CONTEXTNEXT: LEARNING TO PREDICT CONTEXT NEEDS AS KNOWLEDGE EVOLVES

Anonymous authors
Paper under double-blind review

ABSTRACT

External memory enables LLM agents to accumulate experience across tasks without updating model parameters. Yet stored experience does not automatically become effective context. Turning accumulated knowledge into useful context requires learning from past usage to guide knowledge selection and composition for subsequent tasks. Knowledge evolution introduces a further difficulty: revisions change the content available for selection, while the feedback guiding selection remains tied to earlier versions. The challenge is to improve the stored knowledge while preserving and updating what has been learned about its use. We introduce ConteXtNext (CXN), a framework for learning to select and allocate context as its knowledge base evolves. CXN decouples free-form knowledge organization from utility estimation, preserving traceable historical evidence across knowledge evolution and reconstructing utility on demand without maintaining persistent, task-agnostic scores. It formulates context construction as budgeted maximization of task-conditioned set utility. Task rewards and retrospective self-assessment provide approximate contribution credit, while coverage evidence accounts for overlap within candidate contexts. These estimates guide the selection of complementary knowledge through predicted marginal gains in set utility, with the initial context budget adapted to the information demands of related past tasks. We evaluate CXN across three task domains: AppWorld, CRMArena-Pro, and KernelBench. Across the evaluated LLM configurations, CXN improves task success over no-memory baselines by up to 18.7, 37.5, and 11.0 percentage points, respectively. Further analysis on KernelBench shows that CXN's performance gains are accompanied by lower context consumption. CXN uses the fewest context tokens among the compared methods, consuming only 25.2% of the no-memory baseline's context tokens while increasing task success from 74% to 85%.

1 INTRODUCTION

LLM agents operating across tasks can accumulate reusable procedures, constraints, and repair strategies in external memory. This knowledge remains open to revision: later interactions may reveal a missing precondition, narrow the scope of a rule, or motivate reorganizing several experiences. External memory thus provides a means of adapting an agent's knowledge throughout deployment without updating its model parameters. This form of adaptation underlies recent agent memory frameworks such as ACE (Zhang et al., 2026a), ReasoningBank (Ouyang et al., 2026), and ReMe (Cao et al., 2026).

Yet stored experience does not automatically become effective context for a new task. An agent must decide which knowledge is applicable, how different items complement one another, and how much information to supply. Past interactions can inform these decisions by revealing what helped execution, which contributions overlapped, and what information the task required. MemQ (Liao et al., 2026), for example, uses task rewards to update memory Q-values through credit propagation along a provenance graph of memory creation, demonstrating how execution feedback can guide memory reuse. Accumulated interactions thus provide evidence for learning how to select and compose context for subsequent tasks.

Knowledge evolution complicates learning from past usage because revisions, splits, and merges change the content and boundaries to which historical feedback refers. Existing designs often attach

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.

persistent scoring signals directly to stored knowledge units: MemQ maintains query-independent Q-values for memory entries (Liao et al., 2026), while EvolveMem associates typed memories with importance and reinforcement scores (Liu et al., 2026). This ties the granularity of knowledge organization to that of valuation. When knowledge is reorganized, these aggregate scores alone do not determine which historical contributions remain applicable or how they should be reassigned. Context construction further requires estimating what knowledge adds to a particular task given the other content selected: individually valuable entries may provide overlapping information, and their combined value cannot be inferred from independent scores alone. The resulting challenge is to support flexible knowledge reorganization while preserving applicable historical evidence for predicting task-dependent set utility and adapting context supply to information demand.

We introduce **ConteXtNext (CXN)**, a framework for learning to select and allocate context as its knowledge base evolves. CXN decouples knowledge organization from utility estimation by retaining historical observations tied to the knowledge versions they describe and recording explicit lineage across revisions, splits, and merges. Evidence transfer along this lineage obeys conservation constraints, preventing knowledge reorganization from duplicating inherited support. This separation enables knowledge to be freely expressed and reorganized in a file workspace while a structured evidence store maintains its correspondence with past observations. When constructing context, CXN reconstructs utility from this evidence for the current task and candidate set, without maintaining persistent, task-agnostic value scores.

Building on this evidence, CXN formulates context construction as budgeted maximization of predicted, task-conditioned set utility. It combines retrospective self-assessment of individual contributions with external task rewards to construct approximate credit targets. A task-conditioned predictor weights historical cases by task similarity and accounts for information overlap within each candidate context through coverage evidence. Because adding knowledge can change the contributions of both new and selected items, CXN greedily constructs context by the predicted gain in total set utility. A complementary estimator adapts the initial budget to the information demands of related historical tasks, limiting proactive supply when their support for the current task is weak. Subsequent executions provide new evidence for both knowledge refinement and future context decisions.

We evaluate CXN on AppWorld, CRMArena-Pro and KernelBench, covering application automation, enterprise workflows, and GPU kernel generation. Across the evaluated model configurations, CXN improves task success over no-memory baselines by up to 18.7, 37.5, and 11.0 percentage points, respectively. It also outperforms all compared baselines in four of the six experimental settings. On KernelBench, these improvements accompany the lowest context token consumption among the compared methods, at only 25.2% of the no-memory baseline's consumption. An AppWorld ablation provides complementary evidence about context construction: on 100 tasks, CXN achieves 76% success versus 72% for fixed-budget lexical retrieval while supplying 68.8% fewer initial-context characters on average. Context-use analysis further reveals a clear upward trend across both backbones in the fraction of initially supplied knowledge retrospectively identified as useful, suggesting more targeted context supply as experience accumulates.

Our main contributions are:

1. We introduce **ConteXtNext (CXN)**, a framework that decouples free-form knowledge organization from utility estimation. Explicit transformation lineage and evidence transfer under conservation constraints preserve the applicability of historical observations across knowledge revisions and reorganizations, enabling flexible knowledge evolution and utility reconstruction on demand without persistent, task-agnostic value scores.
2. We formulate **context construction as budgeted maximization of predicted, task-conditioned set utility**. Our predictor combines task rewards, retrospective self-assessment, and coverage evidence to estimate approximate contribution credit and guide greedy selection by marginal gains in total set utility. A complementary estimator adapts the initial context budget to historical information demand and the relevance of supporting tasks.
3. We demonstrate **improved task success across three task domains** over no-memory baselines. Analysis on KernelBench and an AppWorld retrieval ablation show higher success with lower context consumption, while context-use dynamics reveal an increasing fraction of initially supplied knowledge retrospectively identified as useful.

2 Related Work

2.1 Experience Memory and Context Construction

Experience-memory systems turn interaction histories into reusable knowledge without updating the base model. Reflection and experience-extraction methods retain lessons or cross-task insights (Shinn et al., 2023; Zhao et al., 2024), while skill- and workflow-oriented systems abstract successful behavior into executable skills or reusable routines (Wang et al., 2024; 2025). Test-time and self-evolving approaches maintain compact, revisable strategies, playbooks, and reasoning or procedural memories (Suzgun et al., 2026; Zhang et al., 2026a; Ouyang et al., 2026; Cao et al., 2026). Together, these works shift memory from episodic replay toward explicit abstraction and revision, with primary emphasis on how experience is represented, organized, and evolved rather than how evolving knowledge should be selected and composed into task-specific contexts.

Feedback-driven methods study how stored experience should shape new context. REMEMBERER and MemRL associate memories with reward or learned utility (Zhang et al., 2023; 2026b), while MemQ propagates downstream credit over dependencies among memories through provenance structures (Liao et al., 2026). EvolveMem adapts memory retrieval based on execution feedback, and MemCon, MeClear, and TRACE-Memory make retrieval, clearance, or admission task dependent (Liu et al., 2026; Jiang et al., 2026; Yang et al., 2026; Wang et al., 2026b). These methods move beyond relevance-only retrieval, but item-level values and dependency credit do not by themselves determine a memory's contribution given the current task and other selected content. MemQ's credit lineage propagates contribution credit to ancestor memories. CXN instead combines knowledge-version lineage with version-bound evidence and reconstructs task- and set-conditioned utility on demand, enabling context construction that accounts for task conditions and interactions among selected knowledge.

2.2 Memory Evolution and Evidence Continuity

Knowledge evolution requires memory systems to preserve correspondence between changing knowledge states and historical usage. A-MEM, Mem0, and MemoryOS support linking, consolidating, updating, and structuring long-term memory (Xu et al., 2025; Chhikara et al., 2025; Kang et al., 2025). MemGuard persists verifier-derived signals for lifecycle governance (Wang et al., 2026a). Source-oriented security work further preserves or constrains authority across memory transformation (Louck, 2026; Xu et al., 2026; Wu & Zhu, 2026). Collectively, these systems treat memory as managed, evolving state whose organization, lifecycle, and source remain relevant beyond its current text.

MemTX and ChronoMem address reliable evolution through validated publication, versioned state, and rollback (Li et al., 2026; Su et al., 2026). These mechanisms determine which state is canonical and recoverable, while governance and source tracking regulate how that state should be trusted and used. They do not determine which historical usage evidence remains applicable after knowledge is revised, split, or merged. Existing work advances experience construction, feedback-driven memory utility, and reliable memory evolution; CXN connects these directions by preserving version-bound usage evidence through knowledge evolution and reconstructing its task- and set-conditioned utility for subsequent context decisions.

3 Method

CXN learns to construct context from successive task interactions while keeping the agent's model parameters fixed. Given a new task q and the active knowledge units $\mathcal{A}_v$ at version v, the context construction objective is $\max_{S \subseteq \mathcal{A}_v} \widehat{F}_v(S \mid q)$ subject to $L(S) \leq B(q, v)$, where $\widehat{F}_v$ predicts set utility, L measures the delivered context length, and B is an initial budget estimated from historical information demand. Both utility and demand must be inferred from observations of earlier knowledge versions. CXN therefore preserves historical evidence independently of current knowledge organization, maintains its applicability through transformation lineage, and reconstructs task- and set-dependent utility when context is requested. The following sections develop these three steps and a greedy selection procedure (Figure 1).

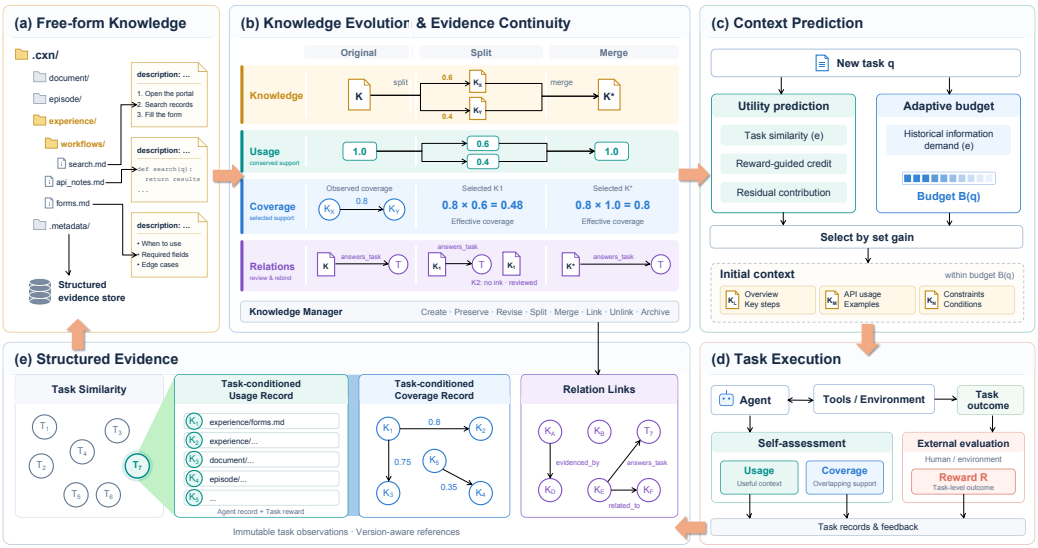


Figure 1: Overview of CXN. A free-form knowledge workspace is paired with persistent historical evidence. Transformation lineage maintains evidence applicability as knowledge evolves, under constraints that prevent duplication of inherited support. For each new task, the predictor reconstructs utility for candidate contexts, selects knowledge by gains in total set utility, and adapts the initial budget to historical information demand. New executions supply evidence for both knowledge refinement and future context decisions.

3.1 Free-form Knowledge and Persistent Evidence

Knowledge can range from short constraints to detailed procedures and code, and its appropriate granularity can change with further experience. Binding accumulated value to the current file boundaries would make reorganization also a decision about how to redistribute that value. CXN instead maintains $\mathcal{M}_v = (\mathcal{F}_v, \mathcal{E}_v)$: freely editable knowledge files $\mathcal{F}_v$ and structured evidence $\mathcal{E}_v$ from which utility can be reconstructed. Persistent state records observations and their correspondence to knowledge; utility is computed for a particular task and candidate context.

Knowledge workspace. The workspace contains distilled *Experiences*, imported *Documents*, and *Episodes* recording task interactions. These materials are addressed through active knowledge units in $\mathcal{A}_v$; each experience file defines one such unit. Experiences have a short description and a free-form Markdown body, and a Manager agent can revise their contents and reorganize the directory hierarchy. Knowledge identities are separate from file locations. Agents browse, search, and read a fixed version snapshot, so feedback can identify the content actually observed even after subsequent edits or moves.

Historical observations. A structured database stores task evidence separately from the knowledge files. For a completed task t, retrospective self-assessment records *Usage* observations o identifying useful knowledge from initial context or subsequent reads. Each observation retains its task, observed version, original targets $A(o)$, and ordinal rank. A directional *Coverage* edge $o' \to o$ records the fraction $c_{o'o} \in [0, 1]$ of o's task-relevant information supplied by an earlier-ranked observation o' from the same task. Coverage is directional: a complete procedure may cover a local tip without the reverse holding. These observations, Coverage edges, and external task reward R_t remain bound to the original execution.

Relations and transformation lineage. The evidence store also maintains typed semantic *Relations* $r = (x, \ell, y) \in \mathcal{R}_v$, such as `prerequisite` and `contradicts`. An `answers_task` relation links an experience or document to a task it directly or substantially answers, providing positive support for newly distilled knowledge before reuse. Relations describe semantic assertions;

transformation lineage separately records how old knowledge units correspond to their successors. Together, these records allow historical evidence to be interpreted against current content without storing persistent, task-agnostic utility scores.

3.2 Knowledge Evolution and Evidence Continuity

Knowledge transformations. After each task, the Manager can *create*, *preserve*, *revise*, *split*, *merge*, or *archive* experience. Pure relocation preserves identity; content transformations declare successor units and their correspondence to the originals. Usage observations and Coverage edges are not rewritten to match the new organization. Instead, lineage determines how their support reaches current knowledge, while semantic Relations receive versioned updates.

Conservation of inherited support. Let $\tau_v(b, a) \in [0, 1]$ be the inherited support from original unit b reaching active unit a. Preservation and continuous revision retain the full share; splitting distributes it according to semantic shares declared by the Manager; merging combines incoming paths. Creation has no inherited support, and archival makes the affected share unavailable. For every origin b, $\sum_{a \in \mathcal{A}_v} \tau_v(b, a) + d_v(b) = 1$, where $d_v(b)$ accounts for discarded or archived support. Shares multiply along a path and add across paths from the same origin, so splitting and subsequently merging retained branches cannot duplicate their support. Partial retention is represented by splitting and discarding. The Manager supplies semantic correspondence, and the system validates the transformation and conservation constraints. This conservation concerns inherited support, not task rewards or predicted utility.

Usage: retaining observed contributions. Keeping observation o and its original targets $A(o)$ fixed, we project its support onto current unit a as $h_v(o, a) = \max_{b \in A(o)} \tau_v(b, a)$. For one target, this is its transfer share; for multiple targets, the maximum yields one support value per observation and current unit. Thus an archived unit can lose active support without deleting the observation that once described it. These projections remain derived from the original records and lineage.

Coverage: tracking represented source knowledge. A historical Coverage edge remains attached to its original observations even when their targets are reorganized. For a candidate context S, the represented fraction of an observation's source knowledge is $z_v(o, S) = \frac{1}{|A(o)|} \sum_{b \in A(o)} \sum_{a \in S} \tau_v(b, a)$. The denominator retains the original targets, regardless of their current descendants. Selecting a descendant with share 0.6 realizes only that fraction of its origin; discarded shares remain unavailable. Section 3.3 uses this realization to estimate residual contributions within S, without modifying the historical Coverage edge.

Relations: maintaining semantic validity. Semantic Relations require valid endpoint correspondence. For an update from v to v', let σ map unchanged endpoints to themselves and revised or merged endpoints to their unique successors; task endpoints remain fixed. The updated relation graph is $\mathcal{R}_{v'} = \{(\sigma(x), \ell, \sigma(y)) : (x, \ell, y) \in \mathcal{R}_v^{\text{cont}}\} \cup \mathcal{R}_{v \to v'}^{\text{edit}}$. Here $\mathcal{R}_v^{\text{cont}}$ contains relations eligible for automatic continuation, and $\mathcal{R}_{v \to v'}^{\text{edit}}$ contains assertions supplied or replaced by the Manager. Relations involving a split require explicit reassignment. A numerical inheritance share alone cannot determine whether a successor remains a prerequisite or answers a task; the system validates the updated graph and preserves prior versions. Detailed update rules appear in Appendix B.

Knowledge changes, lineage, and relation updates are published together. The resulting state supports flexible reorganization while retaining the historical observations needed to reconstruct utility; it does not carry forward a fixed value for each successor.

3.3 Task-Conditioned Set Utility and Adaptive Budgeting

The preceding rules connect historical evidence to current knowledge. To evaluate a candidate context, CXN predicts its total contribution credit as $\widehat{F}_v(S \mid q) = \sum_{a \in S} \widehat{U}_v(a \mid q, S)$. Each term depends on both the task q and the entire set S: adding knowledge can change the contribution attributed to items already selected. We construct approximate credit targets, predict their relevance to the new task, estimate a budget, and then select by gains in this set objective.

Credit targets conditioned on the candidate set. Coverage determines the residual contribution $\rho_v(o, S) = 1 - \max_{o' \to o}[c_{o'o}\, z_v(o', S)]$ after accounting for information supplied by other knowledge in S. The maximum is zero without a reported edge. Taking the largest realized coverage avoids assuming that overlaps from different sources are disjoint. For task t and current unit a, let o_{ta} denote the representative positive observation after consolidating Usage and active `answers_task` support. The representative can depend on v and S. Its effective contribution and reward-guided target are $e_{ta}^{(v)}(S) = \rho_v(o_{ta}, S)\, h_v(o_{ta}, a)$ and $y_{ta}^{(v)}(S) = g(R_t)\, e_{ta}^{(v)}(S)$, respectively. Here g maps external reward to a bounded, nonnegative multiplier, defaulting to 1 without an applicable reward. Self-assessment and answer support identify contributions, h_v connects them to current content, and ρ_v adjusts for the candidate set. Reward scales these approximate credit targets; their sum need not equal the task reward. Other semantic Relations guide discovery without propagating numerical credit, and an answer link does not imply Coverage of other observations.

Task-conditioned utility prediction. CXN retrieves related historical tasks and assigns similarity $s(q, t) \in [0, 1]$, yielding weights $w(q, t) = s(q, t)^\gamma$. Let $\mathcal{H}_a(q)$ contain the matched cases supporting current unit a, including cases whose residual credit becomes zero through Coverage; repeated records of one execution count once. A task-weighted regression with regularization $\lambda > 0$ yields

$$\widehat{U}_v(a \mid q, S) = \arg\min_{u \in \mathbb{R}} \Big\{ \lambda u^2 + \sum_{t \in \mathcal{H}_a(q)} w(q,t) \big(u - y_{ta}^{(v)}(S)\big)^2 \Big\} = \frac{\sum_{t \in \mathcal{H}_a(q)} w(q,t)\, y_{ta}^{(v)}(S)}{\lambda + \sum_{t \in \mathcal{H}_a(q)} w(q,t)}. \quad (1)$$

Similarity controls transfer between tasks, reward scales the targets, and Coverage makes them depend on the proposed set. This nonparametric estimate $\widehat{U}_v(a \mid q, S)$ is reconstructed from evidence for the current knowledge version, task, and context, without learning or maintaining a global score for a.

Adaptive context budgeting. The same historical cases estimate how much context to supply. Let $C_t(v)$ be the total default-delivery length of distinct current units supported by task t's Usage or answer Relations. Each unit is counted once at its full delivery length: inheriting a fraction of an observation does not shorten the knowledge needed to use it. For matched cases $\mathcal{H}^+(q)$ with positive weight and demand, set $s_* = \max_{t \in \mathcal{H}^+(q)} s(q, t)$ and compute

$$\begin{aligned} \widehat{C}(q, v) &= \frac{\sum_{t \in \mathcal{H}^+(q)} w(q,t) C_t(v)}{\sum_{t \in \mathcal{H}^+(q)} w(q,t)}, \\ B_{\text{hist}} &= \text{clip}\big(\lceil \eta \widehat{C}(q, v) \rceil, B_{\min}, B_{\max}\big), \qquad B(q, v) = \big\lceil B_{\min} + s_* \big(B_{\text{hist}} - B_{\min}\big) \big\rceil . \end{aligned} \quad (2)$$

Here η allows variation in demand, and $B_{\min}, B_{\max}$ bound initial supply. Normalization prevents history size alone from increasing the estimate; interpolation toward $B_{\min}$ limits supply when even the strongest match is weak. With no contributing history, $B = B_{\min}$. Budgeting shares the task weights used for utility prediction, but reward does not scale information demand.

Selecting by gains in set utility. With utility and budget defined, CXN constructs S^{init} greedily, evaluating a candidate a by $\Delta_v(a \mid q, S) = \widehat{F}_v(S \cup \{a\} \mid q) - \widehat{F}_v(S \mid q)$. Each evaluation recomputes contributions across the prospective set, accounting for the candidate's effect on knowledge already selected. Starting from $S = \emptyset$, selection prioritizes gains among candidates fitting $B(q, v)$. Lexical relevance supplements this evidence-based objective for cold start and candidates without history; the implemented selector requires a positive combined gain and stops when no further candidate fits with positive gain. Selection therefore has no fixed item count. Appendix A gives the construction procedure and worked examples; prediction parameters, evidence aggregation, and delivery details appear in Appendix B.

The budget constrains initial delivery. During execution, the agent can read additional files or follow Relations to address information gaps. Its subsequent feedback supplies new observations and supports the next knowledge update, closing the loop between evidence preservation and context prediction.

Table 1: Task success (%) on three online task streams. Best results in each column are bold.

Method	AppWorld 417 tasks		CRMArena-Pro 200 tasks		KernelBench L1 100 tasks	
	MiMo	DeepSeek	MiMo	DeepSeek	MiMo	DeepSeek
No Memory	58.99	84.17	39.50	38.50	74.00	99.00
ACE	71.94	86.33	43.50	46.00	74.00	**100.00**
ReasoningBank	69.30	85.37	43.50	45.50	84.00	**100.00**
ReMe	72.66	87.05	37.00	41.50	68.00	**100.00**
MemQ	75.30	89.45	44.00	52.00	**89.00**	**100.00**
CXN	**77.70**	**92.81**	**44.50**	**76.00**	85.00	99.00

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmarks and protocol. We evaluate memory-supported context construction on three online task streams and report results separately for each benchmark. For AppWorld (Trivedi et al., 2024), we use all 417 tasks in the `test_challenge` split to evaluate general experience reuse. From CRMArena-Pro (Huang et al., 2025), we construct a frozen two-phase stream of 200 B2B single-turn operational tasks: 100 `activity_priority` tasks for experience acquisition, followed, after a durable publication barrier, by 100 `wrong_stage_rectification` tasks for cross-workflow transfer. Both task types are evaluated using exact match. For KernelBench (Ouyang et al., 2025), we evaluate all 100 Level-1 tasks. Each task contributes one final benchmark outcome. At its actual retrieval time, a task may access only method state, including usage evidence where applicable, that has already been settled and durably published. We freeze the task admission order and persist the corresponding committed memory frontiers for reproducibility.

Baselines and models. We compare CXN with an otherwise matched No Memory agent and four experience-memory baselines: ACE (Zhang et al., 2026a), ReasoningBank (Ouyang et al., 2026), ReMe (Cao et al., 2026), and MemQ (Liao et al., 2026). Within each setting, all methods use the same backbone model, task-facing tool interface, task order, main-agent execution budget, and authoritative evaluator. These implementations are benchmark adaptations for comparing memory mechanisms, rather than reproductions of the original papers' complete agents. We evaluate MiMo-V2.5-Pro and DeepSeek-V4-Flash as backbone models.

Metrics. Our primary metric is **task success rate**, computed from the authoritative evaluator for AppWorld and CRMArena-Pro and from verifier-confirmed functional correctness for KernelBench. We report successful tasks with their denominators and do not average scores across domains. We additionally track context tokens used for context construction and allocation where available, total logged model tokens including memory operations, cost when usage and pricing records are complete, and evaluation coverage when no authoritative outcome is available.

4.2 MAIN RESULTS

Table 1 reports task success across three online task streams, measuring how effectively different memory methods reuse accumulated experience across tasks.

On AppWorld, CXN improves over No Memory and the strongest reported memory baseline with both backbones. With MiMo, CXN solves 324/417 tasks (77.70%), compared with 246/417 (58.99%) for No Memory and 314/417 (75.30%) for MemQ. With DeepSeek, CXN solves 387/417 tasks (92.81%), compared with 351/417 (84.17%) for No Memory and 373/417 (89.45%) for MemQ. Thus, cross-task memory consistently improves over the matched no-memory agent in these runs, and CXN provides an additional gain over the strongest reported memory baseline.

On CRMArena-Pro, CXN and MemQ perform similarly with MiMo (44.50% versus 44.00%). With DeepSeek, CXN achieves 76.00%, compared with 38.50% for No Memory and 52.00% for MemQ.

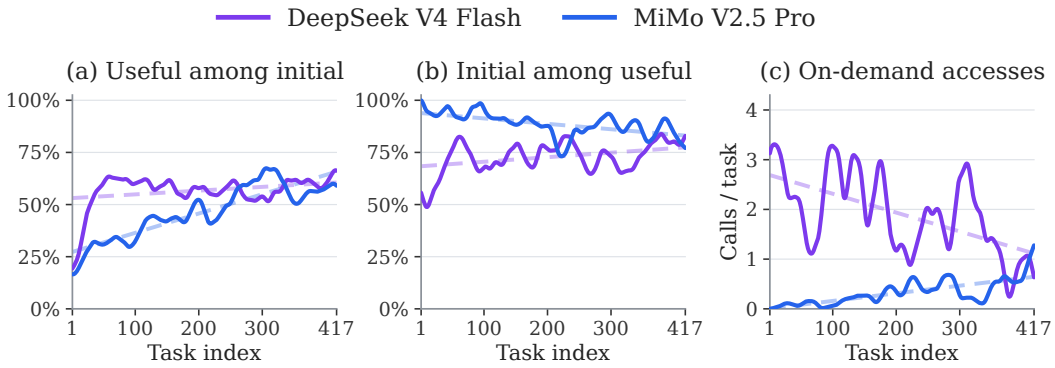


Figure 2: Context-use dynamics on the same 417 AppWorld test-challenge tasks. Curves use a 10% smoothing window; dashed lines indicate linear fits.

The gain is consistent with CXN consolidating prior successes and failures into reusable procedural guidance for sales-stage reasoning, query validation, and error recovery, and applying this knowledge across related workflows. CXN also uses additional memory-side computation and evidence-backed access to prior trajectories; therefore, the result reflects its end-to-end experience reuse capability under the evaluated resource profile.

On KernelBench Level 1 with MiMo, CXN reaches 85.00%, 11 points above No Memory and 4 points below the MemQ. The wider 68–89% range provides clearer separation among methods. With DeepSeek, all six methods attain 99–100% functional success: ACE, ReasoningBank, ReMe, and MemQ solve all 100 tasks, while No Memory and CXN solve 99. Thus, MiMo exposes more headroom for distinguishing memory methods, whereas DeepSeek is near ceiling at this level.

4.3 Ablation Study

Setup and variants. We evaluate two CXN ablations on the first 100 tasks of the AppWorld test-challenge permutation with seed 17, using MiMo V2.5 Pro. Full CXN uses the corresponding prefix of the 417-task run. *Without task reward* removes both reward weighting in utility estimation and external outcome feedback to the Manager, while retaining retrospective usage feedback. *FTS recall* replaces evidence-guided context construction with BM25-ranked full-text retrieval, bypassing historical-task matching, utility prediction, and adaptive budgeting. Its fixed budget of 40,000 Unicode characters equals CXN's maximum main-context budget $B_{\max}$. Both ablations retain knowledge maintenance and on-demand file access. We also include a *no-memory* reference without cross-task experience retention or retrieval.

Results and context use. Figure 3 shows success rates of 76% for full CXN, 74% without task reward, 72% with FTS recall, and 56% for the no-memory reference. The 2-point decrease without reward is consistent with external outcomes complementing retrospective self-assessment in the selection-and-maintenance loop. FTS uses more initial context despite its lower success rate: full CXN delivers an average of 11,322 characters per Recall, compared with 36,275 for FTS, a 68.8% reduction.

4.4 Further Analysis

Token usage analysis. Table 2 reports context token usage on KernelBench Level 1. CXN uses the fewest context tokens with both MiMo (159.66M) and DeepSeek (286.86M). With MiMo, CXN consumes only 25.2% of the context tokens used by No Memory while increasing task success from 74% to 85%. With DeepSeek, CXN uses 45.0% fewer context tokens than No Memory.

Table 2: Token usage on KernelBench L1.

Method	MiMo	DeepSeek
No Memory	634.42 (M)	521.75 (M)
ACE	540.23 (M)	649.34 (M)
ReasoningBank	465.11 (M)	479.30 (M)
ReMe	550.98 (M)	419.62 (M)
MemQ	173.65 (M)	296.59 (M)
CXN	**159.66** (M)	**286.86** (M)

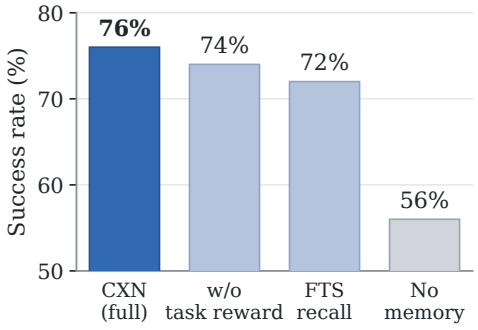


Figure 3: Task success rates for CXN, two memory ablations, and a no-memory reference on AppWorld with MiMo V2.5 Pro. The vertical axis spans 50–80%.

Context-use learning dynamics. Figure 2 compares how CXN uses accumulated knowledge with DeepSeek V4 Flash and MiMo V2.5 Pro. *Useful among initial* measures the fraction of initially delivered source items later labeled useful, and *initial among useful* measures the fraction of useful sources whose reported content was fully covered by the first delivery. From the first to the last 50 tasks, DeepSeek's useful fraction rises from 43.4% to 61.5%, initial coverage from 61.2% to 80.7%, and accesses fall from 2.68 to 0.66 per task. MiMo's useful fraction rises from 28.1% to 57.6%, while coverage decreases from 95.6% to 85.4% and accesses increase from 0.06 to 0.72. Both models therefore show a shared improvement: as experience accumulates, a larger fraction of initially supplied context is subsequently identified as useful. At the same time, their balance between initial delivery and subsequent exploration evolves differently. DeepSeek's initial coverage increases as supplementary accesses decline, whereas MiMo's coverage decreases as accesses increase. These different trajectories indicate that adaptation in context use also depends on the underlying model.

Paired task outcomes. Table 3 compares CXN and No Memory on matched tasks. "CXN only" and "No Memory only" reveal improvements and regressions hidden by aggregate success rates. Across five of the six benchmark–model settings, CXN-only successes outnumber No-Memory-only successes. The largest margins appear on AppWorld with MiMo (109 versus 31) and CRMArena-Pro with DeepSeek (79 versus 4). KernelBench shows a ceiling effect under DeepSeek, where 98 tasks are solved by both methods and only one differs in each direction.

Table 3: Paired task outcomes: CXN vs. No Memory (NM).

Model	Both		Only	
	Pass	Fail	CXN	NM
AppWorld				
MiMo	215	62	109	31
DeepSeek	343	22	44	8
CRMArena-Pro				
MiMo	67	99	22	12
DeepSeek	73	44	79	4
KernelBench L1				
MiMo	61	2	24	13
DeepSeek	98	0	1	1

5 Conclusion

We presented CXN, a framework for learning to select and allocate context as an agent's knowledge base evolves. By preserving version-bound evidence through explicit lineage and conservation-constrained transfer, CXN decouples free-form knowledge organization from utility estimation. It reconstructs task-conditioned set utility on demand and adapts initial context budgets using historical interaction evidence. Experiments across three task domains demonstrate gains in task success over no-memory agents. Further analyses show that higher success can accompany lower context consumption, while a growing fraction of initially supplied knowledge is retrospectively identified as useful as experience accumulates. These findings support maintaining historical usage evidence alongside evolving knowledge as a basis for adapting context selection and allocation to new tasks.

AI USE STATEMENT

In this work, we used generative AI tools for language polishing, improving the readability of the manuscript, and assisting with the search and discovery of related work. We reviewed the AI-assisted revisions to ensure that they preserved the intended technical meaning. We take responsibility for the final content of this work, including text revised with the assistance of generative AI.

REFERENCES

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822, 2026.

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.

Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. Crmarena-pro: Holistic assessment of llm agents across diverse business scenarios and interactions, 2025. URL https://arxiv.org/abs/2505.18878.

Eric Hanchen Jiang, Zhi Zhang, Yuchen Wu, Levina Li, Dong Liu, Xiao Liang, Rui Sun, Yubei Li, Edward Sun, Haozheng Luo, Zhaolu Kang, Aylin Caliskan, Kai-Wei Chang, and Ying Nian Wu. Memory as a controlled process: Learned adaptive memory management for llm agents, 2026. URL https://arxiv.org/abs/2607.13591.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory OS of AI agent. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25961–25970, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1318. URL https://aclanthology.org/2025.emnlp-main.1318/.

Xiaoyang Li, Yiqi Wang, Haohui Lu, Zhi Chen, Mo Li, Pingan Song, Mingkai Zheng, and Taotao Cai. Memtx: Transactional belief commit for stateful agent memory, 2026. URL https://arxiv.org/abs/2607.23929.

Junwei Liao, Haoting Shi, Ruiwen Zhou, Jiaqian Wang, Shengtao Zhang, Wei Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Bo Tang, et al. Memq: Integrating q-learning into self-evolving memory agents over provenance dags. *arXiv preprint arXiv:2605.08374*, 2026.

Jiaqi Liu, Xinyu Ye, Peng Xia, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Evolvemem: Self-evolving memory architecture via autoresearch for llm agents. *arXiv preprint arXiv:2605.13941*, 2026.

Yedidel Louck. Securing llm-agent long-term memory against poisoning: Non-malleable, origin-bound authority with machine-checked guarantees, 2026. URL https://arxiv.org/abs/2606.24322.

Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels?, 2025. URL https://arxiv.org/abs/2502.10517.

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 C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/980ea04d23d1f6908964eba2a74afe45-Paper-Conference.pdf.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Yongye Su, Wujiang Xu, Chaoji Zuo, and Elisa Bertino. Chronomem: Version control and semantic rollback for large language model agent memory, 2026. URL https://arxiv.org/abs/2607.27773.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

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. *Transactions on Machine Learning Research*, 2024. URL https://mlanthology.org/tmlr/2024/wang2024tmlr-voyager/.

Haoyu Wang, Guangyuan Dong, He Liang, Zijing Zhang, Jiachen Luo, Chuang Liu, Chao Xue, and Hao Tang. Memguard: Persisting verifier signals for llm-agent memory governance, 2026a. URL https://arxiv.org/abs/2608.21867.

Jing Wang, Zhu Wang, Yifan Guo, Yulong Yang, and Yunji Liang. Trace-memory: Public-conditioned retrieval and utility-aware evidence admission for personalized generation, 2026b. URL https://arxiv.org/abs/2608.08446.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Ming Wu and Pengyuan Zhu. Agent zero memory: Provenance-aware long-term memory for llm agents, 2026. URL https://arxiv.org/abs/2608.29606.

Jinghan Xu, Yiyong Xiao, Wanru Shao, Hankai Liu, and Xinjin Li. Memory provenance laundering in llm agents: A non-amplification firewall for persistent memory, 2026. URL https://arxiv.org/abs/2607.29167.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 17577–17604. Curran Associates, Inc., 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/19909c36f51abc4856b4560aff3d36d6-Paper-Conference.pdf.

Boyu Yang, Jiazheng Sun, Zilong Lu, Zhi Qiu, Xin Peng, and Jun Zheng. Meclear: Cooperative game-theoretic attribution and risk-aware memory clearance for long-horizon llm agents, 2026. URL https://arxiv.org/abs/2609.09115.

Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. Large language models are semi-parametric reinforcement learning agents. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 78227–78239. Curran Associates, Inc., 2023. doi: 10.52202/075280-3419. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/f6b22ac37beb5da61efd4882082c9ecd-Paper-Conference.pdf.

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, 2026a. URL https://arxiv.org/abs/2510.04618.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory, 2026b. URL https://arxiv.org/abs/2601.03192.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

A Algorithm Details and Worked Examples

A.1 Context Construction Procedure

Given a task q and a fixed knowledge version v, CXN constructs initial context through the following steps. The procedure connects the quantities in Section 3 without prescribing a particular retrieval backend or storage representation.

1. **Match history and identify candidates.** Retrieve related historical tasks, assign weights $w(q,t)$, and identify candidate units in $\mathcal{A}_v$. Lexical relevance also admits candidates without prior usage evidence.
2. **Resolve historical support.** Use transformation lineage to compute $h_v(o,a)$ from each observation's original targets. Consolidate Usage and answer support for each task–unit pair, retaining the historical Coverage edges and counting each execution once.
3. **Estimate initial demand.** Compute $B(q,v)$ from the delivery lengths supported by matched tasks using Equation 2; use $B_{\min}$ when no history contributes. Initialize $S = \emptyset$.
4. **Evaluate additions.** For each candidate a satisfying $L(S \cup \{a\}) \leq B(q,v)$, evaluate the prospective set $S' = S \cup \{a\}$. Compute its source realizations z_v, Coverage residuals ρ_v, representative contributions, and reward-guided targets. Reconstruct utilities with Equation 1 for all units in S', then obtain the set gain $\Delta_v(a \mid q, S)$.
5. **Select and repeat.** Prioritize feasible candidates by evidence-based gain, with lexical relevance as supplementary support (Section 3.3). Add the highest-priority candidate with positive combined gain and reevaluate the enlarged set. Stop when no feasible candidate has positive combined gain, and deliver $S^{\text{init}} = S$.

Inherited support is resolved for version v, while Coverage residuals and utilities are evaluated for each proposed set. Subsequent agent reads can supply additional context; their usage feedback becomes evidence for later tasks.

A.2 Evidence Continuity and Set-Dependent Credit

The following illustrative example uses hypothetical knowledge and feedback. One historical task has useful observations o_b and o_c targeting a procedure b and a local tip c, respectively. The earlier observation o_b covers 80% of o_c's task-relevant information, so $c_{o_b o_c} = 0.8$. Assume this is the only Coverage edge and c remains unchanged.

Splitting, merging, and discarding. Suppose b is split into b_1 and b_2 with inherited shares 0.6 and 0.4. Then $h_v(o_b, b_1) = 0.6$ and $h_v(o_b, b_2) = 0.4$, while $h_v(o_c, c) = 1$. If both branches are merged into m with full retention at a later version v', their paths give total support $\tau_{v'}(b, m) = 0.6 + 0.4 = 1$. Alternatively, discarding b_2 leaves active support 0.6 and unavailable support 0.4. In both cases, o_b and its original target remain unchanged. Conservation prevents duplication of support; the semantic validity of the declared correspondence remains a separate requirement.

Coverage under partial delivery. Consider the split state before any merge or discard. For $S = \{c, b_1\}$, only 0.6 of o_b's source is represented: $z_v(o_b, S) = 0.6$. Effective coverage of o_c is therefore $0.8 \times 0.6 = 0.48$, leaving $\rho_v(o_c, S) = 0.52$. Including both branches realizes the full source and lowers the residual to 0.2. The same historical edge supports both calculations.

From contributions to selection gains. For this example, let the single historical task have weight $w(q, t) = 1$, reward multiplier $g(R_t) = 1$, and regularization $\lambda = 1$. With no additional evidence, Equation 1 reduces to $\widehat{U}_v(a \mid q, S) = e^{(v)}_{ta}(S)/2$. Thus b_1 and b_2 contribute 0.3 and 0.2 whenever selected, while c contributes $\rho_v(o_c, S)/2$. Table 4 shows the resulting set utilities.

Table 4: Illustrative set utilities after splitting b. All values use the same historical observations and Coverage edge.

Context S	$z_v(o_b, S)$	$\rho_v(o_c, S)$	$\widehat{F}_v(S \mid q)$
$\{c\}$	0.0	1.00	0.50
$\{c, b_1\}$	0.6	0.52	0.56
$\{c, b_2\}$	0.4	0.68	0.54
$\{c, b_1, b_2\}$	1.0	0.20	0.60

Suppose c is already selected, the branches have equal delivery lengths, and only one branch fits the remaining budget. Under the evidence-based objective, adding b_1 yields gain $0.56 - 0.50 = 0.06$, whereas adding b_2 yields 0.04. Although b_1's individual contribution is 0.3, its addition also reduces the contribution attributed to c. This is why selection evaluates the change in total set utility. These gains isolate the evidence-based objective; the example omits lexical supplementation.

A.3 Budget Illustration

As a separate example, take the budget parameters in Appendix B and an estimated historical demand $\widehat{C}(q, v) = 20{,}000$ characters. The demand margin gives $B_{\text{hist}} = 30{,}000$. A strongest task match $s_* = 0.8$ yields $B(q, v) = 26{,}000$; holding estimated demand fixed, $s_* = 0.2$ yields $14{,}000$. Thus weaker matches move initial supply toward the minimum budget. Reward multipliers affect contribution credit but do not enter this demand calculation.

A.4 A Learned Experience File from AppWorld

The following Markdown file is reproduced verbatim from the final CXN memory of the 417-task AppWorld test-challenge run with MiMo V2.5 Pro and seed 17. It organizes a rule for using verified-purchaser evidence into applicability conditions, an operational pattern, supporting cases, and limitations.

The two tasks cited in the file occur at positions 83 and 404, and both have successful benchmark outcomes. At task 404, an earlier version of this file was delivered inline by Recall and marked useful in Record. The final version reproduced below also includes evidence from that later task.

File: `experience/collections-and-queries/amazon-verified-purchaser-filtering.md`

```
---
description: Filter Amazon product Q&A answers and reviews by the
is_verified field when a task requires trusting only verified
purchasers.
```

```
---

## Applicability

When a task specifies that only verified purchasers should be trusted
for product information (e.g., battery life, dimensions, quality
claims), the Amazon API provides an `is_verified` boolean on both Q&A
answers and reviews. Use this field to filter results before extracting
the answer.

## Pattern

1. **Q&A answers**: Each answer object includes `is_verified:
true/false`. When multiple answers exist for the same question,
verified and non-verified answers may provide different values. Filter
to `is_verified=true` first; if no verified answers exist, note the gap
rather than falling back to unverified answers.

2. **Reviews**: Each review object includes an `is_verified` field.
Apply the same filtering when the task requires verified purchaser
data.

3. **Conflict handling**: Verified and non-verified answers may
disagree (e.g., battery life reported as 13, 14, or 15 hours). The
verified purchaser answer is authoritative when the task restricts
trust to verified purchasers. Do not average or blend values across
verification status.

4. **Coverage gaps**: Some products may have no verified purchaser
answers or reviews for a specific question. Report the absence rather
than using unverified data.

## Evidence

- **Q&A case** (task 258796c_2): "How many hours does the battery of
Kobo Clara HD eReader last?" required trusting only verified
purchasers. Amazon Q&A (question_id 10765) had 3 answers; only
answer_id 21159 had `is_verified=true` stating "It lasts 15 hours."
Non-verified answers gave 13 and 14 hours. Task passed with reward 1/1
using the verified answer.

- **Reviews case** (task 258796c_3): "How many hours does the battery
of HP Pavilion 15 Laptop last?" required trusting only verified
purchasers. Amazon product 1570 had 9 reviews; only review_id 15968 had
`is_verified=true`, reporting "Lasts 31 hours." The other reviews
mentioning battery (28 and 30 hours) were from non-verified purchasers
and were excluded. All Q&A answers were also non-verified. Task passed
with reward 1/1 using the sole verified review.

## Risks and limits

- The `is_verified` flag indicates purchase verification, not expertise
or accuracy. A verified purchaser's claim can still be imprecise.
- If the task does not explicitly restrict to verified purchasers,
including unverified data may be appropriate.
- The API field name and semantics may vary across AppWorld versions;
confirm schema before relying on it.
```

A.5 Organization of Learned AppWorld Experiences

The same final snapshot contains 108 experience files in six directories. The tree below shows all six directories and 15 selected filenames, including the example above; ellipses give the number of omitted files. The stored knowledge spans action workflows, API behavior, queries, entity relations, failure modes, and verification. Directory and file names are preserved from the snapshot.

```
experience/ (108 files)
|-- action-workflows/ (67 files)
|   |-- amazon-order-reimbursement-via-venmo.md
|   |-- gmail-email-body-data-extraction-to-csv.md
|   |-- phone-alarm-weekly-creation-from-email.md
|   |-- spotify-playlist-songs-backup-to-csv-email.md
|   `-- ... (63 additional files)
|-- api-behavior/ (3 files)
|   |-- gmail-forward-email-from-thread.md
|   |-- gmail-inbox-threads-filtering-by-label-and-read.md
|   `-- gmail-label-replacement-semantics.md
|-- collections-and-queries/ (10 files)
|   |-- amazon-cart-and-wishlist-cost-aggregation.md
|   |-- amazon-verified-purchaser-filtering.md
|   |-- spotify-liked-songs-filtering-by-library-membership.md
|   `-- ... (7 additional files)
|-- entities-and-relations/ (1 file)
|   `-- phone-contact-to-amazon-order-date-linking.md
|-- failure-modes/ (26 files)
|   |-- gmail-reply-to-email-draft-not-send-sends-immediately.md
|   |-- payment-card-number-float64-precision-loss.md
|   |-- spotify-bulk-library-migration-budget-exhaustion.md
|   `-- ... (23 additional files)
`-- verification-and-recovery/ (1 file)
    `-- amazon-order-venmo-reconciliation.md
```

B Implementation Details

Historical matching and credit aggregation. Historical tasks are retrieved using lexical matching and task embeddings, optionally followed by LLM reranking. We use the resulting similarity for both utility prediction and budgeting, with $\gamma = 3$ and a zero-centered regularizer $\lambda = 1$. Repeated records of the same execution count as one case. Representative observation selection prefers coverage residual, then inherited share, with ordinal rank breaking ties; Usage and answer support are consolidated rather than counted twice for the same task–unit pair. The reward mapping g is a clipped linear function configured for each external reward channel and defaults to 1 without an applicable reward. Answer Relations supply structural priorities when historical gains tie.

Relation continuation. In the relation-update rule in Section 3.2, $\mathcal{R}_v^{\text{cont}}$ excludes relations explicitly ended or replaced and those incident to split or archived units. Unchanged endpoints retain their identities; revised and merged endpoints map to their unique successors; task endpoints remain fixed. The Manager supplies $\mathcal{R}_{v \to v'}^{\text{edit}}$, including any relations retained after a split. The system validates the resulting graph, and previous snapshots retain its history.

Budget parameters and delivery. We use $B_{\min} = 10{,}000$, $B_{\max} = 40{,}000$ Unicode characters and a demand margin $\eta = 1.5$. Historical demand counts distinct supported units at their default delivery length: full text for experiences and documents, and summaries for episodes. The main budget counts delivered text, references, and formatting. An auxiliary allowance of $2{,}500$ characters supports experience descriptions for further discovery. These limits apply to initial delivery; additional file reads during execution remain available.