its current plan P_t, which is a directed acyclic graph (DAG), from the historical experience graph $G = (V, E)$, which can contain cycles and multiple edges between states. A reusable operation is

$$e = (u, v, d, \mathcal{I}, \mathcal{O}, \mathcal{C}, \mathcal{R}, \pi), \tag{1}$$

where u, v are abstract source and target states, d is an operation description, $\mathcal{I}$ and $\mathcal{O}$ specify inputs and outputs, $\mathcal{C}$ records conditions and effects, $\mathcal{R}$ contains capability references, and π records provenance and versions. Task specific entities and values become placeholders in shared experience; their actual bindings remain in H_t. An operation with several required inputs retains their conjunction, including when a retrieved path supplies only one of them.

3.2 Retrieval and Connections Guided by the Plan

The planner uses q and H_t to specify remaining steps S_t and their direct dependencies D_t, without first conditioning the plan on available memory. For each step s_i, operation retrieval uses description embeddings f:

$$C_i = \mathrm{TopK}_{k=3}\{e \in E : \cos\big(f(s_i), f(d_e)\big) \geq \tau_{\text{retrieve}}\}. \tag{2}$$

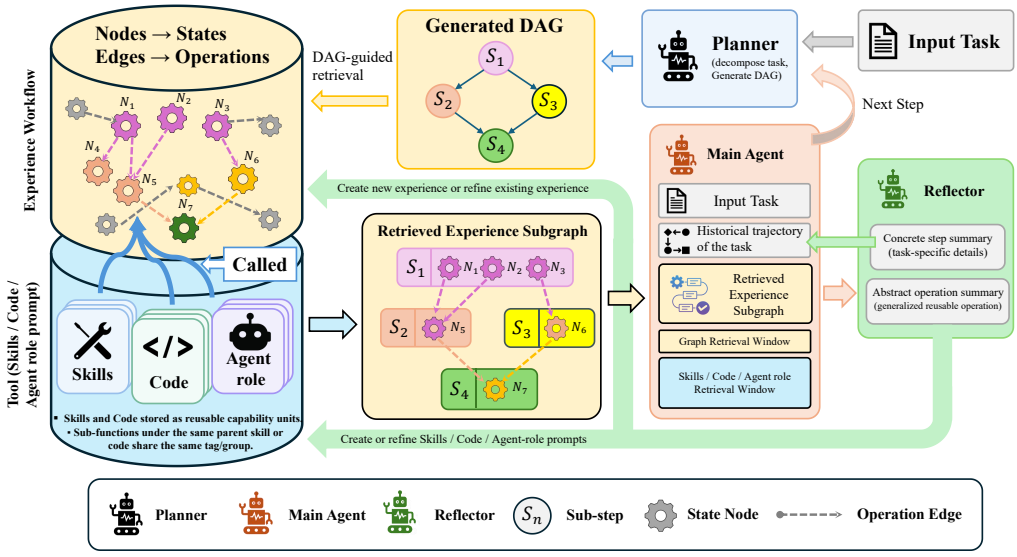

Figure 2: Overview of EvoWeave. A planner generates a task DAG that guides retrieval from an experience graph, whose nodes represent states and edges represent operations. The main agent uses retrieved experience and relevant capabilities, while the reflector separates concrete step summaries from abstract operation summaries. Shared memory updates follow the declared construction or evaluation protocol. Capability and role refinement in the diagram describe the broader architecture; evaluated implementation boundaries are detailed in Appendix B.

Filtering precedes ranking, and an empty candidate set is valid. Candidates retain edge identities, endpoints, and contracts; operations sharing a source state remain distinct.

For each direct dependency $(i, j) \in D_t$, consider a candidate pair $a \in C_i$, $b \in C_j$. A connection searches from the target state v_a to the source state u_b and retains both matched operations:

$$\Gamma_{ij} = a \circ \mathrm{BFS}_G(v_a, u_b) \circ b. \tag{3}$$

Breadth first search minimizes the number of connecting operation edges under fixed search limits. At most nine candidate pairs are examined per plan dependency. Ties use candidate similarity and stable identifiers. We do not add connection requests for transitive plan dependencies. An unsuccessful search reports either a gap or a budget truncation, instead of inventing an edge.

Connections sharing a plan step must use a consistent candidate to form one chain. If local choices cannot be reconciled, we retain separate fragments and expose a candidate conflict. This conservative procedure is not a global assignment of minimum total cost. Similarly, a graph path does not establish that all inputs are bound, entities are consistent, or concurrent writes are safe. The main agent checks these conditions against actual observations before acting.

3.3 Execution and Capability Access

The main agent receives the plan, retrieved operation cards, connecting fragments, and missing conditions. Before opening capability bodies, it can inspect an operation, retrieve neighbors, search descriptions, or request a path. Cards expose identifiers, contracts, provenance, and versions; subsequent requests load the relevant API or function body. Official API definitions remain immutable, with learned usage information stored separately. New code capabilities require validation and tests. The agent generates current task actions using these references and can reject a retrieved suggestion or request replanning. Events recording memory references are distinct from evidence that a stored program was directly executed.

3.4 Reflection and Controlled Memory Updates

Following execution, reflection produces two distinct artifacts. Concrete history records actual inputs, outputs, errors, and evidence references. Abstract experience records reusable descriptions, placeholders, conditions, and capability references. Every update branch performs this abstraction, including additions to existing operations. A separate similarity threshold τ_{merge} selects candidates for comparison; similarity alone does not authorize a merge. Contract checks distinguish compatible evidence additions, variants, and new operations. For example, requesting a payment and transferring money must remain distinct despite lexical similarity.

Failures retain the observed error and any evidenced recovery, without promoting expected outputs into successful outcomes. Embedding failures are distinguished from legitimate retrieval without a match. Memory proposals undergo structural, capability, and contract validation before publication. Having three persistent roles (planner, main, and reflector) does not imply exactly three model calls per round: active retrieval and merge decisions may introduce additional calls. Appendix B provides pseudocode and default parameters for the reference implementation.

4 Experiments

4.1 Benchmarks, Backbones, and Reporting Protocol

We analyze five settings from four benchmarks: AppWorld Test-Normal and Test-Challenge (Trivedi et al., 2024), WebArena (Zhou et al., 2024), ALFWorld (Shridhar et al., 2020), and ScienceWorld (Wang et al., 2022). AppWorld reports task goal completion (TGC) and scenario goal completion (SGC); WebArena reports success rate (SR). ALFWorld and ScienceWorld are reported with their own benchmark specific scores, whose scoring semantics need not be identical. Higher values are better for all seven columns. We average matching metrics equally over backbones and never combine heterogeneous metrics into one score. Differences are computed before rounding; display values are rounded afterward.

The backbones are DeepSeek-V3.2 (Liu et al., 2025), GPT-4.1-mini, and Qwen3.8-27B. Qwen is served locally on an institutional HPC cluster (anonymized for review); DeepSeek-V3.2 and GPT-4.1-mini are accessed through API calls. The analysis uses aggregate results for these configurations. Token budgets are not equalized by construction across methods; Appendix F reports the tokens each one actually consumes.

We compare No-Memory and Flat-Memory with AWM (Wang et al., 2024), ReasoningBank (Ouyang et al., 2026), Mem0 (Chhikara et al., 2025), ExpeL (Zhao et al., 2024a), and A-MEM (Xu et al., 2026). These comparisons concern the evaluated implementations and do not establish the maximum performance attainable by each method under other adaptations or budgets.

Memory construction protocol. No memory is built from training or development data. Every run starts from an empty memory and accumulates experience online during evaluation: test tasks are executed one at a time in the order in which the benchmark releases them, and experience written after a task becomes available to all later tasks in the same run. Each of the 15 combinations of backbone and evaluation setting is an independent run that starts empty, so no experience crosses backbones or benchmarks. Reflection records what the agent observes within an episode, namely API return values, environment errors, and observations, together with an episode level success judgement that a language model derives from the trajectory itself. Methods that need a success signal, such as ExpeL and AWM, take it from this same module, so no method receives a more reliable signal than another. The official task level score is computed after an episode terminates, is used only for reporting, and never reaches the memory of any method. Every memory based baseline runs under exactly this protocol, from the same empty start and the same task order, so the comparison isolates how experience is organized rather than how much data each method was allowed to see. The setting is therefore continual learning over an evaluation stream, applied identically to every method; Appendix A.3 records the protocol in full.

Main results are reported as mean $\pm$ standard deviation (SD). We report descriptive differences without significance claims. Appendix A details the evaluation scope, aggregation rules, and comparability of benchmark and category summaries.

Table 1: Main results reported as mean ± standard deviation (SD), with the ± SD terms shown as subscripts. All entries retain the original score scale from 0 to 100. Within each backbone block, the highest mean in a column is shaded and the second highest is underlined; tied values receive the same mark.

	AppWorld Normal		AppWorld Challenge		WebArena	ALFWorld	ScienceWorld
Method	TGC	SGC	TGC	SGC	SR	Score	Score
DeepSeek-V3.2							
No-Memory	$47.02_{\pm 2.75}$	$30.36_{\pm 3.30}$	$23.02_{\pm 1.97}$	$9.35_{\pm 1.83}$	$42.00_{\pm 1.36}$	$69.40_{\pm 2.41}$	$64.70_{\pm 2.52}$
Flat-Memory	$49.05_{\pm 3.03}$	$32.14_{\pm 3.68}$	$23.02_{\pm 2.16}$	$9.35_{\pm 2.01}$	$43.00_{\pm 1.49}$	$71.94_{\pm 2.59}$	$67.20_{\pm 2.77}$
AWM	$52.02_{\pm 2.92}$	$35.00_{\pm 3.62}$	$24.99_{\pm 2.12}$	$10.50_{\pm 1.99}$	$46.01_{\pm 1.44}$	$73.88_{\pm 2.47}$	$68.70_{\pm 2.67}$
ReasoningBank	$52.01_{\pm 3.99}$	$35.00_{\pm 4.95}$	$26.00_{\pm 2.92}$	$11.88_{\pm 2.76}$	$43.00_{\pm 1.97}$	$71.34_{\pm 3.46}$	$67.70_{\pm 3.66}$
Mem0	$47.50_{\pm 3.08}$	$30.71_{\pm 3.71}$	$22.49_{\pm 2.19}$	$9.06_{\pm 2.04}$	$42.51_{\pm 1.52}$	$69.40_{\pm 2.69}$	$65.20_{\pm 2.82}$
ExpeL	$50.95_{\pm 2.95}$	$33.93_{\pm 3.63}$	$24.99_{\pm 2.14}$	$10.50_{\pm 2.01}$	$45.00_{\pm 1.46}$	$76.12_{\pm 2.45}$	$74.10_{\pm 2.70}$
A-MEM	$47.98_{\pm 3.25}$	$31.07_{\pm 3.92}$	$23.02_{\pm 2.32}$	$9.35_{\pm 2.16}$	$43.00_{\pm 1.60}$	$70.90_{\pm 2.81}$	$66.70_{\pm 2.98}$
EvoWeave	$54.05_{\pm 2.88}$	$37.14_{\pm 3.63}$	$27.00_{\pm 2.13}$	$11.65_{\pm 2.02}$	$49.00_{\pm 1.43}$	$75.88_{\pm 2.45}$	$76.20_{\pm 2.65}$
GPT-4.1-mini							
No-Memory	$35.71_{\pm 3.18}$	$20.71_{\pm 3.53}$	$16.98_{\pm 2.22}$	$6.33_{\pm 2.03}$	$30.69_{\pm 1.58}$	$45.97_{\pm 3.07}$	$40.00_{\pm 3.03}$
Flat-Memory	$36.67_{\pm 3.52}$	$21.43_{\pm 3.93}$	$16.02_{\pm 2.42}$	$5.90_{\pm 2.21}$	$30.69_{\pm 1.74}$	$48.96_{\pm 3.37}$	$41.00_{\pm 3.33}$
AWM	$38.69_{\pm 3.42}$	$22.86_{\pm 3.86}$	$17.99_{\pm 2.38}$	$6.76_{\pm 2.18}$	$33.69_{\pm 1.69}$	$51.04_{\pm 3.27}$	$44.00_{\pm 3.21}$
ReasoningBank	$39.71_{\pm 3.42}$	$24.64_{\pm 3.92}$	$20.00_{\pm 2.41}$	$7.77_{\pm 2.22}$	$35.69_{\pm 1.69}$	$51.94_{\pm 3.23}$	$46.00_{\pm 3.18}$
Mem0	$35.71_{\pm 3.57}$	$20.71_{\pm 3.95}$	$16.02_{\pm 2.46}$	$5.90_{\pm 2.25}$	$30.69_{\pm 1.77}$	$45.07_{\pm 3.42}$	$40.00_{\pm 3.39}$
ExpeL	$38.69_{\pm 3.46}$	$22.86_{\pm 3.90}$	$17.99_{\pm 2.41}$	$6.76_{\pm 2.20}$	$32.71_{\pm 1.70}$	$54.03_{\pm 3.29}$	$48.00_{\pm 3.24}$
A-MEM	$36.67_{\pm 3.78}$	$21.43_{\pm 4.21}$	$16.98_{\pm 2.62}$	$6.33_{\pm 2.40}$	$31.70_{\pm 1.87}$	$48.06_{\pm 3.62}$	$42.00_{\pm 3.57}$
EvoWeave	$40.60_{\pm 4.59}$	$26.79_{\pm 5.03}$	$21.01_{\pm 3.15}$	$7.47_{\pm 2.87}$	$36.70_{\pm 2.29}$	$57.97_{\pm 4.47}$	$51.00_{\pm 4.39}$
Qwen3.8-27B							
No-Memory	$52.02_{\pm 3.03}$	$35.00_{\pm 3.75}$	$29.02_{\pm 2.26}$	$12.95_{\pm 2.17}$	$43.99_{\pm 1.50}$	$71.94_{\pm 2.60}$	$66.10_{\pm 2.77}$
Flat-Memory	$54.05_{\pm 3.32}$	$37.14_{\pm 4.18}$	$30.02_{\pm 2.50}$	$13.53_{\pm 2.41}$	$45.00_{\pm 1.64}$	$74.03_{\pm 2.80}$	$68.00_{\pm 3.05}$
AWM	$57.02_{\pm 3.18}$	$40.36_{\pm 4.09}$	$30.98_{\pm 2.42}$	$14.24_{\pm 2.35}$	$48.00_{\pm 1.59}$	$75.97_{\pm 2.67}$	$70.50_{\pm 2.94}$
ReasoningBank	$57.98_{\pm 3.15}$	$41.43_{\pm 4.06}$	$33.00_{\pm 2.42}$	$15.54_{\pm 2.38}$	$50.00_{\pm 1.58}$	$77.01_{\pm 2.61}$	$72.20_{\pm 2.91}$
Mem0	$52.98_{\pm 3.39}$	$36.07_{\pm 4.23}$	$29.02_{\pm 2.53}$	$12.95_{\pm 2.43}$	$45.00_{\pm 1.67}$	$71.94_{\pm 2.90}$	$67.00_{\pm 3.11}$
ExpeL	$55.95_{\pm 3.22}$	$39.29_{\pm 4.11}$	$30.98_{\pm 2.44}$	$14.24_{\pm 2.37}$	$47.00_{\pm 1.60}$	$78.96_{\pm 2.61}$	$74.10_{\pm 2.97}$
A-MEM	$54.05_{\pm 3.57}$	$37.14_{\pm 4.48}$	$30.02_{\pm 2.68}$	$13.53_{\pm 2.58}$	$46.01_{\pm 1.77}$	$74.03_{\pm 3.01}$	$69.00_{\pm 3.27}$
EvoWeave	$61.07_{\pm 4.36}$	$43.92_{\pm 5.59}$	$35.97_{\pm 3.33}$	$17.82_{\pm 3.25}$	$49.00_{\pm 2.17}$	$79.03_{\pm 3.72}$	$73.70_{\pm 4.02}$

4.2 Main Results

Table 1 shows that EvoWeave has the highest reported mean in 16 of the 21 combinations of backbone and metric. Across backbones, Normal TGC increases from 44.92 to 51.91 and Challenge TGC from 23.01 to 27.99 relative to No-Memory. In both cases the strongest baseline macro mean comes from ReasoningBank, over which EvoWeave gains 2.01 and 1.66 points, respectively. Averages across backbones for WebArena SR, ALFWorld Score, and ScienceWorld Score reach 44.90, 70.96, and 66.97, respectively, compared with 38.89, 62.44, and 56.93 without memory.

The gains are not uniform. ReasoningBank exceeds EvoWeave on DeepSeek and GPT Challenge SGC and on Qwen WebArena SR. ExpeL has the higher mean on DeepSeek ALFWorld and Qwen ScienceWorld. Some differences are small relative to the reported standard deviations. These exceptions matter: matching and connecting historical operations does not guarantee that the selected experience fits the current task or improves every backbone.

4.3 Category Performance

Figure 3 separates category behavior from benchmark averages. For EvoWeave, AppWorld Challenge ranges from 17.01 on Gmail to 36.83 on Spotify, while ALFWorld ranges from 61.57 for Use Two Objects to 79.26 for Examine in Light. ScienceWorld varies from 45.63 for Mendelian genetics to 88.41 for Living and Non-living Things. Thus, a strong aggregate does not imply uniform performance. The local axis ranges expose differences among methods without altering the original scores. All 15 panels for individual backbones appear in Appendix C, preserving reversals and the

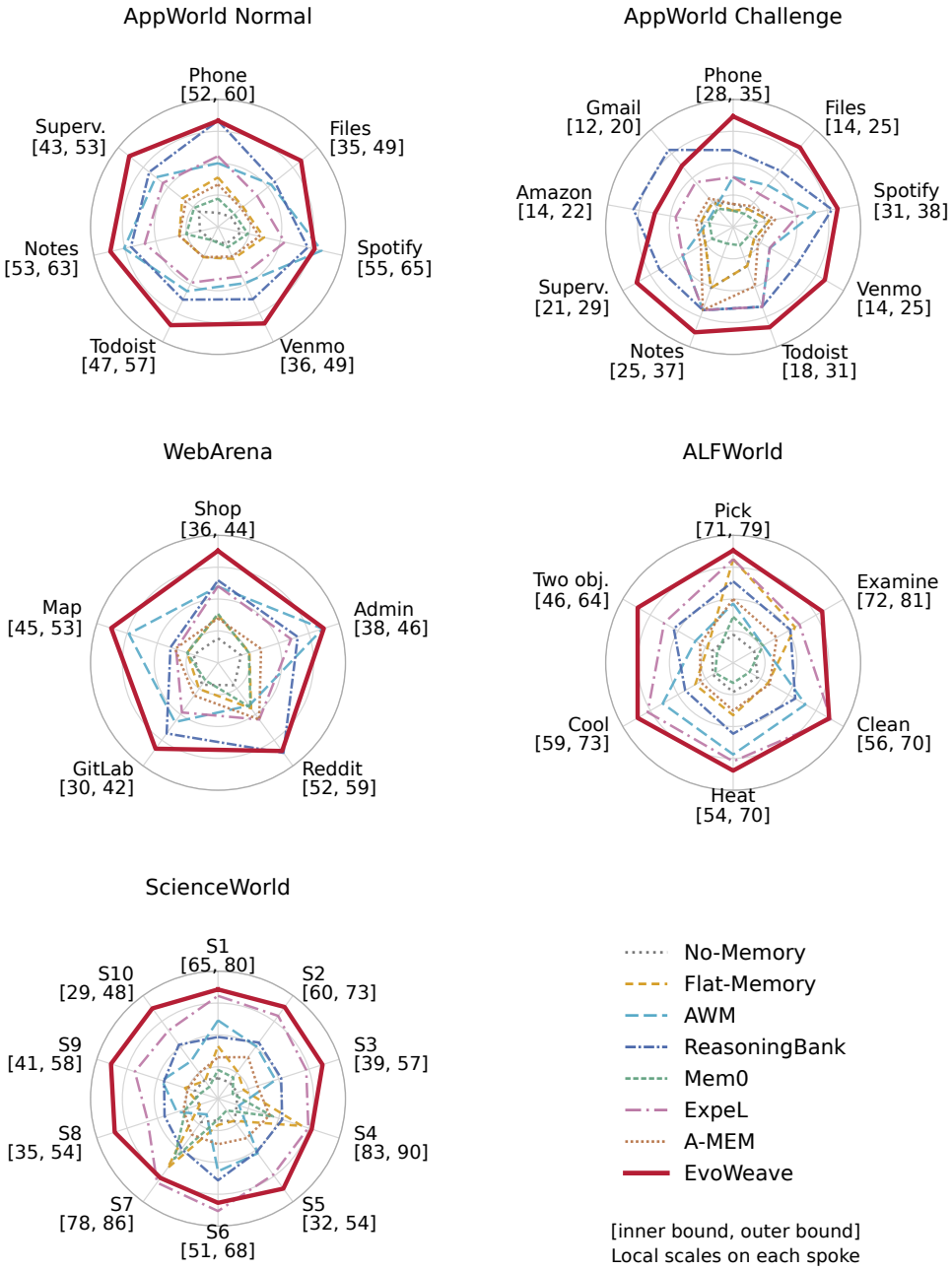

Figure 3: Category performance averaged equally over three backbones. Each spoke uses a local linear range determined by all eight methods; brackets give its inner and outer bounds in original score units. Compare methods on the same spoke: distances across spokes, panels, and polygon areas are not directly comparable. Labels and the scaling rule are defined in Appendix C. These category aggregates are distinct from Table 1; see Appendix A.

original Supervisor category. WebArena category means weighted by task counts sit slightly above the main table because the five site categories cover 764 of the 812 tasks and exclude the 48 cross site tasks; Appendix A quantifies that difference for each method.

Table 2: AppWorld ablation, averaged equally over backbones. Overall TGC uses weights based on task counts of 168 Normal and 417 Challenge tasks within each backbone. These configurations are specific to AppWorld; only point estimates are shown. Configurations are ordered so that the full system appears last, and the highest value in each column is shown in bold.

Configuration	Normal TGC	Challenge TGC	Overall TGC
A: No memory	44.92	23.01	29.30
B: Whole workflow RAG	46.59	23.02	29.79
C: Operation cards	47.93	24.72	31.39
D: C + directed connections	50.74	26.51	33.47
E: C + active retrieval	49.90	25.94	32.82
F: EvoWeave (full)	**51.91**	**27.99**	**34.86**

Table 3: Mean memory size over the five settings. Counts describe the reported graph snapshots, not an inferred average topology. Capability links connect the corresponding node type to operations. Counts of role nodes do not establish that learning of role prompts was evaluated.

Experience layer	States	Operations	
Mean count	27.0	157.6	
Capability layer	Code	Skills	Roles
Mean nodes	185.2	82.2	6.8
Mean links to operations	157.6	362.0	99.4

4.4 Ablation: Decomposition, Connections, and Active Retrieval

Table 2 reports six configurations. A removes memory and B retrieves whole workflows, so they coincide with the No-Memory and Flat-Memory rows of Table 1; C retrieves independent operation cards; D adds directed connections to C; E adds active retrieval to C without historical connections; and F is the full system, which adds active graph retrieval to D. Appendix D reports every backbone separately.

Operation cards improve Overall TGC from 29.79 for whole workflow retrieval to 31.39. Adding directed connections further raises it to 33.47, a difference of 2.08 points. Active retrieval without connections reaches 32.82, improving on C by 1.44 points. The full system reaches 34.86, another 1.39 points above D and 2.04 points above E. These comparisons support the hypothesis that relational context contributes beyond decomposition and additional lookup opportunities. The ordering of D and E is not reproduced by every backbone: E slightly exceeds D on DeepSeek, while GPT and Qwen keep D above E (Appendix D). All six configurations share the memory construction protocol of Appendix A.3, the same candidate pool, and the same capability library, so they differ only in how retrieved experience is connected and queried.

4.5 Memory Structure and Operation Reuse

Table 3 summarizes graph size. An average snapshot contains 27.0 state nodes and 157.6 operation edges; capability references extend across code, skills, and role nodes. These values describe the separation of transitions from the capabilities they reference, but do not measure successful composition or graph quality. The counts for individual settings vary substantially: AppWorld Challenge has 431 operations, compared with 14 in ALFWorld (Appendix E). These arithmetic summaries characterize size and composition without defining a shared adjacency structure.

Figure 4 summarizes recurring operations within task types. The equal type means of overlap/reference (%) are 48.5/49.2 for AppWorld Normal, 57.3/58.8 for Challenge, 35.3/36.7 for WebArena, 96.5/96.9 for ALFWorld, and 90.2/92.1 for ScienceWorld. Cross type comparisons are excluded from the overlap averages. These substantial within type overlaps motivate operation level abstraction: related tasks can reuse shared steps even when complete workflows differ. Appendix G defines the averages and provides all type specific values.

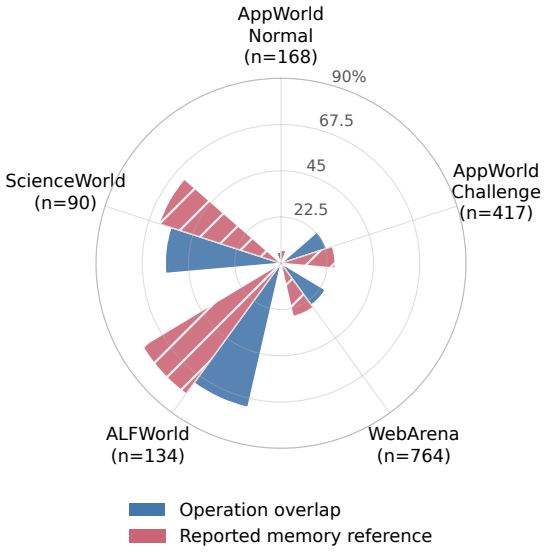

Figure 4: Mean operation overlap and reported memory referencing across task types. Each type receives equal weight; overlap comparisons are restricted to tasks of the same type. Labels show type counts, and bars show percentages on a scale from 0 to 100. Benchmark specific overlap proxies and reference rates are distinct measures (Appendix G).

4.6 Context and Runtime Tradeoffs

The efficiency results show 6,155.46 injected memory tokens per question for EvoWeave, versus 26,300.80 for Flat-Memory, 19,890.00 for AWM, and 11,872.00 for ReasoningBank: reductions of 76.60%, 69.05%, and 48.15%, respectively. These figures support selective context delivery, but the total input token measure falls by only 0.99% relative to No-Memory. EvoWeave has the fewest rounds (15.94) and the highest seconds per round (10.44), alongside lower KV reuse (45%). Less injected memory therefore does not by itself demonstrate lower latency or cost. Appendix F gives all efficiency metrics and explains the boundaries of token and runtime comparisons.

5 Conclusion

EvoWeave organizes reusable experience as operation edges with explicit contracts and connects retrieved fragments according to a current task dependency plan. This design preserves local reuse opportunities while exposing missing conditions and inconsistent connections. Results across three backbones and five settings are broadly favorable, and AppWorld ablations associate both operation decomposition and directed connections with improved task completion. The framework also highlights that smaller memory context can incur additional retrieval and reasoning overhead. Further work should examine generalization to unseen operation compositions and the effect of retrieval overhead on end to end performance.

Ethics Statement

This work follows the ICLR Code of Ethics. No human participants or animal experiments were involved in this study. All benchmarks used in this work, namely AppWorld, WebArena, ALFWorld, and ScienceWorld, are publicly released research benchmarks that run in simulated environments, and we used them in accordance with their respective licenses and usage policies. Throughout the research process, we made efforts to minimize potential biases and prevent discriminatory outcomes. No personally identifiable information was collected or utilized, and no experiments were performed that would pose potential privacy or security risks. We remain committed to ensuring transparency, ethical conduct, and research integrity throughout the study.

Reproducibility Statement

We have taken the following steps to make this work reproducible. The operation representation, the plan guided retrieval and connection procedure, and the reflection and memory update rules are defined in Section 3; Appendix B gives the corresponding pseudocode (Algorithm 1), the complexity bound on the number of bounded searches, and the complete default parameters of the reference implementation (Table 5), including retrieval and merge thresholds, candidate counts, search depth and expansion limits, and the budget for visible operation cards. All four benchmarks are publicly available, and we use their released task splits without modification; the benchmarks, backbones, baseline methods, metrics, and aggregation rules are enumerated in Section 4 and summarized in Table 4. Appendix A states the aggregation formulas, the task counts used for weighted means, and the exact sense in which category, graph, overlap, and efficiency summaries are computed over populations that differ from the main benchmark means. Appendices D to G report the per backbone tables, per setting graph statistics, token and runtime metrics, and category level overlap definitions underlying every figure in the main text. Anonymized code, the scripts that generate every table and figure from a single data file, and that data file itself are provided at the repository linked in the abstract, so all reported numbers can be regenerated from the released artifacts.

Two limitations bound exact reproduction and we state them explicitly. First, decoding is stochastic and two of the three backbones are served through provider APIs whose deployed checkpoints we do not control, so individual runs will not match digit for digit; we therefore report means with standard deviations and make only descriptive comparisons, as explained in Appendix A. Second, the baselines are re-implementations evaluated under our harness rather than numbers copied from their original papers. They all run under the common memory construction protocol of Appendix A.3, so the comparison holds the memory source, the task order, the success judgement module, and the per run isolation fixed. Token budgets are not equalized by construction; Appendix F reports the injected and total input tokens of every method, and EvoWeave has the smallest counts of both, so its results are not obtained from a larger context.

AI Use Statement

Large Language Models (LLMs) were utilized to support the writing and language refinement of this manuscript. In particular, the LLMs were employed to improve the clarity, readability, and overall linguistic quality of the text across different sections of the paper. Their assistance included rephrasing sentences, correcting grammatical issues, and improving the coherence and flow of the manuscript.

It should be emphasized that the LLM was not involved in developing the research ideas, designing the methodology, or conducting the experiments. All research questions, concepts, methodological decisions, and analyses were independently developed and carried out by the authors. The role of the LLM was strictly limited to language-related assistance and did not extend to the scientific content, experimental analysis, or interpretation of the data.

The authors remain fully responsible for all content presented in the manuscript, including any text that was generated or subsequently refined with the assistance of an LLM. All LLM-assisted text was reviewed by the authors to ensure compliance with relevant ethical standards and to avoid plagiarism or any form of scientific misconduct.

References

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363*, 2024.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pp. 9112–9141, 2024.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations*, volume 2024, pp. 54067–54089, 2024.

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

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569, 2024.

Dongge Han, Camille Couturier, Daniel Madrigal Diaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. Legomem: Modular procedural memory for multi-agent llm systems for workflow automation. *arXiv preprint arXiv:2510.04851*, 2025.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*, 2023.

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

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

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36:23813–23825, 2023.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. Clin: A continually learning language agent for rapid task adaptation and generalization. *arXiv preprint arXiv:2310.10134*, 2023.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Weihang Su, Jianming Long, Qingyao Ai, Qiaozhi He, Yichen Tang, Changyue Wang, Yiteng Tu, Yingbo Wang, and Yiqun Liu. Skill retrieval augmentation for agentic ai. *arXiv preprint arXiv:2604.24594*, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

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

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. Craft: Customizing llms by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations*, volume 2024, pp. 40097–40125, 2024.

Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems. *Advances in Neural Information Processing Systems*, 38:12988–13018, 2026.

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

Haiteng Zhao, Chang Ma, Guoyin Wang, Jing Su, Lingpeng Kong, Jingjing Xu, Zhi-Hong Deng, and Hongxia Yang. Empowering large language model agents through action learning. *arXiv preprint arXiv:2402.15809*, 2024b.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *International Conference on Learning Representations*, volume 2024, pp. 19036–19066, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A Evaluation Scope and Statistical Reporting

This appendix describes the scope and aggregation of the experimental results. The analysis combines benchmark summaries, category scores, ablations, graph statistics, and efficiency measurements. These summaries answer different questions and need not have identical task populations or aggregation rules. We therefore interpret each metric within its evaluation setting and distinguish descriptive comparisons from statistical significance.

A.1 Evaluation Scope

Item	Evaluation scope and reporting convention
Benchmarks	AppWorld Test-Normal and Test-Challenge, WebArena, ALFWorld, and ScienceWorld. These give five settings from four benchmarks.
Backbones	DeepSeek-V3.2, GPT-4.1-mini, and Qwen3.8-27B.
Deployment	Qwen served locally on an institutional HPC cluster (anonymized for review); DeepSeek-V3.2 and GPT-4.1-mini accessed through API calls.
Methods	No-Memory, Flat-Memory, AWM, ReasoningBank, Mem0, ExpeL, A-MEM, and EvoWeave.
Main metrics	AppWorld TGC and SGC, WebArena SR, and separate scores for ALFWorld and ScienceWorld, all on scales from 0 to 100.
Memory construction	Online over the evaluation stream. Every run starts from an empty memory, uses the released benchmark task order, and is independent across the 15 backbone and setting combinations. No training or development data is used, and episode level success is judged by a language model module shared by all methods rather than by the benchmark scorer.
Aggregation	Matching metrics are averaged equally over backbones. AppWorld Overall TGC weights Normal and Challenge by 168 and 417 tasks, respectively.
Uncertainty	Main results show mean $\pm$ standard deviation. Ablations show point estimates. Cross-method differences are descriptive.
Auxiliary analyses	Category performance, graph composition, token and runtime statistics, and operation overlap are analyzed separately from benchmark means.

Table 4: Evaluation scope and reporting conventions.

The AppWorld ablation combines Normal and Challenge using the two test counts in Table 4. This arithmetic relation does not by itself establish the identity of the evaluated tasks. Experience is acquired online across test tasks under the protocol of Section 4, which is applied identically to every memory based method; Appendix A.3 records it in full.

A.2 Benchmark and Category Aggregation

Benchmark means and category summaries are treated as distinct measurements. For ScienceWorld, weighting category scores by their task counts reproduces the main means to numerical precision. This relation does not hold uniformly across all benchmarks.

For AppWorld, all 48 comparisons between main TGC and the corresponding Supervisor score agree to within 0.014 points, so the two views of the same runs are mutually consistent. The category analysis nevertheless keeps the Supervisor score as a distinct measure rather than treating it as interchangeable with main TGC, because the two are aggregated over different populations.

For WebArena, the main table reports success rate over the full benchmark of 812 tasks (Zhou et al., 2024). The category analysis covers the 764 tasks that belong to a single site, namely Shopping, Shopping Admin, GitLab, Map, and Reddit, whose counts appear in Figure 11. The remaining 48 tasks span more than one site and therefore belong to no single site category. Weighting the five category scores by their task counts consequently yields values 1.23 to 1.69 points above the main means: the category weighted mean omits the cross site tasks, on which every configuration scores lower than on single site tasks. For a fixed task split this difference is proportional to the gap between a method's single site and cross site success rates, and it is smallest for EvoWeave in all three backbones, at most 1.26 points against at least 1.41 points for every baseline. Among the compared methods, EvoWeave therefore has the narrowest gap between tasks confined to one site

and tasks that must be composed across sites. ALFWorld means weighted by category counts differ from the main table by approximately -0.060 to $+0.074$ points. We therefore use category plots to describe variation within categories, without deriving additional benchmark means from them.

AppWorld category memberships in the overlap analysis sum to 183 for Normal and 795 for Challenge, exceeding their respective task totals. Memberships can overlap, so these category counts do not form a partition. Moreover, a global pairwise overlap measure includes tasks from different categories, whereas within category overlap excludes those pairs. The global Jaccard mean is therefore not generally equal to an average of within category Jaccard means, even when weighted by task counts. Figure 4 specifically averages within type overlap with equal weight per type.

A.3 Memory Construction and Leakage Controls

Section 4 states the protocol; this subsection records it in full. Experience is acquired online: a run visits the test tasks of one evaluation setting in sequence, and whatever the reflector writes after a task is available to the tasks that follow it in the same run. The setting is therefore continual learning over an evaluation stream, and it is the setting in which reusable experience is intended to be useful.

Three properties fix what the protocol can and cannot draw on. First, no task is attempted twice, and the agent receives no ground truth for the task it is solving. Several of the compared methods distinguish successful from unsuccessful episodes: ExpeL contrasts the two to distil insights and AWM induces workflows from episodes it treats as successful. That distinction is produced by a single language model judgement module that reads the trajectory, and every method, including EvoWeave, uses the same module with the same prompt. The official evaluator runs only after an episode terminates and its output reaches no method's memory. Stored experience therefore consists of observations the agent obtained by acting and of judgements formed from those observations, not of labels supplied by the benchmark, and a difference between two methods cannot arise from one of them having been given a more reliable success signal. Second, the task order is the order in which the benchmark releases its tasks; it is not chosen by us and it is identical for every method. Third, runs do not share memory, so a result for one backbone or benchmark is not carried by experience gathered in another.

Every memory based method in Table 1 runs under these same conditions: the same empty start, the same task order, the same absence of benchmark labels in memory, and the same per run isolation. The methods therefore differ only in how experience is represented, selected, and injected, which is the comparison the main results are intended to support.

A.4 Uncertainty and Claims

The report of 16 highest means counts metric comparisons; it is not a sample of 21 independent trials or a test of significance. The $\pm$ values denote standard deviations, not standard errors or confidence intervals. Paired significance tests require outcomes on the same tasks and an explicit treatment of repeated seeds and shared task templates. The displayed marginal means and standard deviations alone do not determine confidence intervals for differences, so no statistical significance is claimed.

B Retrieval Procedure and Capability Management

B.1 Bounded Retrieval and Connection Procedure

Algorithm 1 specifies the retrieval flow. The search optimizes the number of historical connector edges locally. Candidate endpoint operations remain part of the returned fragment even when the connector is empty because $v_a = u_b$. A path that exceeds the limit on visible operation cards is returned with an explicit truncation status or omitted as a complete connection.

With at most $k = 3$ candidates per step, each direct plan dependency induces at most $k^2 = 9$ pair searches. If each search expands at most B states, the expansion budget is at most $9|D_t|B$; actual runtime also depends on examined adjacency lists and embedding retrieval. This is a budget bound, not a claim that all retrieved fragments have globally optimal joint cost. For a failed bounded search we record whether the reachable subgraph was exhausted or the search stopped at an unexhausted frontier.

Algorithm 1 Retrieval of operation fragments guided by the plan

Input: Task q, concrete history H_t, graph G, embedding function f, search limits L

$P_t = (S_t, D_t) \leftarrow \text{Planner}(q, H_t)$
for each step $s_i \in S_t$ do
    $C_i \leftarrow$ up to three highest scoring operations matching s_i above the threshold
    Retain edge IDs, endpoints, contracts, references, and versions
end for
$F \leftarrow \varnothing$; initialize candidate assignments for shared steps
for each direct dependency $(i, j) \in D_t$ in a stable order do
    Search $v_a \rightsquigarrow u_b$ for compatible $a \in C_i, b \in C_j$ with bounded BFS
    Rank admissible paths by length, candidate similarity, then stable IDs
    if a consistent connection is available within the budget then
        Add a, its connector, and b to F; record candidate assignments
    else
        Retain separate candidates; record gap, conflict, or search truncation
    end if
end for
Expose P_t, F, unmatched candidates, and all unbound prerequisites to the main agent
return operation cards and statuses, subject to the budget for visible context

Reference parameter	Default value
Operation retrieval cosine threshold / maximum candidates	0.72 / 3
Cosine threshold for merge candidates / maximum candidates	0.82 / 3
BFS maximum depth / expansions	8 / 1,000
Maximum visible operation cards	18
Skill retrieval cosine threshold / maximum cards	0.35 / 8
Code retrieval cosine threshold / maximum cards	0.35 / 4

Table 5: Default parameters of the reference implementation. These specify the retrieval procedure and are distinct from the settings of individual experimental runs.

B.2 Execution Evidence, Capabilities, and Updates

The main agent can use graph operations corresponding to `open`, `neighbors`, `search`, and `path`. The response format specifies direction, input requirements, observed or abstract outputs, versions, and truncation status. Retrieving a card and opening a referenced implementation are separate events. If generated code invokes a registered API or stored function whose definition has not been opened, the execution check first opens that definition and requests regenerated code. An unknown API is not created from a guessed name.

Capability counts describe stored code, skill, and role records. They do not by themselves establish automatic decomposition of arbitrary programs or learning of role prompts. The retrieval formulation assumes stable identifiers for plan steps across rounds so that dependencies and candidate assignments remain consistent.

The reflector generates concrete and abstract summaries in one response, with a further merge decision when needed. Concrete event summaries preserve actual observed values and their sources. Abstract summaries parameterize those values and retain distinctions between contracts. Validation checks placeholders against concrete evidence, verifies capability references and versions, and keeps incompatible effects separate. A memory reference indicates that an operation informed the new action; it is not evidence that a stored code template was executed unchanged. Additional retrieval and merge calls can increase the number of model calls beyond the three persistent roles, and bounded connection search does not guarantee global path optimality.

B.3 Illustrative Dependency Example

Consider a request to split a bill among roommates. Reading the bill produces an amount and currency; retrieving the roommate list produces recipients. Computing a split requires *both* outputs. A

historical path from the operation that reads the bill to the computation does not satisfy the recipient prerequisite. If another fragment supplies the roommate list, the two fragments must still refer to compatible entities and the same current task instance. Finally, requesting payments and sending money have different effects and remain distinct operations. This example illustrates contract checks and AND dependencies; it is not presented as an observed benchmark trajectory.

C Per Backbone Category Performance

Figures 5 to 7 present 15 radar plots, with five settings for each backbone. Every plot includes all eight methods and all categories. A local linear scale maps each category's scores to radial positions, using the same bounds for all methods on that spoke.

Local axis ranges. For a spoke with minimum a and maximum b over the eight methods, let $p = \max(0.15(b-a), 0.5)$. We round $a-p$ downward and $b+p$ upward, using increments of 1 when $b-a \geq 3$ and 0.5 otherwise, and clip the resulting bounds L, U to the valid score range from 0 to 100. The radius is $r = (x-L)/(U-L)$. All methods on that spoke share these bounds, printed as $[L, U]$ beside its label. The center denotes L, the outside circle denotes U, and intermediate rings divide this interval into quarters. Ranges are computed separately for each plotted population. Consequently, compare methods within a spoke; equal radii across spokes or backbones need not represent equal scores, and polygon area is not an aggregate performance measure.

Label key. Files denotes File_System, Notes denotes SimpleNote, Superv. denotes Supervisor, Shop denotes Shopping, and Admin in WebArena denotes Shopping Admin. ALFWorld labels are Pick and Place, Examine in Light, Clean and Place, Heat and Place, Cool and Place, and Use Two Objects. The ScienceWorld mapping is S1: Changes in the state of matter; S2: Temperature and Melting Point; S3: Electricity and Electrical Conductivity; S4: Living and Non-living Things; S5: Plant growth; S6: Chemical mixing; S7: Animal Lifespan; S8: life cycle; S9: Physics Mechanics; S10: Mendelian genetics. Supervisor is displayed as a distinct score, with its relationship to main TGC discussed in Appendix A.

D Ablation Results for Each Backbone

Tables 6 to 8 preserve the complete ablation values. For each configuration and backbone b, the reported Overall TGC is consistent with

$$\text{Overall}_b = \frac{168\,\text{Normal}_b + 417\,\text{Challenge}_b}{585}. \tag{4}$$

The main text table then takes an equal mean over the three backbones. These ablations report point estimates. F is the full system, that is D with active graph retrieval; E is C with active retrieval but without directed historical connections. The highest value in each column is shown in bold. The configurations share the memory construction protocol of Appendix A.3, the same candidate pool, and the same capability library, as described in Section 4.

Table 6: DeepSeek-V3.2 AppWorld ablation.

Configuration	Normal TGC	Challenge TGC	Overall TGC
A: No memory	47.02	23.02	29.91
B: Whole workflow RAG	49.05	23.02	30.50
C: Operation cards	49.09	24.36	31.46
D: C + directed connections	52.75	25.21	33.12
E: C + active retrieval	52.20	25.67	33.29
F: EvoWeave (full)	**54.05**	**27.00**	**34.77**

E Graph Statistics by Evaluation Setting

The macro means in Table 3 weight each setting equally, including Normal and Challenge as separate settings. The mean state and operation counts are 27.0 and 157.6. Mean counts of code, skill,

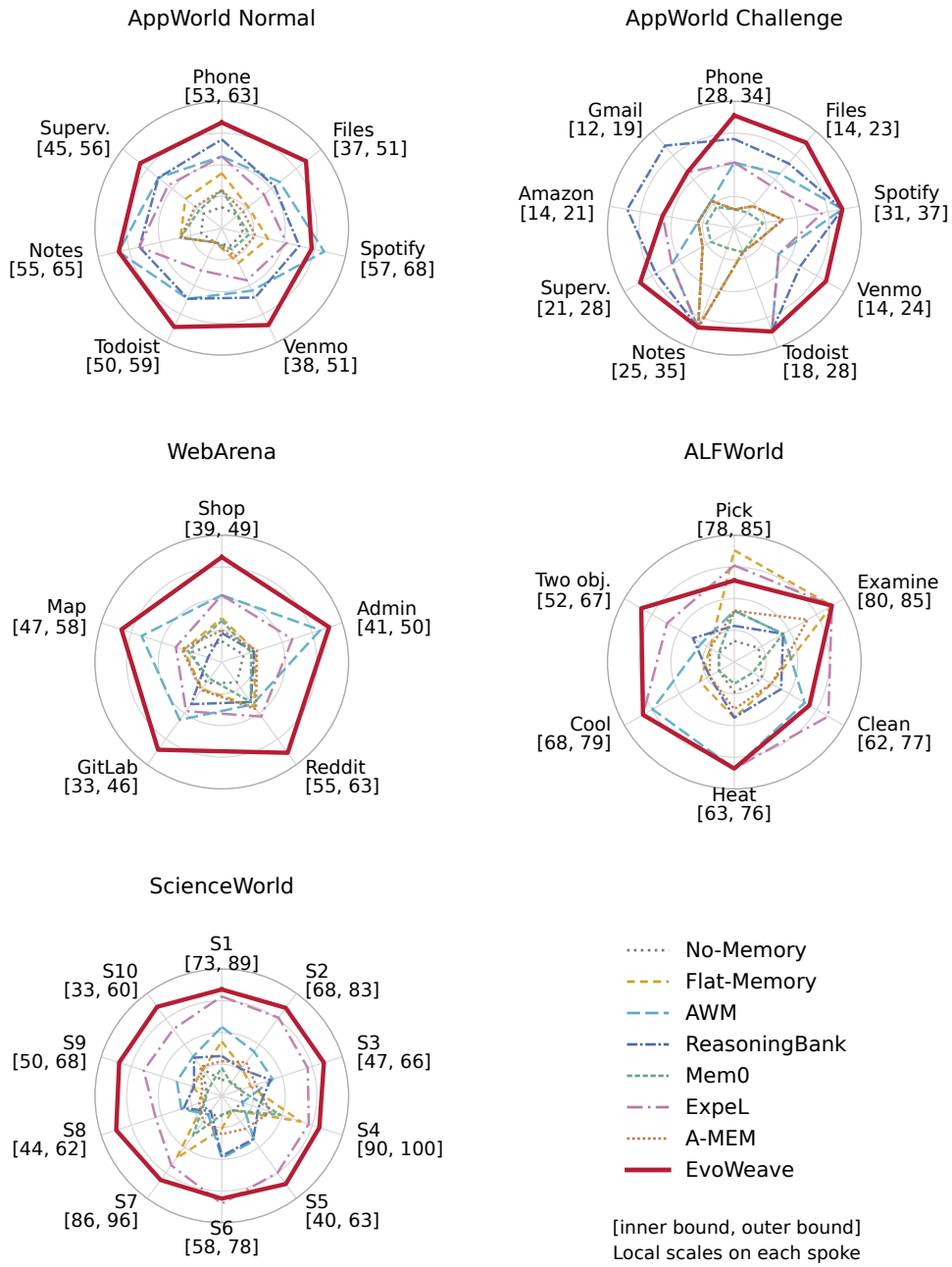

Figure 5: DeepSeek-V3.2 category performance across five settings. Each spoke uses the local range printed beside its category. All eight methods are retained.

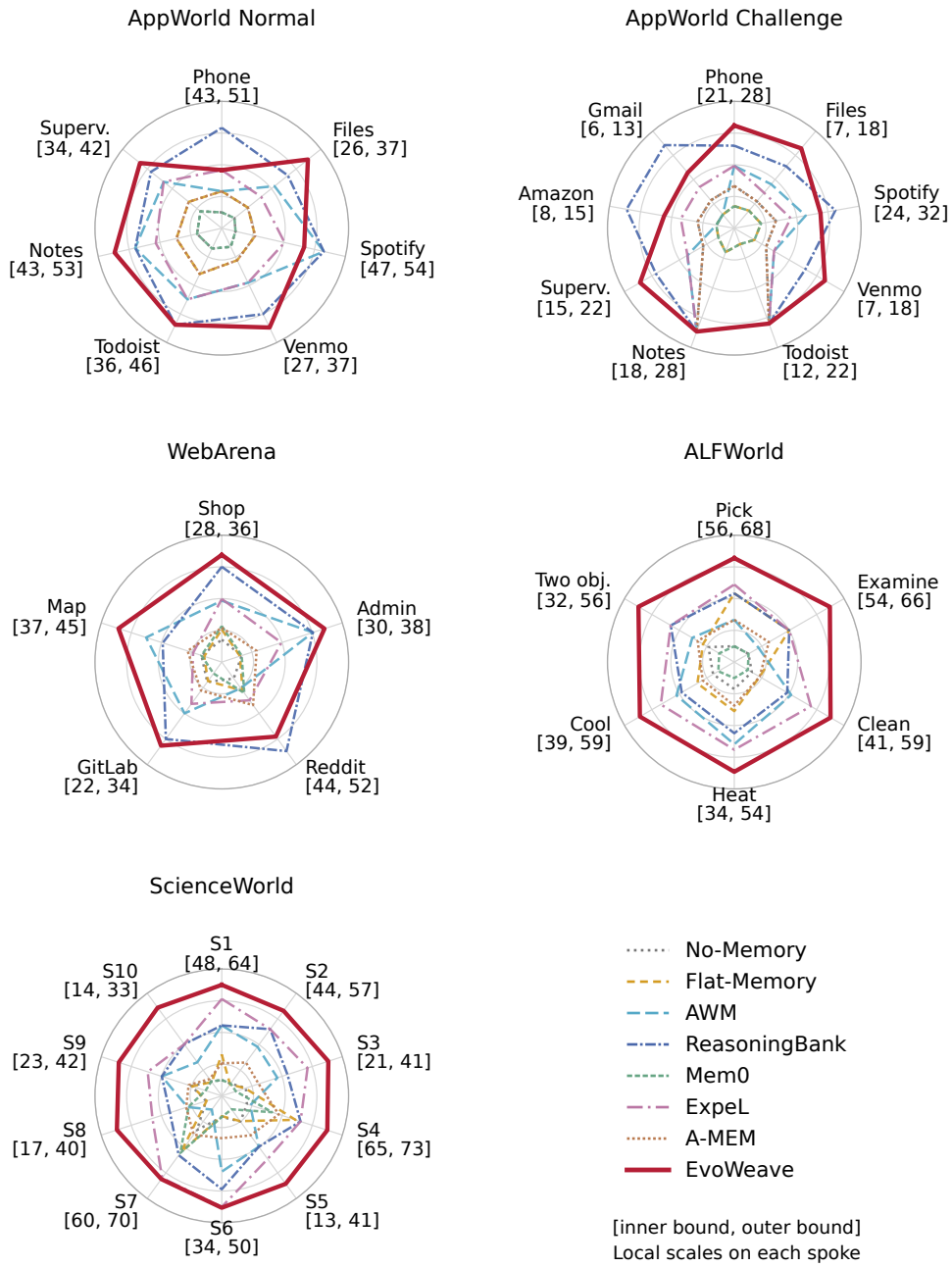

Figure 6: GPT-4.1-mini category performance across the five settings. These curves are inputs from individual backbones to the equal averages over backbones in Figure 3.

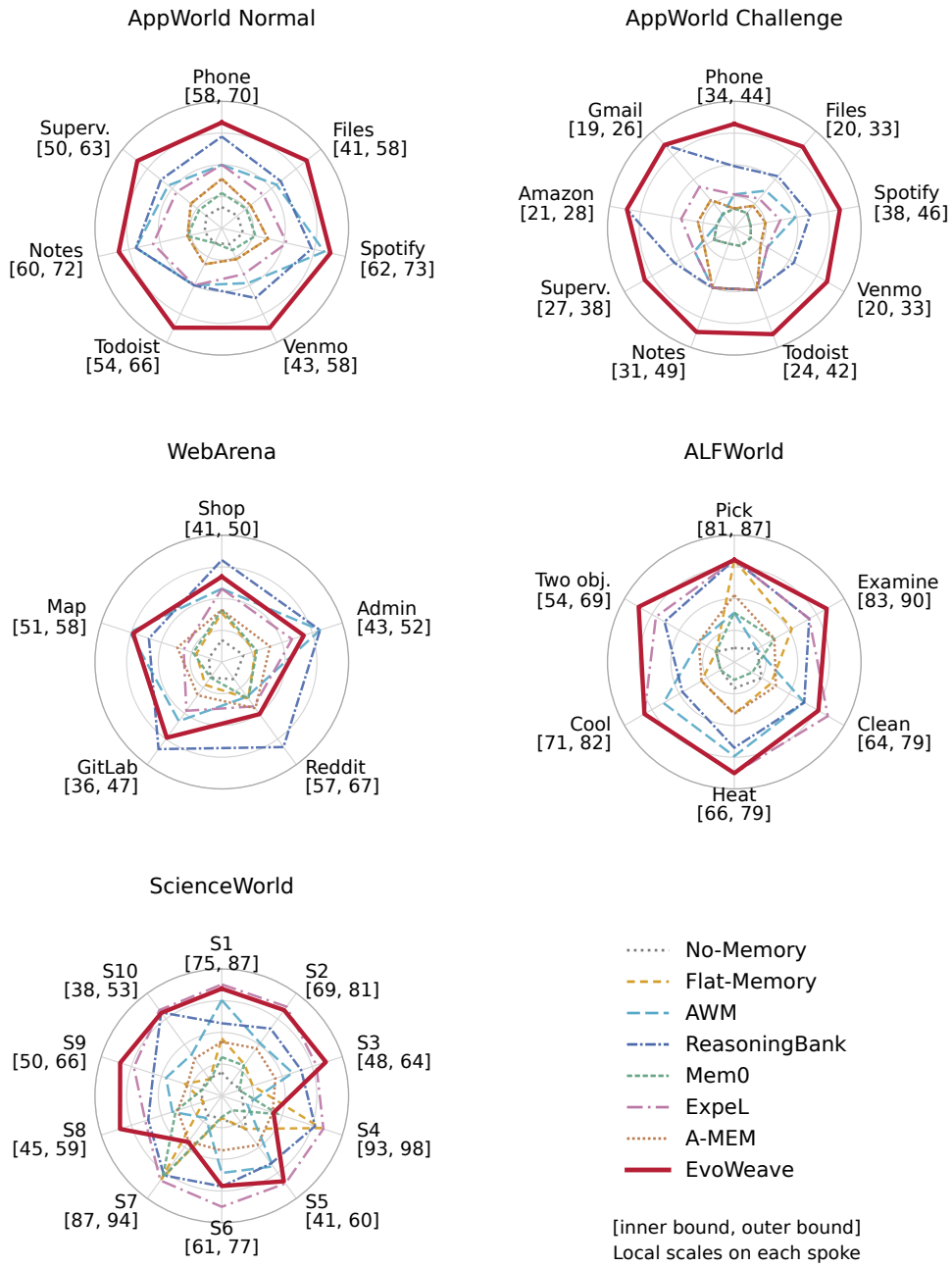

Figure 7: Qwen3.8-27B category performance across the five settings. Each spoke uses the local score range printed beside its category.

Table 7: GPT-4.1-mini AppWorld ablation.

Configuration	Normal TGC	Challenge TGC	Overall TGC
A: No memory	35.71	16.98	22.36
B: Whole workflow RAG	36.67	16.02	21.95
C: Operation cards	38.50	17.81	23.75
D: C + directed connections	39.84	19.80	25.56
E: C + active retrieval	39.11	18.86	24.68
F: EvoWeave (full)	**40.60**	**21.01**	**26.64**

Table 8: Qwen3.8-27B AppWorld ablation.

Configuration	Normal TGC	Challenge TGC	Overall TGC
A: No memory	52.02	29.02	35.63
B: Whole workflow RAG	54.05	30.02	36.92
C: Operation cards	56.20	31.99	38.94
D: C + directed connections	59.63	34.51	41.72
E: C + active retrieval	58.40	33.30	40.51
F: EvoWeave (full)	**61.07**	**35.97**	**43.18**

Table 9: Reported graph statistics for each setting. Ops. denotes directed operation edges. Links denotes references from capabilities to operations, not state transition edges.

	States	Ops.	Code		Skills		Roles	
Setting			Nodes	Links	Nodes	Links	Nodes	Links
AppWorld Normal	37	173	405	173	56	142	9	117
AppWorld Challenge	51	431	467	431	139	825	9	216
WebArena	18	67	17	67	180	562	5	61
ALFWorld	9	14	11	14	6	27	1	8
ScienceWorld	20	103	26	103	30	254	10	95

and role nodes are 185.2, 82.2, and 6.8; the corresponding mean counts of capability links are 157.6, 362.0, and 99.4. These are arithmetic summaries of five graph snapshots. They do not identify average paths, motifs, reachability, degree distributions, or provenance quality. Such analyses require exported adjacency lists and the exact snapshot associated with each run.

Figure 8 visualizes the composition of each snapshot with two horizontal bars. The experience layer combines state nodes and operation edges. The tool layer, termed the capability layer in the method, contains code, skill, and role nodes together with their links to operations. Each bar is normalized separately: a component with count n_k has share $100n_k/\sum_j n_j$, where the sum includes the displayed components of that layer. The labels retain absolute counts so that equal bar lengths do not imply equal graph sizes. These shares describe the composition of recorded graph elements, rather than memory usage or execution frequency.

F Token Usage and Runtime Metrics

For a memory baseline injecting I_b tokens per question, the injection saving is $100(1 - I_{\mathrm{EvoWeave}}/I_b)$. This yields 76.60% for Flat-Memory, 69.05% for AWM, and 48.15% for ReasoningBank. The reduction in total input relative to No-Memory is $100(1 - 102665.46/103690) = 0.99\%$. Injected memory tokens are zero for No-Memory, so a relative injection saving against that baseline is undefined. Lower KV reuse and more seconds per round qualify any interpretation of token savings as a runtime improvement.

Mean total latency cannot be recovered by multiplying average round count by average seconds per round: the two quantities can covary across tasks and exclude different sources of overhead. The summaries also do not establish a common accounting of reflection, validation, active search, retries, and offline construction. API monetary cost and local GPU usage require different accounting.

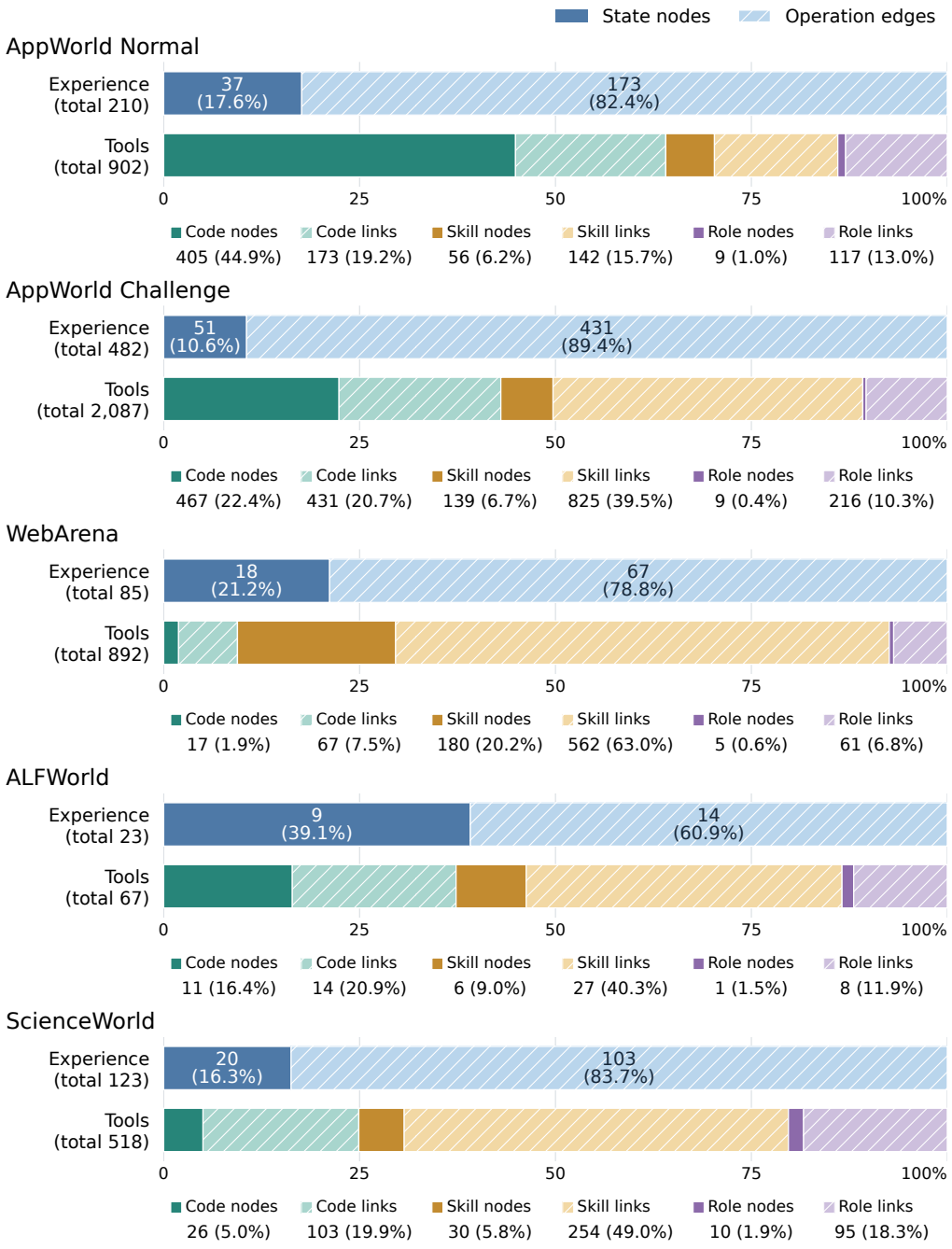

Figure 8: Graph composition for the five evaluation settings. Each panel shows the experience layer and tool layer, each normalized to 100%. Labels report counts and shares within the corresponding layer; totals appear beside the bars. Solid fills indicate nodes and hatched fills indicate operation edges or capability links. The keys below each panel give all six tool components, including those with very small shares. Source counts are retained in Table 9.

Table 10: Efficiency metrics. Injected, Input, and Output denote token counts per question. Rounds and seconds per round are averages. The summaries do not fully specify the model calls included in each token metric.

Method	Injected	Input	KV reuse (%)	Output	Rounds	s/round
No-Memory	0.00	103,690.00	70	4,925.00	18.52	9.52
Flat-Memory	26,300.80	127,710.80	78	4,829.20	17.96	9.79
AWM	19,890.00	114,140.00	78	4,479.80	16.78	9.66
ReasoningBank	11,872.00	103,192.00	77	4,341.00	16.32	9.54
Mem0	8,500.00	113,140.00	76	4,974.00	18.66	9.51
ExpeL	28,011.60	123,961.60	78	4,564.20	17.02	9.82
A-MEM	16,490.00	118,160.00	70	4,839.20	18.04	9.88
EvoWeave	6,155.46	102,665.46	45	4,574.00	15.94	10.44

These metrics therefore characterize context and execution tradeoffs without establishing a ranking by total latency or monetary cost.

G Category Operation Overlap and Referencing

Let O_i denote the operation set associated with task i and $\mathcal{T}_{d,c}$ the tasks of type c in setting d. Within a type, overlap is summarized by the mean pairwise Jaccard similarity

$$J_{d,c} = \frac{100}{\binom{|\mathcal{T}_{d,c}|}{2}} \sum_{i<j,\ i,j\in\mathcal{T}_{d,c}} \frac{|O_i \cap O_j|}{|O_i \cup O_j|}, \tag{5}$$

using benchmark specific operation proxies: gold database model sets for AppWorld, evaluation operators for WebArena, high level actions for ALFWorld, and action verbs for ScienceWorld. These sets omit ordering, arguments, and repeat counts. Overlap therefore describes shared vocabulary, with executable reuse additionally requiring compatible inputs and preconditions.

For Figure 4, we average the type specific percentages with equal weight. With K_d types and reference percentage $R_{d,c}$,

$$\overline{J}_d = \frac{1}{K_d} \sum_{c=1}^{K_d} J_{d,c}, \qquad \overline{R}_d = \frac{1}{K_d} \sum_{c=1}^{K_d} R_{d,c}. \tag{6}$$

The type counts are 9, 9, 5, 6, and 10 for AppWorld Normal, AppWorld Challenge, WebArena, ALFWorld, and ScienceWorld. Overlap is computed within each type, excluding cross type pairs. The averages give equal weight to types, without weighting by task counts or averaging across backbones. Memory reference percentages are analyzed separately from operation overlap. Their event definition and denominator are not sufficiently specified to interpret them as execution success rates.

Figures 9 to 13 show all 39 types and their task counts. AppWorld memberships can overlap. ScienceWorld's auxiliary sample covers 90 tasks from 30 templates with three instances each, grouped into ten topics; complete benchmark coverage is not established. Perfect overlap in small types should be interpreted in light of the size of the underlying vocabulary and of the sample.

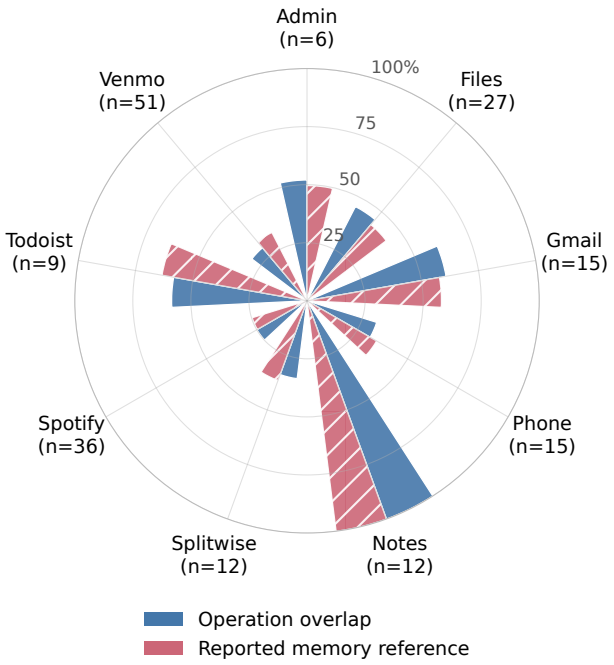

Figure 9: AppWorld Normal: category overlap and reported memory referencing. Categories reflect proxies based on database models; their task counts need not partition the overall task set.

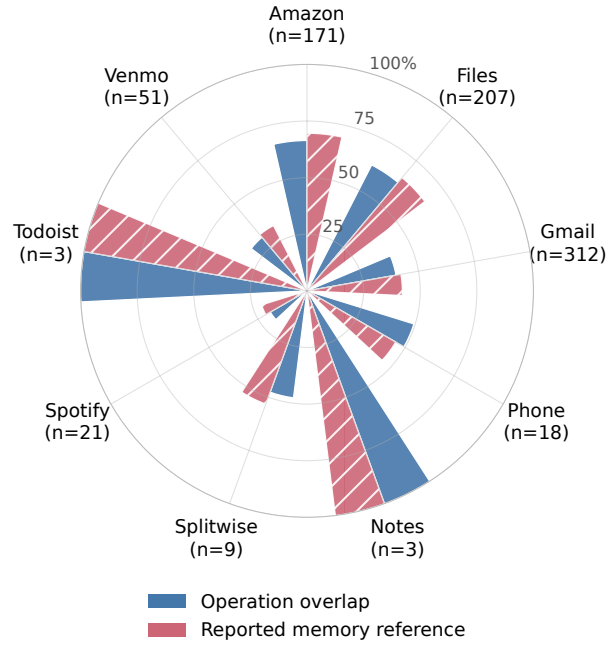

Figure 10: AppWorld Challenge: category percentages, with task counts shown alongside labels. The nine types receive equal weight in Figure 4. Task membership can overlap across categories.

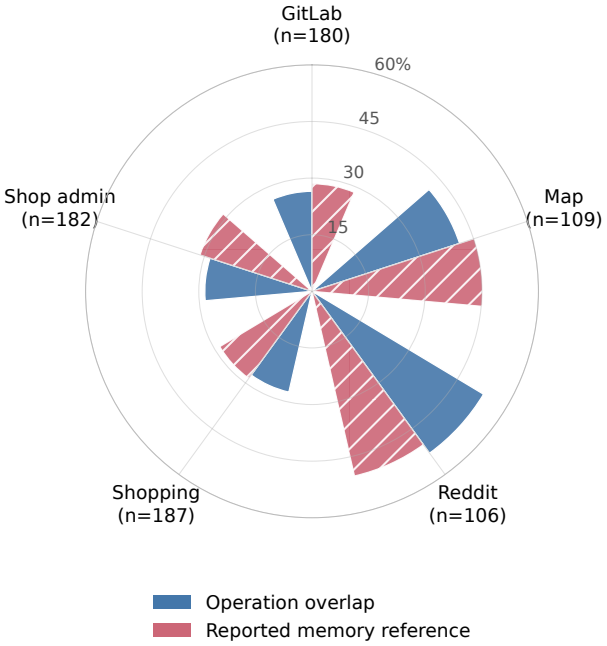

Figure 11: WebArena: overlap of evaluation operators and reported memory referencing. The five single site categories cover 764 of the 812 tasks; the remaining 48 span more than one site and are included in the main table but not in any category, as explained in Appendix A.

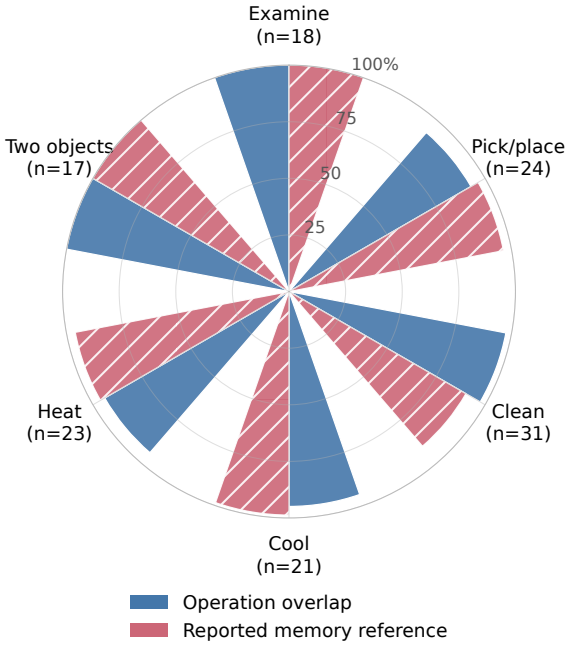

Figure 12: ALFWorld: overlap of high level action sets and reported memory referencing. Repeated action vocabulary does not itself establish matching objects, preconditions, or successful execution.

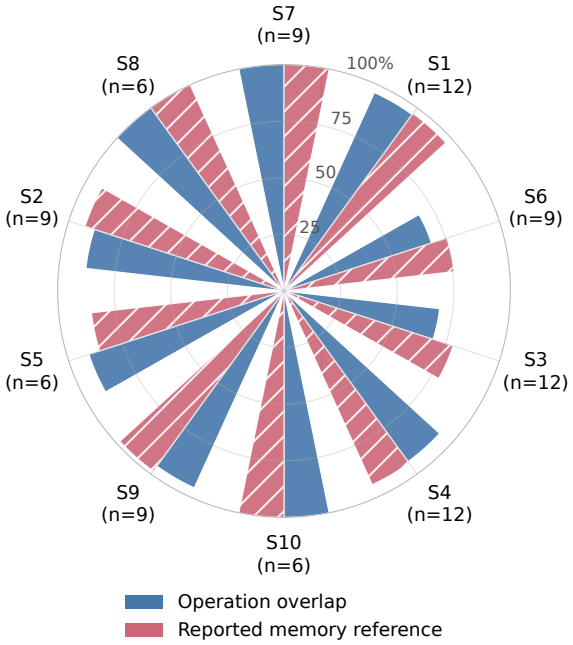

Figure 13: ScienceWorld: overlap of gold verb sets and reported memory referencing across 90 tasks. S1 to S10 follow the topic key in Appendix C.

H Further Evaluation Directions

The existing results motivate, but do not replace, stronger mechanism tests. A token matched comparison of C and D should preserve retrieved operation text and capability references while controlling the number of visible tokens; otherwise, the graph variant can benefit simply from more context. F should receive the same budget for searching descriptions and opening bodies as E, with adjacency and path information as the intended difference. A comparison with fixed capabilities should freeze API definitions and code functions across configurations, isolating changes to experience organization.

A composition test should distinguish no overlap, overlap in one step, partial overlap in several steps, novel combinations of known operations, and requirements with AND joins. The task split must be determined without selecting on final success. Retrieval sensitivity should report both behavior when there are no matches and the effect of thresholds, candidate counts, path depth, and expansion limits. Candidate conflicts, truncated searches, and missing prerequisites should be logged rather than absorbed into a generic failure rate.

A frozen memory condition would complement the online protocol of Appendix A.3 by building memory on a separate split and then evaluating with it fixed, which isolates transfer from continual accumulation. Reporting accuracy as a function of position in the evaluation stream would characterize how quickly useful experience is acquired. Human review of a sample of reused operations could measure whether a reference was relevant, correctly parameterized, and actually adopted. These directions extend the present analyses; they are not additional experimental results.