CONTEXTGRAPH: IMPROVING CODING AGENTS WITH AGENT-SELECTED LESSONS FROM EXPERIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents repeat costly debugging mistakes even when a lesson that would avoid them was learned on an earlier task, often in another repository. We study what makes experience memory help such agents, using ContextGraph, an experience graph that distills successful and failed trajectories into strategies and warnings with the conditions under which they apply, lets the agent select them with its full context before planning and after errors, and adds every completed task back to memory. ContextGraph improves over no memory by 7.6, 13.7, and 11.3 points on SWE-bench Verified, SWE-ContextBench Related-Lite, and Datacurve DeepSWE, reaching 69.2%, 35.1%, and 72.1%, the highest rate among the evaluated memory methods on each. Its ablations show that two choices carry the gain: memory should store abstract lessons and let the agent select them for each task. On Related-Lite, raw trajectory excerpts selected by the agent and a fixed set of abstract lessons given to every task both stay at the no-memory level (20.5% and 20.8% versus 21.4%), and abstract lessons retrieved by embedding similarity leave ContextGraph only 2.9 of its 13.7 points. Updating memory after every task adds further gains, and lessons from other repositories alone carry more than half of the gain on Related-Lite and DeepSWE. The gains hold for strong agents: with Gemini 3.1 Pro, ContextGraph raises SWE-bench Verified pass@1 from 82.3% to 88.9%, above the 82.8% that five attempts reach without memory. We release the code and prompts so that others can build on these findings.

1 INTRODUCTION

Each repair a coding agent attempts produces experience: where the fault was, which reproduction exposed it, and which approach wasted effort. Memory lets later tasks reuse this experience, so the agent improves from its own work without changing its weights or code, unlike agents that rewrite themselves (Yin et al., 2025; Zhang et al., 2026a). Memory methods differ in what they store (trajectories, insights, or playbook entries), how a task receives it (similarity top-k or the whole playbook), and whether memory changes during use (Shinn et al., 2023; Zhao et al., 2024; Ouyang et al., 2026; Zhang et al., 2026b) (Table 1). These choices matter: under a shared agent, backbone, and budget, six memory systems change resolution by between -2.1 and $+9.9$ points over no memory (Section 4.2). We ask which choices make experience memory work for coding agents; Figure 1 previews the answer.

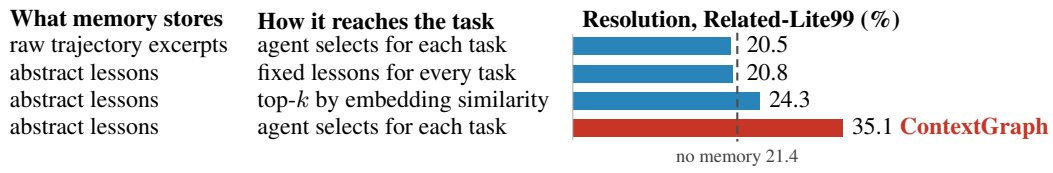

Figure 1: Experience memory helps a coding agent most when it stores *abstract lessons* and the *agent selects them for each task*. Removing either from ContextGraph leaves at most 2.9 of its 13.7 points. Related-Lite99, DeepSeek V4 Pro 0813 with mini-SWE-agent, five-run means; all other components unchanged, except that the fixed-lessons row has no memory access at all (Table 3).

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 2 shows what a useful memory holds. In Django Trac #33018, empty query results are mishandled under negation. A failed `python-statemachine` attempt had spent its effort setting up a full Django application, and a successful `sievelib` trajectory supported reproducing the failure before editing. From these two attempts, the agent selected a strategy (reproduce the failure in a minimal script), a warning (do not set up the full framework first), and the condition under which both apply (the failure appears without the full application). Guided by them, it traced the fault to `Exists.as_sql()`, corrected its handling, and added a regression test; the no-memory attempt made broad formatting changes without fixing the bug. Neither source repository is Django: the decision and its condition transferred, and the agent wrote the local fix.

We answer this question with **ContextGraph** (Section 3), whose five design choices can each be switched off and measured. It distills successful and failed trajectories into strategies and warnings with the conditions under which they apply, links them to their source experience and the errors they address, lets the agent select lessons with its full context before planning and again after errors, and distills every completed task back into memory. Across four benchmarks, five backbones, and three agent scaffolds, ContextGraph improves repair. With DeepSeek V4 Pro it improves over no memory by 7.6, 13.7, and 11.3 points on SWE-bench Verified, SWE-ContextBench Related-Lite, and Datacurve DeepSWE, the largest gain among the evaluated memory methods on each; with Gemini 3.1 Pro it raises Verified pass@1 from 82.3% to 88.9%; and with GPT-5.5 in Codex it solves 3 of 20 long-horizon LoLBench Python tasks at pass@1, where no memory and six memory baselines solve 1. Its ablations and analyses yield five takeaways for building coding-agent memory:

What works for coding-agent memory. Takeaways 1 and 2 carry the gain: without either, ContextGraph keeps at most 2.9 of its 13.7 points (Figure 1). Takeaways 3–5 show what else the gain draws on and where it holds.

1. **Abstract experience into lessons; do not replay trajectories.** Raw trajectory excerpts stay at the no-memory level even when the agent selects them for each task (Section 4.3).
2. **Match lessons to each task, and let the agent choose them.** A fixed playbook on every task gives no gain; agent-directed selection beats similarity top-k by 10.8 points and random summaries from the same pool by 8.0 (Section 4.4).
3. **Update memory after every task.** Online updates add 6.4 points over a static pool, and their lead grows along the task stream and across retries (Section 4.5).
4. **Share experience across repositories, and collect enough of it.** Lessons from other repositories alone carry more than half of the gain on two benchmarks; 50 initial summaries give no gain, 300 give the full 13.7 points (Section 4.6).
5. **Strong agents benefit too.** With Gemini 3.1 Pro, memory raises Verified pass@1 from 82.3% to 88.9%, removing 37% of the remaining failures; with GPT-5.5 in Codex, ContextGraph solves 3 of 20 LoLBench Python tasks at pass@1, where every other method solves 1 (Section 4.7).

These findings extend the abstraction and pool-size effects of Memory Transfer Learning (Kim et al., 2026) to repository-level repair. We contribute ContextGraph, an experience graph that brings condition-structured lessons (Majumder et al., 2023; Fu et al., 2024) to coding agents and integrates memory into planning, error recovery, and learning after every task; the five takeaways, each backed by controlled ablations or analyses in Section 4; and the code, prompts, and graph schema, released so that others can build on them.

2 Related Work

Abstracting experience for coding agents. ExpeL and ReasoningBank extract trajectory lessons (Zhao et al., 2024; Ouyang et al., 2026). SWE-Exp and ExpeRepair reuse repair experience, and STAIR abstracts past repairs into hierarchical plans (Chen et al., 2025; Mu et al., 2026; Xu et al., 2026). MemGovern distills human repair histories into experience cards that code agents search and browse (Wang et al., 2026), and Agent KB serves planning guidance and diagnostic feedback from a cross-framework experience base (Tang et al., 2025). Memory Transfer Learning finds that abstraction determines how well coding memories transfer across domains and that transfer grows with the memory pool (Kim et al., 2026); we observe both effects in repository-level repair (Sections 4.3 and 4.6). CLIN stores causal abstractions of the form "X may be necessary for Y" (Majumder et al., 2023), AutoGuide pairs guidelines with the context in which they apply (Fu et al., 2024), and Agent Workflow Memory induces reusable workflows (Wang et al., 2025b), all for web navigation and simulated environments. ContextGraph brings condition-structured lessons to repository-level

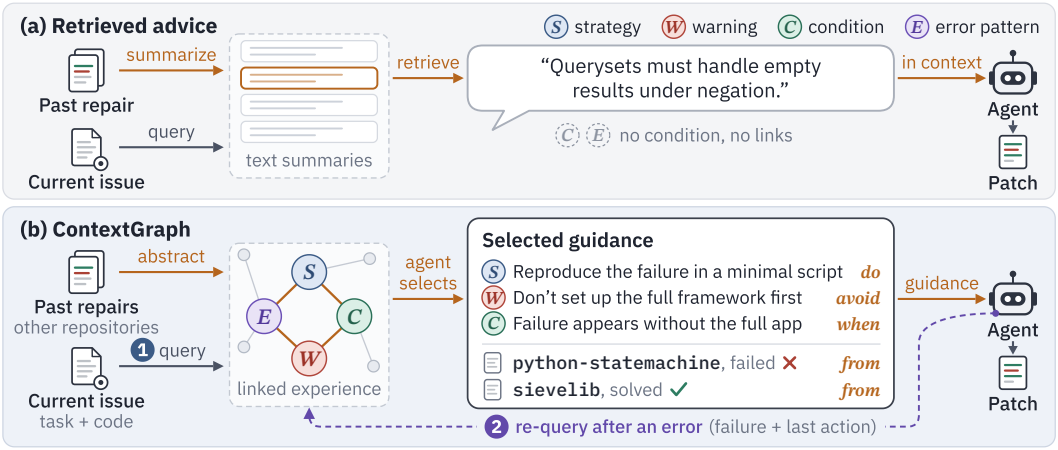

Figure 2: Retrieved advice versus ContextGraph on Django Trac #33018 (illustration). (a) A flat summary store returns one requirement sentence without condition or source. (b) The agent selects a strategy, a warning, and the condition under which they apply, each linked to the failed `python-statemachine` or the successful `sievelib` attempt, before planning and again after an error.

Table 1: Design space of coding-agent memory; every method except ExpGraph is compared in Section 4. Organization describes how stored experience is connected; selection describes how lessons are chosen for a task. ✓: supported by the method's description and open-source implementation; ○: partial or not enabled in the default configuration used in our comparison; ×: absent. The last row gives the change in ContextGraph's Related-Lite99 resolution when it gives up each choice (Table 3); the static playbook also has no memory access at all.

Method	Memory unit / organization	Selection	After errors	Keeps failures	Online updates
ExpeL	Insights and trajectories / flat	Top-k similarity	×	✓	×
ReasoningBank	Reasoning strategies / flat	Top-k similarity	×	✓	✓
ACE	Playbook entries / sectioned playbook	Whole playbook in context	×	✓	✓
ExpeRepair	Episodes and rules / stage-indexed	Top-k per stage	✓	✓	✓
ExpGraph	Experience nodes / graph	PageRank + learned controller	×	✓	✓
Supermemory	Facts and relations / relational store	Hybrid search	×	○	✓
ContextGraph	Strategies and warnings / typed graph	Agent-directed search	✓	✓	✓
ContextGraph without it (pp)	−14.6 raw trajectories; −4.0 flat pool	−10.8 top-k; −14.3 static playbook	−1.9	—	−6.4

repair and links lessons to their source trajectories and the errors they address. RETester generates tests from bug reports, MIRA verifies repair memories against evidence, and TRACE turns corrections into runtime checks (Wang et al., 2025a; Liu et al., 2026; Zhou et al., 2026); in ContextGraph, the agent applies lessons through its own reproductions and tests.

Retrieving related experience. A-MEM links notes, Zep organizes temporal facts, and HippoRAG retrieves through graph traversal (Xu et al., 2025; Rasmussen et al., 2025; Gutiérrez et al., 2024). ExpGraph combines experience graphs, personalized PageRank, utility ranking, and a trained retrieval controller (Feng et al., 2026), and Supermemory combines relational updates and hybrid retrieval (Daga et al., 2026). SWE-ContextBench evaluates memory-assisted repair and finds that a supplied summary of the related history helps more than the supplied history itself (Zhu et al., 2026). ContextGraph lets the agent search its graph with its full context and select lessons; MemGovern also pairs abstracted experience with agent search, and our ablations measure how much such a pairing matters. Table 1 places these methods in the design space that Section 4 measures.

Updating memory and improving agents. MemGPT and SWE-MeM manage memory access and coding histories (Packer et al., 2023; Gao et al., 2026); ACE evolves context, MemOp optimizes memory utility, and AdaRepair-Mem selects experience by repair stage (Zhang et al., 2026b; Guo et al., 2026; Luo et al., 2026). Gödel Agent and the Darwin Gödel Machine modify agent code, RRSI

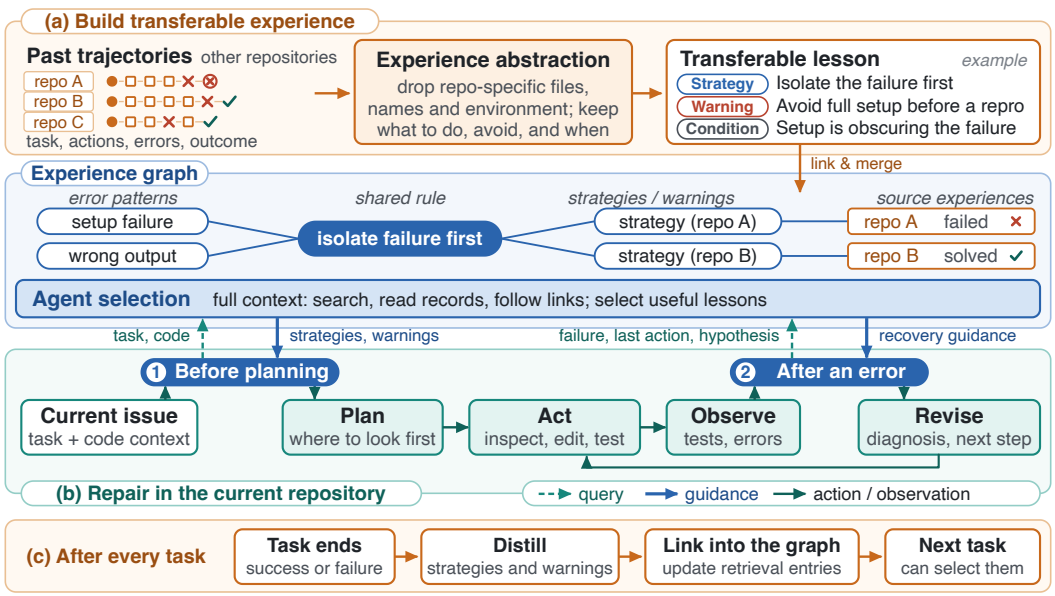

Figure 3: ContextGraph. (a) Past trajectories, including those from other repositories, are abstracted into strategies and warnings with their conditions and linked to their sources and error patterns. (b) The agent selects lessons with its full context before planning and again after an error. (c) Every completed task, successful or failed, is distilled into the graph for the tasks that follow.

regularizes harness changes, and AlphaEvolve evolves programs through executable evaluation (Yin et al., 2025; Zhang et al., 2026a; Xia et al., 2026; Novikov et al., 2025). ContextGraph improves the agent without changing its parameters or code: it adds lessons from successes and failures after each task and makes them immediately available.

3 ContextGraph: Memory for Planning, Recovery, and Learning

Setting. Tasks $q_1, \ldots, q_T$ arrive as a stream in issue creation-time order; each gives the agent an issue and a repository, and the agent produces a patch. Its trajectory τ_t records actions, observations, and the outcome. Memory starts from a graph G_0 built from trajectories $\mathcal{D}_0$ of other tasks. While solving q_t, the agent selects lessons from G_{t-1}; afterward, ContextGraph distills τ_t into $G_t = \mathrm{Update}(G_{t-1}, \tau_t)$ (Figure 3). $\mathcal{D}_0$ excludes the evaluated tasks, updates use only agent-visible observations, and lessons may come from any repository.

3.1 Abstract Experience into Conditioned Lessons

A language model extracts lessons from the task, actions, errors, and outcome. Each lesson is a record

$$\ell = (\kappa,\ c,\ a,\ o,\ s), \qquad \kappa \in \{\text{strategy}, \text{warning}\}, \tag{1}$$

read as "when c holds, do a (for a warning, avoid a) to achieve or preserve o," where s links the lesson to its source trajectory. For the initial graph, abstraction removes repository-specific names while retaining the decision and its condition: the failed framework setup of Figure 2 becomes the warning not to set up the full framework first when the failure appears without the full application. Lessons distilled online keep the files and commands the agent used, so later tasks in the same repository can reuse them.

Lessons live in a typed graph

$$G = (V, E), \qquad E \subseteq V \times \mathcal{R} \times V. \tag{2}$$

Besides lessons (strategies and warnings), V holds trajectories, fragments (action–observation segments), canonical rules (merged lessons), error patterns, problem summaries, playbook entries

Algorithm 1 ContextGraph across a stream of coding tasks

Input: initial trajectories $\mathcal{D}_0$, coding tasks $q_1, \ldots, q_T$.
Output: candidate patches and the updated experience graph G_T.
1. Extract strategies and warnings from $\mathcal{D}_0$; build G_0 and its retrieval entries.
2. **For each** task q_t in issue creation-time order:
   a. Use the full task context $C_{t,i}$ to search G_{t-1} and select lessons $\mathcal{M}_{t,i}$ for planning.
   b. Execute investigation, editing, and testing actions; record the trajectory τ_t.
   c. After an error, search again with the updated full context; revise the plan.
   d. Continue the repair loop until the task ends or its budget is exhausted.
   e. Distill experience from the trajectory and agent-visible outcomes.
   f. Set $G_t = \mathrm{Update}(G_{t-1}, \tau_t)$ and refresh retrieval entries; expose the new experience to later tasks.

(retrieval-ready lessons), and communities of related fragments. Seven relation types in $\mathcal{R}$ link lessons to their source trajectories and canonical rules, rules to the errors they address, trajectories to their fragments and problem summaries, and fragments to the errors they caused and to their communities. These links keep both a reusable lesson and the observations that explain when it applies. Appendix Table 13 lists the node and edge counts of an illustrative graph built from Nebius SWE-agent trajectories. The anonymous repository (Section 4.1) releases the extraction prompts, graph schema, and selection code.

3.2 Let the agent select lessons for each task

At step i of task q_t, the agent holds its full context $C_{t,i}$: the issue, inspected code, actions, and failures. Selection is a short decision process run by the coding agent's own policy π_θ:

$$m_j \sim \pi_\theta(\cdot \mid C_{t,i}, h_j), \qquad m_j \in \{\textsc{search}(x),\ \textsc{read}(v),\ \textsc{follow}(v, r),\ \textsc{keep}(\ell),\ \textsc{stop}\}, \quad (3)$$

where h_j records the preceding memory actions and their results. Searches with a query x return ranked candidates from all of G_{t-1}; the agent reads source records, follows links of type $r \in \mathcal{R}$, and keeps a lesson ℓ when its condition c matches the current task and error. At STOP, the kept lessons form $\mathcal{M}_{t,i}$, so the number of selected lessons varies by task. The agent applies them through local reproductions, edits, and tests (Appendix F).

3.3 Query before planning and after errors; update after every task

Before planning, the agent searches with the full task context for investigation, reproduction, and testing strategies, and for warnings about costly approaches to avoid. **After an error,** it adds the failed action, observed error, relevant code, and current hypothesis to that context, searches again to revise its diagnosis, and uses repository tools to try the next step. **After each task,** ContextGraph distills the successful or failed trajectory and links new lessons to existing experience, and later tasks can select them immediately (Algorithm 1). A task late in the stream therefore selects from a larger graph than an early one, and a retry of an unresolved task can select the lessons distilled from its own earlier failure.

4 What works for coding-agent memory

4.1 Setup

Tasks and metrics. We evaluate all 500 SWE-bench Verified tasks (Verified500), the 99-task Related-Lite subset of SWE-ContextBench (Related-Lite99), and the 113-task Datacurve DeepSWE benchmark (DeepSWE113) (Jimenez et al., 2024; Zhu et al., 2026; Huang et al., 2026) with DeepSeek V4 Pro 0813 and mini-SWE-agent (SWE-agent Team, 2025; Yang et al., 2024), and 20 Python tasks from LoLBench (Peng, 2026) with GPT-5.5 and Codex. Their initial memory comes from Nebius SWE-bench-extra trajectories, SWE-bench cold-start trajectories, LoLBench repository trajectories, and other LoLBench Python tasks, respectively (Table 2). A task is resolved when its patch passes the benchmark's official evaluation. Resolution is the percentage of tasks resolved over completed evaluations; pass@k reports success within k attempts; Appendix B gives confidence intervals. Related-Lite99 and DeepSWE113 in the main comparison report means over five

runs, each reset to the same initial memory pool. All methods share the agent, backbone, and a per-task budget of 500 steps and $20 on Verified500 and Related-Lite99, $50 on DeepSWE113, and $100 on LoLBench; ContextGraph distills lessons with Claude 4 Sonnet.

Table 2: Benchmarks and initial memory. The first three use DeepSeek V4 Pro 0813 with mini-SWE-agent; LoLBench uses GPT-5.5 with Codex. ContextGraph updates its memory after every task; baseline updates follow each implementation's default behavior. No initial pool contains an evaluated task (Appendix C).

Benchmark	Tasks	Initial experience source	Reported result
Verified500	500	Nebius SWE-bench-extra trajectories	Resolution
Related-Lite99	99	SWE-bench cold-start trajectories	Five-run mean
DeepSWE113	113	LoLBench repository trajectories	Five-run mean
LoLBench Python	20	Other LoLBench Python tasks	pass@1, pass@3

Data separation and feedback. Initial memory excludes the evaluated tasks' trajectories and reference solutions; one instance-level and patch-similarity audit is applied to every pool of the main comparison (Appendix C). No pool contains an evaluated task, and removing entries whose patch has token-Jaccard similarity of at least 0.85 to an evaluated task's reference patch takes out between 3 and 18 entries per pool, at most 1.3%. Tasks enter the evaluation stream in issue creation-time order, and ContextGraph's memory grows task by task on every benchmark, including Related-Lite99, whose original protocol keeps its pool fixed (Zhu et al., 2026). Memory updates use only agent-visible actions and observations, including the agent's own test runs. Official test results are never shown to the agent or written into memory; beyond scoring, they only decide which tasks the retry experiment of Section 4.5 attempts again. Code, extraction prompts, and the graph schema are released in an anonymous repository.[1]

Baselines. We compare no memory, FAISS Flat, ExpeL, ExpeRepair, ReasoningBank, ACE, and Supermemory (Zhao et al., 2024; Mu et al., 2026; Ouyang et al., 2026; Zhang et al., 2026b; Daga et al., 2026). FAISS Flat uses vector similarity to retrieve memory without graph traversal. We adapt ExpeRepair and ACE to mini-SWE-agent, retaining their memory procedures. Each baseline is a complete memory system with its own construction, selection, and updates; the ablations that follow isolate individual choices. Appendix Table 7 lists all values.

4.2 ContextGraph improves repair on every benchmark

Figure 4 shows each memory method's gain over no memory. ContextGraph improves by 7.6, 13.7, and 11.3 points on Verified500, Related-Lite99, and DeepSWE113, reaching 69.2%, 35.1%, and 72.1%, the highest rate on each. Averaged over the three benchmarks, its gain of 10.9 points is more than twice that of the strongest baseline, ExpeRepair (5.2), while ExpeL falls below no memory on two of the three. On Verified500 and Related-Lite99 the gain exceeds its conservative 95% binomial half-width (7.6 against 5.9, and 13.7 against 12.4; Appendix B). The following subsections give one takeaway each.

4.3 Abstract experience into lessons; do not replay trajectories

Table 3 changes one component at a time on Related-Lite99 under the main protocol and keeps the others unchanged, except in the static playbook, which has no memory access at all. *Raw trajectories* skips abstraction and stores trajectory excerpts instead of strategies and warnings; the agent still selects them with its full context. This erases the entire gain: 20.5% against 21.4% without memory and 35.1% for the full system (Figure 1); this drop and the static playbook's both exceed their conservative 95% half-widths (Appendix Table 8). SWE-ContextBench shows the same contrast for supplied history with Claude Sonnet 4.5: supplying the known related trajectory adds 1.01 points, whereas supplying a summary of it adds 8.08 (Appendix Table 16). Most lessons the agent selects come from other repositories (Section 4.6). An excerpt from another repository carries that repository's code; an abstract lesson carries the decision and the condition under which

[1] https://anonymous.4open.science/r/contextgraph-kit/

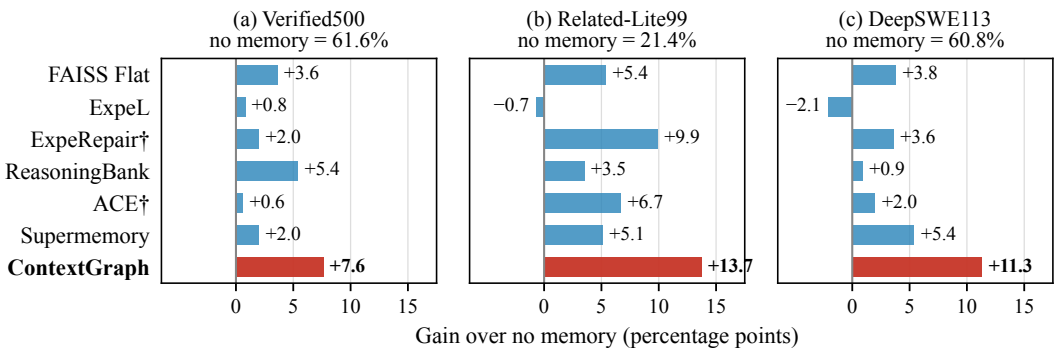

Figure 4: Gain in resolution over no memory (percentage points) with DeepSeek V4 Pro 0813 and mini-SWE-agent; panel titles give the no-memory rate. Related-Lite99 and DeepSWE113 are five-run means. ContextGraph (red) has the largest gain on every benchmark; Appendix Table 7 gives absolute rates. † marks our adaptations to mini-SWE-agent.

it applies, and the agent writes the local change. Raw trajectories drop the condition together with the rest of the lesson format, and none of our ablations isolates the condition's own contribution (see Limitations).

Table 3: Ablation on Related-Lite99 with DeepSeek V4 Pro 0813 and mini-SWE-agent, five-run means. Each variant keeps the remaining components unchanged, except the static playbook, which has no memory access at all; Δ is the change from the full system in percentage points.

Variant	Resolution (%)	Δ
ContextGraph (full)	**35.1**	—
Falls to the no-memory level		
Raw trajectories (no abstraction)	20.5	−14.6
Static playbook (fixed lessons, no memory access)	20.8	−14.3
Keeps part of the gain		
Fixed top-k retrieval (no agent-directed selection)	24.3	−10.8
Static pool (no online updates)	28.7	−6.4
No graph links (flat lesson pool)	31.1	−4.0
Planning-time search only (no search after errors)	33.2	−1.9
No memory	21.4	−13.7

4.4 Match lessons to each task, and let the agent choose them

Abstract lessons still have to reach the right task. The *static playbook* places the lessons the agent selects most often in the system prompt of every task, with no memory access at all, and stays at the no-memory level (20.8%): even these lessons do not help when every task receives the same fixed set. ACE, which also gives every task its whole playbook but updates it online, reaches 28.1%. With the same abstract lessons in memory, *fixed top-k*, which retrieves the k lessons most similar by embedding, reaches 24.3%, while agent-directed selection reaches 35.1%; over a flat pool without graph links it still reaches 31.1%. ReasoningBank, which also retrieves abstract strategies by top-k similarity, reaches a similar 24.9%, and FAISS Flat, which retrieves by vector similarity, 26.8%. With DeepSeek V4 Pro, ContextGraph exceeds random summaries from the same 300-summary pool by 8.0 points (35.1% against 27.1%), and with all three backbones of Figure 5 it exceeds an oracle that is handed one known related summary (35.1%, 28.6%, and 27.6% against 27.8%, 24.5%, and 15.8%).

The choices work together. Removing abstraction, agent-directed selection, online updates, graph links, or search after errors one at a time costs 14.6, 10.8, 6.4, 4.0, and 1.9 points, more in total than the 13.7-point gain, so the choices build on each other rather than adding up. Abstrac-

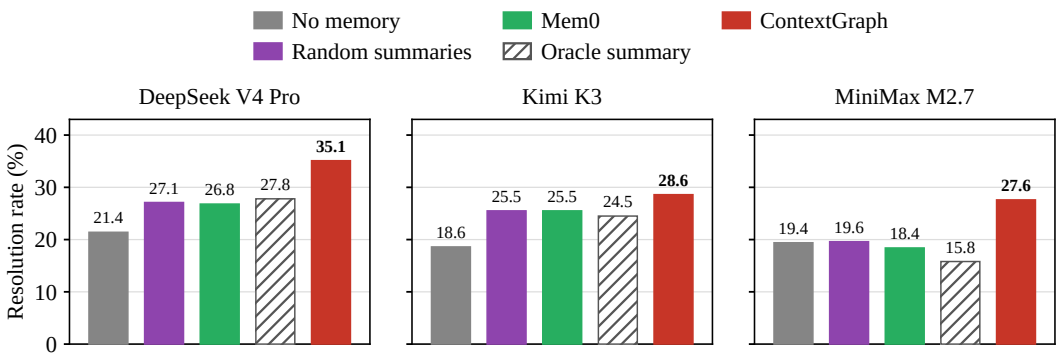

Figure 5: ContextGraph (red) has the highest Related-Lite99 resolution (%) with every backbone and exceeds the oracle (hatched), which is handed one known related summary. All methods share the same 300-summary initial pool per backbone. The DeepSeek V4 Pro ContextGraph and no-memory bars are the main-comparison values and the other DeepSeek V4 Pro bars follow the same protocol; Kimi K3 and MiniMax M2.7 are single runs over completed evaluations.

tion and agent-directed selection carry the gain: without abstraction, the other four choices together add nothing (20.5%); without agent-directed selection, they add 2.9 points (24.3%).

4.5 Update memory after every task

Online updates add 6.4 points (Table 3); even without them, ContextGraph (28.7%) stays 8.0 points above ExpeL (20.7%), which also keeps a static pool. A separate retry study on Related-Lite99 shows how the benefit of updating builds up (Table 4). It uses DeepSeek V4 Pro with the main-comparison budget, a curated pool of 291 repair records of other tasks with their reference fixes and verification hints, and three settings, all run within the study: no memory, a static pool, and online updates. Each later pass retries the tasks that remain unresolved. Online updates and the static pool start from the same pool, so their difference isolates what the agent learns during the run. They match in the first third of the stream (5.4 versus 5.6 of 33 tasks); the lead of online updates then grows to 1.0 and 3.0 tasks, while no memory stays flat (4.6, 4.8, 4.8). Retries compound the benefit, and with online updates a retry can draw on lessons distilled during the run, including from the task's own failed attempt: two retry passes add 5.4 tasks with online updates, 2.8 with the static pool, and 2.0 without memory, and one pass with online updates (21.2 tasks) already resolves more than three passes with the static pool (20.2). Sequential attempts behave the same way with Gemini 3.1 Pro on Verified500: with memory updated after each attempt, pass@5 adds 2.8 points over pass@1, against 0.5 without memory (Table 6). One pass with the same pool also improves over no memory with DeepSeek V3.2, GLM 5.2, and Kimi K3 (Appendix Table 10).

Table 4: Related-Lite99 retry experiment (curated episodic memory, DeepSeek V4 Pro, 500 steps and $20 per task), means over five runs in issue creation-time order. Left: tasks resolved in the first pass within each third of the stream (of 33). Right: cumulative tasks resolved (of 99) after one, two, and three passes. In one run, the first-pass gain also holds on the 89 tasks without near-duplicate source fixes (Appendix D).

Setting	Tasks 1–33	Tasks 34–66	Tasks 67–99	pass@1	pass@2	pass@3
No memory	4.6	4.8	4.8	14.2	15.8	16.2
ContextGraph, static pool	5.6	6.2	5.6	17.4	19.2	20.2
ContextGraph, online updates	5.4	7.2	**8.6**	**21.2**	**24.8**	**26.6**

4.6 Share experience across repositories, and collect enough of it

We log the source repository of every lesson selected in the main comparison (Table 5): 78.4%, 86.2%, and 64.1% come from another repository on Verified500, Related-Lite99, and DeepSWE113.

With same-repository lessons excluded at selection time, ContextGraph still improves over no memory by 1.4, 7.3, and 6.7 points, more than half of the full gain on Related-Lite99 and DeepSWE113. The repository's own lessons add the rest (6.2, 6.4, and 4.6 points), which on Verified500 is most of the gain. The DeepSWE113 initial pool contains no DeepSWE113 repository (Appendix C), so its same-repository lessons were all learned online from earlier tasks in the stream: updating memory is how a new repository acquires its own experience. The amount of experience matters as well. The Related-Lite99 initial pool holds 300 experience summaries distilled from its audited SWE-bench cold-start trajectories (Appendix D); with the agent and selection procedure fixed, resolution rises from 20.2% to 26.3%, 28.3%, and 35.1% as the pool grows from 50 to 100, 200, and 300 summaries (single runs for 50–200; the 300-summary value is the five-run mean of the main comparison). Fifty summaries do not yet help; each larger pool resolves more.

Table 5: Cross-repository transfer in the main comparison. Cross-repository lessons are the percentage of selected lessons whose source repository differs from the target. Other repositories only excludes same-repository lessons at selection time, including lessons added online.

Benchmark	Cross-repository lessons (%)	ContextGraph full pool	ContextGraph other repositories only	No memory
Verified500	78.4	**69.2**	63.0	61.6
Related-Lite99	86.2	**35.1**	28.7	21.4
DeepSWE113	64.1	**72.1**	67.5	60.8

4.7 Strong agents benefit too

Table 6: SWE-bench Verified500 with Gemini 3.1 Pro and live-SWE-agent. Both arms follow the data-separation protocol in Section 4.1. Gains are relative improvements, (ContextGraph − No memory)/No memory, in percent.

Metric	No memory	ContextGraph	Relative gain
pass@1	82.3%	**88.9%**	+8.0%
pass@3	82.6%	**90.4%**	+9.4%
pass@5	82.8%	**91.7%**	+10.7%

With Gemini 3.1 Pro and live-SWE-agent on Verified500, ContextGraph raises pass@1 from 82.3% to 88.9% and pass@5 from 82.8% to 91.7% (Table 6): one attempt with memory resolves more tasks than five without, and memory removes 37% of the failures that remain without it at pass@1 and 52% at pass@5. On 20 long-horizon LoLBench tasks with Codex and GPT-5.5, and a graph built from other Python tasks, ContextGraph solves 3 tasks at pass@1 (15.0%), while no memory and each of the six memory baselines solve 1 (5.0%); with three attempts, it solves 5 (25.0%; Appendix Table 15). The gain also holds across backbones on Related-Lite99, where the memory methods of Figure 5, including Mem0 (Chhikara et al., 2025), draw from the same 300-summary pool within each backbone: ContextGraph improves over no memory by 13.7, 10.0, and 8.2 points with DeepSeek V4 Pro, Kimi K3, and MiniMax M2.7 and has the highest rate with each. Memory also saves tokens: on Verified500 with DeepSeek V4 Pro, ContextGraph uses 201K tokens per task against 236K without memory, 14.8% fewer.

5 Conclusion

We asked what makes experience memory work for coding agents. Memory helps most when it stores abstract lessons rather than trajectories and lets the agent select them for each task; removing either from ContextGraph leaves at most 2.9 of its 13.7 points on Related-Lite99. On that foundation, updating memory after every task adds further gains. Lessons from other repositories alone carry more than half of the gain on Related-Lite99 and DeepSWE113, and the gain grows with the amount of experience, and strong agents benefit as well. We release the code, prompts, and graph schema so that others can build on these findings.

REFERENCES

Silin Chen, Shaoxin Lin, Yuling Shi, Heng Lian, Xiaodong Gu, Longfei Yun, Dong Chen, Lin Cao, Jiyang Liu, Nu Xia, and Qianxiang Wang. SWE-Exp: Experience-driven software issue resolution, 2025. URL https://arxiv.org/abs/2507.23361v2.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Soham Daga, Sreeram Sreedhar, and Dhravya Shah. Supermemory is state of the art in agent memory, 2026. URL https://supermemory.ai/research/longmembench/. Supermemory Research, LongMemEval-S evaluation report, May 2026.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Zhigang Hua, Yan Xie, Shuang Yang, Ge Liu, and Jiaxuan You. ExpGraph: Model-agnostic experience learning with graph-structured memory for LLM agents, 2026. URL https://arxiv.org/abs/2605.30712v1.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2403.08978.

Shuzheng Gao, Wenhao Zeng, Zhaojian Yu, Jianqiao Wangni, Chaozheng Wang, Kai Cai, Shilin He, and Michael R. Lyu. SWE-MeM: Learning adaptive memory management for long-horizon coding agents, 2026. URL https://arxiv.org/abs/2606.28434v1. Version 1.

Xuehang Guo, Zora Zhiruo Wang, Qingyun Wang, Graham Neubig, and Xingyao Wang. Enhancing software engineering through closed-loop memory optimization, 2026. URL https://arxiv.org/abs/2606.05646v1.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/6ddc001d07ca4f319af96a3024f6dbd1-Abstract.html.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks, 2026. URL https://arxiv.org/abs/2607.07946.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Kangsan Kim, Minki Kang, Taeil Kim, Yanlai Yang, Mengye Ren, and Sung Ju Hwang. Memory transfer learning: How memories are transferred across domains in coding agents, 2026. URL https://arxiv.org/abs/2604.14004v1. Version 1.

Yining Liu, Chenyu Yang, Boyan Li, Rui Mao, and Yuyu Luo. MIRA: Evidence-verified repair memory for Text-to-SQL correction, 2026. URL https://arxiv.org/abs/2608.06950v1.

Z. C. Luo, J. C. Guo, W. J. He, S. Y. Wang, J. C. Yu, F. M. Zhao, Y. Chen, T. Cao, L. Q. Liu, N. Zheng, W. Xu, J. Jiang, and Z. M. Zhao. AdaRepair-Mem: Adaptive experience orchestration for repository-level program repair, 2026. URL https://arxiv.org/abs/2609.20130v1.

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, 2023. URL https://arxiv.org/abs/2310.10134.

Fangwen Mu, Junjie Wang, Lin Shi, Song Wang, Shoubin Li, and Qing Wang. ExpeRepair: Dual-memory enhanced LLM-based repository-level program repair, 2026. URL https://arxiv.org/abs/2506.10484v2. Version 2; accepted at FSE 2026.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems, 2023. URL https://arxiv.org/abs/2310.08560.

Yun Peng. LoLBench: Long-horizon tasks on large software systems. Hugging Face dataset, 2026. URL https://huggingface.co/datasets/lolbench26/LoLBench.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025. URL https://arxiv.org/abs/2501.13956.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

SWE-agent Team. mini-SWE-agent: The minimal software engineering agent. Software repository, 2025. URL https://github.com/SWE-agent/mini-swe-agent.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, Ge Zhang, Jiaheng Liu, Xingyao Wang, Sirui Hong, Chenglin Wu, Hao Cheng, Chi Wang, and Wangchunshu Zhou. Agent KB: Leveraging cross-domain experience for agentic problem solving, 2025. URL https://arxiv.org/abs/2507.06229.

Haibo Wang, Zhuolin Xu, and Shin Hwei Tan. Testing refactoring engine via historical bug report driven LLM, 2025a. URL https://arxiv.org/abs/2501.09879v2.

Qihao Wang, Ziming Cheng, Shuo Zhang, Fan Liu, Rui Xu, et al. MemGovern: Enhancing code agents through learning from governed human experiences, 2026. URL https://arxiv.org/abs/2601.06789v2.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the International Conference on Machine Learning*, 2025b. URL https://arxiv.org/abs/2409.07429.

Peng Xia, Rujun Han, Zifeng Wang, Yanfei Chen, Yufan Zhuang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, Huaxiu Yao, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. RRSI: Regularized recursive self-improvement of agent harnesses, 2026. URL https://arxiv.org/abs/2609.24972v1. Version 1.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Yisen Xu, Jiayuan Zhou, Ruiqi Pan, and Tse-Hsun Chen. Reusing past repairs through hierarchical trajectory abstraction for coding agents, 2026. URL https://arxiv.org/abs/2607.29658v1. Version 1.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel Agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, 2025. doi: 10.18653/v1/2025.acl-long.1354. URL https://aclanthology.org/2025.acl-long.1354/.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=pUpzQZTvGY.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026b. URL https://arxiv.org/abs/2510.04618v3.

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, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Yujun Zhou, Kehan Guo, Haomin Zhuang, Xiangqi Wang, Yue Huang, Zhenwen Liang, Pin-Yu Chen, Tian Gao, Nuno Moniz, Nitesh V. Chawla, and Xiangliang Zhang. Getting better at working with you: Compiling user corrections into runtime enforcement for coding agents, 2026. URL https://arxiv.org/abs/2606.13174v1.

Jiayuan Zhu, Junde Wu, Minhao Hu, Shengda Zhu, Jiazhen Pan, Weixiang Shen, Yijun Yang, Fenglin Liu, Jianye Hao, Yueming Jin, Qirong Ho, and Min Xu. SWE Context Bench: A benchmark for context learning in coding, 2026. URL https://arxiv.org/abs/2602.08316v3. Version 3.

A MAIN BENCHMARK COMPARISONS

Table 7: Repair resolution rates (%). Panel A tabulates Figure 4; the LoLBench comparison is in Table 15. Panel B groups published reference results by their reported configuration. Bold marks the best result per benchmark in Panel A. An em dash indicates that the cited source does not report the corresponding benchmark.

A. Three-benchmark comparison

Method	Verified500	Related-Lite99	DeepSWE113
No memory	61.6	21.4	60.8
FAISS Flat	65.2	26.8	64.6
ExpeL	62.4	20.7	58.7
ExpeRepair†	63.6	31.3	64.4
ReasoningBank	67.0	24.9	61.7
ACE†	62.2	28.1	62.8
Supermemory	63.6	26.5	66.2
ContextGraph	**69.2**	**35.1**	**72.1**

Verified500, Related-Lite99, and DeepSWE113 use DeepSeek V4 Pro 0813 and mini-SWE-agent. Related-Lite99 and DeepSWE113 rates are means across five runs, each reset to the same initial memory pool, with tasks evaluated in issue creation-time order; per-run rates use completed evaluations as the denominator. ContextGraph adds lessons from completed successful and failed attempts to memory for subsequent retrieval; baselines use their open-source defaults for memory updates. The initial Verified500 memory draws on Nebius SWE-bench-extra trajectories, which contain none of the evaluated instances; the DeepSWE113 memory draws on trajectories from LoLBench repositories (Appendix C). † ExpeRepair and ACE are our adaptations to mini-SWE-agent. Method sources: ExpeL (Zhao et al., 2024), ExpeRepair (Mu et al., 2026), ReasoningBank (Ouyang et al., 2026), ACE (Zhang et al., 2026b), and Supermemory (Daga et al., 2026).

B. Published reference results, grouped by their reported configuration

Method	Backbone	Verified500 (%)	Related-Lite99 (%)	Source
ExpeRepair	Claude 3.5 Sonnet	57.2	—	ER
ExpeRepair	Claude 4 Sonnet	74.6	—	ER
ReasoningBank	Gemini 2.5 Flash	38.8	—	RB
ReasoningBank	Gemini 2.5 Pro	57.4	—	RB
No memory	Claude Sonnet 4.5	—	26.26	SC
Supermemory	Claude Sonnet 4.5	—	30.30	SC
Oracle summary	Claude Sonnet 4.5	—	34.34	SC

ER: ExpeRepair, Table 1; RB: ReasoningBank, Table 2; SC: SWE-ContextBench, Table 4. ER uses its own repair pipeline with occasional o4-mini fallback; RB uses mini-SWE-agent; SC uses Claude Code. Oracle summaries supply the known related history. Published scores use their original budgets and memory settings; Appendix G reproduces the full published SWE-ContextBench Lite table.

B STATISTICAL UNCERTAINTY

Table 8 gives 95% intervals for the main-text comparisons. For a rate p over n tasks we use the binomial standard error $\sqrt{p(1-p)/n}$; for a difference between two methods we combine the two standard errors as if the rates were independent. Both methods run on the same tasks, so a paired test would give narrower intervals; the values below are therefore conservative. On Related-Lite99 and DeepSWE113 the rates are five-run means; averaging runs reduces run-to-run variance but not the variance from sampling the tasks, which the binomial bound captures.

Table 8: Differences in percentage points with conservative (unpaired) binomial 95% half-widths. Bold marks differences that exceed their half-width. Ablation rows are on Related-Lite99 and measure the drop from the full system.

Benchmark	Comparison	Difference	Half-width
Verified500 (n=500)	ContextGraph − no memory	**7.6**	5.9
	ContextGraph − ReasoningBank	2.2	5.8
	Other repositories only − no memory	1.4	6.0
	Gemini 3.1 Pro pass@1: ContextGraph − no memory	**6.6**	4.3
Related-Lite99 (n=99)	ContextGraph − no memory	**13.7**	12.4
	ContextGraph − ExpeRepair	3.8	13.1
	Other repositories only − no memory	7.3	12.0
DeepSWE113 (n=113)	ContextGraph − no memory	11.3	12.2
	ContextGraph − Supermemory	5.9	12.0
	Other repositories only − no memory	6.7	12.5
Ablation (Related-Lite99)	Raw trajectories	**14.6**	12.3
	Static playbook	**14.3**	12.3
	Fixed top-k retrieval	10.8	12.6
	Static pool	6.4	13.0
	No graph links	4.0	13.1
	Planning-time search only	1.9	13.2

C OVERLAP AUDIT OF INITIAL MEMORY POOLS

Every initial memory pool of the main comparison (Table 2) was audited before evaluation so that no evaluated task, its reference patch, or a near duplicate of that patch can be retrieved. The audit has two steps and was applied to each of these pools in the same way. The curated episodic pool of the retry experiment (Section 4.5) predates this audit; that experiment therefore also reports its first-pass result with the ten near-duplicate targets removed.

Instance-level exclusion. Each source trajectory carries its task identifier, repository, and base commit. Trajectories whose identifier matches an evaluated task are removed. The Nebius SWE-bench-extra trajectories that seed the Verified500 pool come from a different task collection, and none of the 500 SWE-bench Verified evaluation instances appears among them, so the evaluated issues, their reference patches, and their hidden tests are absent from memory. The Related-Lite99 pool starts from SWE-bench trajectories and grows task by task during evaluation; its initial trajectories contain none of the 99 targets. The DeepSWE113 pool contains only LoLBench repository trajectories; no DeepSWE task or repository snapshot appears in it, and the LoLBench and DeepSWE113 repository sets are disjoint. The LoLBench pool excludes the 20 evaluated Python tasks.

Patch-similarity exclusion. Instance identifiers do not catch the same fix recorded under a different task. We therefore compare the reference patch of every evaluated task with every patch in the pool using token-Jaccard similarity, the criterion that identified ten near-duplicate sources in the curated episodic pool (Section 4.5). Pool entries with similarity of at least 0.85 to any evaluated task's reference patch were removed before the main comparison. Table 9 summarizes the audit for each pool: the instance check removed nothing, and the patch-similarity check removed between 3 and 18 entries per pool, at most 1.3% of any pool.

Table 9: Overlap audit of the initial memory pools used in the main comparison (sources as in Table 2). Counts are source trajectories. Instance match removes trajectories whose task identifier is an evaluated task; patch similarity removes trajectories whose patch has token-Jaccard similarity of at least 0.85 to an evaluated task's reference patch.

Pool	Source trajectories	Instance match	Patch similarity	Remaining
Verified500	1,762	0	18	1,744
Related-Lite99	1,500	0	6	1,494
DeepSWE113	500	0	3	497
LoLBench Python	400	0	5	395

Retrieval-time check. The selection logs of the main comparison record the source task of every lesson the agent selects. After the audit above, no initial-pool lesson originates from an evaluated task; lessons added online come only from tasks earlier in the stream and, when a task is attempted more than once (pass@k and the retry experiment), from earlier attempts at the same task. Section 4.6 uses the same logs to report how many originate from a different repository than the target. Official benchmark test results are never part of the agent's context or stored memory, including between repeated attempts (Section 4.1).

D ADDITIONAL SYSTEM COMPARISONS

These comparisons examine summary and episodic memory across models and attempt budgets. Two configurations of ContextGraph appear in them and are named explicitly here. The *summary-memory configuration* is the experience graph of Section 3 built from an initial pool of 300 experience summaries distilled from the audited SWE-bench cold-start pool (Appendix C), with the mini-SWE-agent scaffold and the main-comparison budget; it is the Related-Lite99 run of the main comparison, which uses the experience graph on all three benchmarks. Figure 5 and the pool-size study of Section 4.6 use it to examine other backbones and initial pool sizes. The *episodic-memory configuration* serves 291 curated repair records with prior code history, reference fixes, and verification hints; Tables 10 and 11 use it. Table 12 gives an early 50-task development comparison on SWE-bench Verified. Resolution rates are percentages. Each comparison runs its own no-memory control under its configuration, so rates are compared within a table.

Memory access and evaluation. The agent uses its full current context to search the experience pool, read records, and follow links to relevant strategies. In the backbone comparison, the Kimi K3 and MiniMax M2.7 runs use one attempt per task and report rates over completed evaluations; MiniMax M2.7 uses an API compatibility layer. The pool-size study varies the initial pool from 50 to 300 summaries (Section 4.6). In the curated-episode comparison, each pass adds the distilled lessons of its completed attempts to the pool before the next pass retries unresolved tasks (Section 4.5). Official test results are withheld from the agent throughout, including between attempts.

Source-similarity sensitivity. A separate analysis removes ten targets whose source fixes have token-Jaccard similarity of at least 0.85. These are distinct tasks, not target trajectories in the memory pool. On the remaining 89 tasks, DeepSeek V4 Pro resolves 21.35% with memory versus 14.61% without memory ($p = .031$). Appendix C applies the same criterion to every initial pool of the main comparison.

Table 10: Curated-episode Related-Lite99 replication, one pass per backbone. Resolution rates are percentages over completed evaluations. The last column gives no-memory/ContextGraph rates on the fixed 89-task subset after removing ten near-duplicate targets, followed by the paired p value. N/C only counts tasks resolved exclusively by no memory/ContextGraph.

Backbone	No memory	ContextGraph	N/C only	p	89-task subset
DeepSeek V4 Pro	14.14	21.21	0/7	.016	14.61/21.35 ($p = .031$)
DeepSeek V3.2 exp	16.33	20.20	4/8	.39	16.85/20.22 ($p = .55$)
GLM 5.2	17.17	24.24	1/8	.039	16.85/23.60 ($p = .070$)
Kimi K3	29.29	31.31	3/5	.73	29.21/31.46 ($p = .73$)

Table 11: One run of the Related-Lite99 retry experiment (Section 4.5) with curated episodic memory updated after every task and carried across passes; five-run means are in Table 4. Resolved counts are cumulative. Wins count tasks solved only with memory; losses count tasks solved only without it.

Retry budget	No memory	ContextGraph	Wins/losses
1 pass	14/99	**21/99**	7/0
2 passes	16/99	**25/99**	9/0
3 passes	16/99	**27/99**	12/1

Table 12: Separate 50-task development comparison. Completion uses all 50 tasks; resolution uses completed evaluations.

Method	Completion (%)	Resolution (%)
ContextGraph	96.0	66.7
ExpeL	100.0	64.0
FAISS Flat	98.0	61.2
No memory	100.0	58.0

Table 13: Experience-graph snapshot illustrating the schema, built from 1,795 Nebius SWE-agent trajectories (the audited Verified500 pool of Table 9 keeps 1,744 of its 1,762 source trajectories), using text-embedding-3-large vectors with 3,072 dimensions for every node. Playbook entries are indexed for retrieval and carry no edges.

Node label	Count	Role
Trajectory	1,795	Source episode
Fragment	13,813	Action and observation slice
ErrorPattern	586	Error type and keywords
Strategy	9,600	Extracted lesson (strategy or warning)
CanonicalRule	7,835	Consolidated rule
PlaybookEntry	8,910	Retrieval-ready entry
Community	781	Linked-fragment summary
ProblemSummary	1,795	Episode problem abstraction
Total nodes	45,115	
Relation	**Count**	**From → to**
ADDRESSES_ERROR	18,584	CanonicalRule → ErrorPattern
HAS_FRAGMENT	13,813	Trajectory → Fragment
IN_COMMUNITY	13,466	Fragment → Community
DERIVED_FROM	9,600	Strategy → Trajectory
MERGED_INTO	8,910	Strategy → CanonicalRule
CAUSED_ERROR	7,807	Fragment → ErrorPattern
SUMMARIZES	1,795	ProblemSummary → Trajectory
Total edges	73,975	Seven typed relations

E ADDITIONAL BENCHMARK AND AGENT RESULTS

This appendix tabulates the DeepSeek V4 Pro column of Figure 5 (Table 14) and gives the LoLBench comparison from Section 4.7. The three-benchmark comparison is tabulated in Table 7; additional comparisons appear in Appendix D.

Table 14: DeepSeek V4 Pro column of Figure 5 (300-summary initial pool; the ContextGraph and no-memory values are those of the main comparison). Bold marks the best automatic memory result; the oracle receives a known related summary.

Method	Resolution rate (%)
No memory	21.4
Random summaries	27.1
Mem0	26.8
ContextGraph	**35.1**
Oracle summary	27.8

LoLBench Python subset. We select 20 Python tasks from LoLBench (Peng, 2026), which evaluates long-horizon development from enhancement proposals in large software systems. The agent is Codex with GPT-5.5. The initial experience graph is built from other Python tasks, separate from the 20 evaluation tasks. During evaluation, ContextGraph adds lessons from completed attempts, including successes and failures, for subsequent retrieval. Baseline memory updates follow each method's open-source defaults.

Table 15: Success rates on 20 LoLBench Python tasks with Codex and GPT-5.5, as reported in Section 4.7. pass@k reports the fraction of tasks solved within k attempts.

Method	Metric	Success rate (%)
No memory	pass@1	5.0
FAISS Flat	pass@1	5.0
ExpeL	pass@1	5.0
ExpeRepair	pass@1	5.0
ReasoningBank	pass@1	5.0
ACE	pass@1	5.0
Supermemory	pass@1	5.0
ContextGraph	pass@1	**15.0**
ContextGraph	pass@3	**25.0**

F FULL-CONTEXT AGENT-DIRECTED MEMORY SELECTION

At step i of task q_t, the agent uses the complete context $C_{t,i}$: the issue and conversation, inspected code, edits, actions, and observed failures. It can search the whole experience pool, read source records, follow links between strategies and errors, and compare lessons across repositories. Its selection follows Equation 3: searches return ranked candidates, and the agent decides which records to inspect and retain, so the number of selected lessons varies by task. Each memory action m_j and its result extend the history to h_{j+1} for the next action; at STOP, the kept lessons form $\mathcal{M}_{t,i}$.

The selected memory contains strategies and warnings, their source context, and the conditions under which they are useful. Before planning, selection helps choose an investigation strategy. After an error, the updated context helps revise the diagnosis and find additional guidance. After each completed task, the graph incorporates new experience for subsequent tasks. The pool therefore evolves along the issue-ordered task stream; official benchmark test results are never part of the agent's context or stored memory.

G PUBLISHED SWE-CONTEXTBENCH REFERENCE RESULTS

Table 16 reproduces the resolved rates from Table 4 of Zhu et al. (2026). Supermemory leads the automatic memory settings reported there; oracle summaries use known task relationships to supply the relevant history. These published results provide context for our Related-Lite experiments. Direct comparisons require matching task IDs, backbone, agent, and attempt budget.

Table 16: Published SWE-ContextBench Lite results: 99 related tasks, Claude Sonnet 4.5. Rates are copied as reported, including OpenViking's 29.20%. Δ is the percentage-point change from no context. Oracle rows are separated from automatic retrieval.

Setting	History access	Resolved (%)	Δ (pp)
No context	None	26.26	0.00
Free context	Agent-selected trajectories	26.26	0.00
Free summary	Agent-selected summaries	22.22	-4.04
Mem0	Automatic retrieval	24.24	-2.02
OpenViking	Automatic retrieval	29.20	$+2.94$
Supermemory	Automatic retrieval	**30.30**	$+4.04$
Oracle context	Supplied related trajectory	27.27	$+1.01$
Oracle summary	Supplied related summary	**34.34**	$+8.08$

On the separate 376-task set, their Table 3 reports a best resolution rate of 23.94% for GPT-5.3 Codex with oracle summaries. Supermemory's 55.95% in their Table 4 is a test-level FAIL_TO_PASS rate, not task resolution.

LIMITATIONS

Our ablations do not isolate a lesson's applicability condition: the raw-trajectory variant removes abstraction together with the rest of the lesson format, so we identify abstraction and agent-directed selection, not condition matching itself, as the choices that carry the gain. A variant that keeps both but strips only the condition would measure the condition's share. All ablations use Related-Lite99 with one backbone; repeating them on other benchmarks and backbones would show how far each takeaway generalizes. A retry control in which a no-memory agent reflects on its own failures (Shinn et al., 2023), longer task streams, and more languages and agent scaffolds are natural next steps.

REPRODUCIBILITY STATEMENT

The anonymous repository (Section 4.1) releases the experience-graph implementation used in our experiments (`research/`) with its extraction prompts and graph schema, the scripts and data that generate the figures (`research/figures`), and a packaged single-machine version of ContextGraph with an MCP interface. The main text defines the memory workflow, benchmark settings, and reported metrics. The appendices provide the overlap audit of every initial memory pool of the main comparison, numerical comparisons, and supplementary implementation details. ContextGraph's main comparisons include online experience updates; baseline updating follows the corresponding open-source implementation.

ETHICS STATEMENT

The reported evaluation uses software-engineering benchmarks. No confidential enterprise scenarios or data are included. Retrieved strategies and warnings guide the agent's investigation; patches are evaluated through the benchmark harnesses.

AI USE STATEMENT

Generative AI assisted literature review, method development, implementation, experiments, analysis, and writing, including drafting and revising the abstract.