

# CLAWVM: Harness-Managed Virtual Memory for Stateful Tool-Using LLM Agents

Mofasshara Rafique  
Independent Researcher  
Switzerland  
mofasshara@hotmail.com

Laurent Bindschaedler  
Max Planck Institute for Software Systems  
Saarbrücken, Saarland, Germany  
bindsch@mpi-sws.org

## Abstract

Stateful tool-using LLM agents treat the context window as working memory, yet today’s agent harnesses manage residency and durability as best-effort, causing recurring failures: lost state after compaction, bypassed flushes on reset, and destructive writeback. We present CLAWVM, a virtual memory layer that manages state as typed pages with minimum-fidelity invariants, multi-resolution representations under a token budget, and validated writeback at every lifecycle boundary. Because the harness already assembles prompts, mediates tools, and observes lifecycle events, it is the natural enforcement point; placing the contract there makes residency and durability deterministic and auditable. Across synthetic workloads, 12 real-session traces, and adversarial stress tests, CLAWVM eliminates all policy-controllable faults whenever the minimum-fidelity set fits within the token budget, confirmed by an offline oracle, and adds median  $< 50 \mu\text{s}$  of policy-engine overhead per turn.

**CCS Concepts:** • **Software and its engineering** → **Virtual memory**; • **Computer systems organization** → **Reliability**; • **Computing methodologies** → **Intelligent agents**.

**Keywords:** stateful tool-using agents, context management, persistent memory, agent harness, virtual memory, openclaw

## ACM Reference Format:

Mofasshara Rafique and Laurent Bindschaedler. 2026. CLAWVM: Harness-Managed Virtual Memory for Stateful Tool-Using LLM Agents. In *Sixth European Workshop on Machine Learning and Systems (EuroMLSys '26)*, April 27–30, 2026, Edinburgh, Scotland Uk. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3805621.3807648>

## 1 Introduction

General-purpose tool-using large language model (LLM) agents, persistent assistants that manage email, calendars, smart-home devices, messaging, web automation, and dozens of other services, now run for hours or days, accumulating

state across hundreds of tool calls and multiple sessions. OpenClaw [27] and its derivatives (NanoClaw [33], PicoClaw [36], ZeroClaw [46]) are the most prominent examples; coding agents such as Claude Code, Codex CLI, and Cursor represent a narrower but equally demanding subclass. These *stateful tool-using agents* maintain durable state across sessions and operate over extended durations that span multiple context windows.

For these agents, the model context window is scarce working memory: it must hold constraints, the active plan, recent dialogue, and tool outputs needed for correct action, while transcripts and artifacts reside in durable backing stores. Correct operation depends on having the needed state resident at the right time and fidelity. When the needed state is absent, or the correct state is silently discarded, agents repeat work, violate user preferences, and lose progress mid-plan. These are not rare edge cases: public issue trackers and practitioner reports document recurring rules and constraints lost after context summarization [9, 21, 23, 39], state silently dropped on session reset [1, 14, 22, 24], and persistence operations that overwrite rather than merge [7].

Modern agent harnesses such as OpenClaw provide building blocks: pruning, retrieval, compaction, pre-compaction memory flushes, and external memory plugins [4, 20, 25, 27, 28]. These improve recall quality, but none provides an enforceable contract over residency, durability, or auditability. Practitioners have responded with configuration tweaks and add-on layers [4, 8, 16, 20, 26], yet flushes can still be bypassed, writeback can still be destructive, and no mechanism guarantees that critical state survives lifecycle transitions. Operating systems learned this lesson decades ago: when a runtime manages a fast, scarce tier alongside slow, durable storage, the answer is virtual memory, not best-effort heuristics. Prior work borrows the paging metaphor [13, 30], but no production harness enforces it.

We present CLAWVM, a harness-managed virtual memory layer that closes this gap (Figure 1). CLAWVM manages agent state as typed *pages* that can be kept at full detail, compressed, reduced to structured fields, or shrunk to a pointer under token-budget pressure. Each page carries a minimum-fidelity invariant (how far it may degrade before reclaiming space), and the harness enforces these invariants at every lifecycle



This work is licensed under a Creative Commons Attribution 4.0 International License.

*EuroMLSys '26, Edinburgh, Scotland Uk*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2605-7/2026/04

<https://doi.org/10.1145/3805621.3807648>

boundary through staged, validated writeback. When invariants are violated or state is lost, the system raises observable *faults* that make the failure diagnosable and replayable.

Only the agent harness mediates between the model and its state. Structurally, it is an OS kernel for agent state: it controls what stays resident and when state becomes durable. Placing the memory contract here makes residency and durability explicit, deterministic, and replayable without retraining models or replacing retrieval backends. Multi-resolution representations prevent compaction from being all-or-nothing: pages degrade gracefully, preserving enough to reconstruct. Validated writeback prevents the destructive overwrites of today’s free-form flush prompts. Together, multi-resolution representations, validated writeback, and observable faults close three gaps that retrieval quality, compaction tuning, and external memory cannot: critical state must survive destruction, dirty state must be committed before it is lost, and recall failures must be diagnosable.

We implement a CLAWVM prototype with a harness integration layer and evaluate the policy and fault model on OpenClaw-derived workloads with deterministic replays, comparing against retrieval-only and practitioner-configured compaction+retrieval baselines [25, 26, 28]. We also validate the hook integration in a live harness across lifecycle regressions, deterministic replay suites, and trace-derived replays from real coding-agent sessions, following the evaluation methodology of recent memory benchmarks [3, 35].

Across four OpenClaw-derived workload families and six token-budget levels, CLAWVM eliminates all policy-controllable faults (refetch, duplicate-tool, post-compaction bootstrap, and flush-miss) from 67.8 mean faults per workload-budget configuration (retrieval baseline) and 1.5 (practitioner-configured compaction+retrieval) to zero, and reduces paging instability by 77.4% and 11.4% respectively. An offline oracle with future knowledge of demand confirms zero remaining headroom: the online policy already achieves the optimum fault count. These results hold across all token budgets, 12 diverse real-session traces, and 30 task-level replays (100% success vs. 76.7% for the practitioner-configured baseline at the tightest budget), while adding median  $< 50 \mu\text{s}$  of policy-engine overhead per turn.

This paper makes the following contributions:

- A virtual memory contract for agent state: typed pages with minimum-fidelity invariants and multi-resolution representations under a token budget.
- A fault model that makes memory-management decisions observable and replayable, with offline oracle analysis for tuning and regression.
- A staged writeback protocol with deterministic, non-destructive commit at every lifecycle boundary.
- A CLAWVM prototype evaluated on synthetic, trace-derived, and adversarial workloads.

## 2 Background and Motivation

Stateful tool-using agents operate under a long-running control plane, the *agent harness*, that routes sessions, assembles each model call from conversational context and durable state, mediates tool invocation, and emits lifecycle events such as compaction, pruning, and reset [27, 29]. Because the harness manages both tiers, every prompt-assembly decision (what to include, at what fidelity, what to drop) is a page-replacement decision: the context window is physical memory, durable stores are disk.

Harnesses like OpenClaw provide practical building blocks: pruning low-value spans, retrieval via BM25 (term-frequency ranking) and vector search, compacting transcripts near the model’s limit, and inserting a pre-compaction flush turn that asks the model to externalize durable notes [25, 27, 28]. External layers (Mem0, Cognee, QMD) add auto-capture, hybrid recall, and graph-backed retrieval [4, 16, 20]. These mechanisms improve recall but treat residency and durability as best-effort: nothing enforces what must stay resident, when state becomes durable, or how paging decisions can be audited.

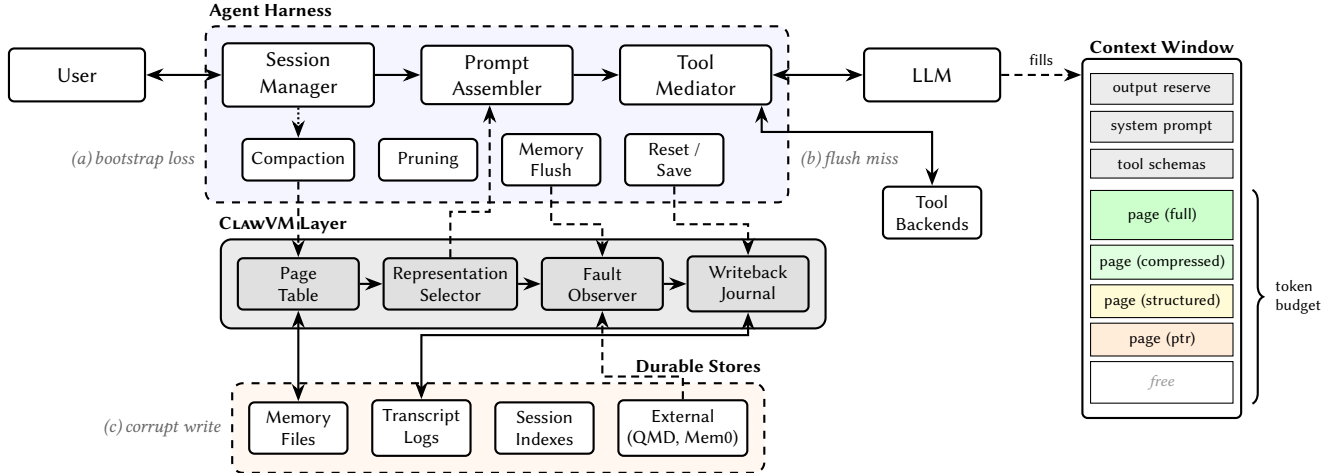
The consequences are concrete. Consider an agent 40 turns into a morning routine orchestrating email triage, calendar conflicts, and smart-home configuration, with loaded preferences (meeting priorities, lighting schedules, notification routing), a multi-step plan, and results from a dozen tool calls. Compaction fires: the summary preserves the high-level goal but drops the plan’s current step, routing rules, and evidence that conflicts were already resolved. The agent re-queries calendars it already inspected (duplicate tool calls), routes a notification to a deprioritized channel (bootstrap loss), and restarts from step 1 of a plan it had completed through step 5. On reset, 40 turns of accumulated decisions vanish; the runtime evicts dirty state without writing it back, and reload carries no guarantees.

These failures are not isolated. Field reports document three recurring classes at lifecycle edges: *residency failures* (compaction drops directives and tool outputs [9, 21, 23, 39, 40]), *durability failures* (flushes bypassed on reset, destructive overwrites [1, 7, 14, 18, 22, 24]), and *observability failures* (empty recall without reason codes [6, 37]). Practitioners trace these to three root causes: **capture is optional**, **recall is optional**, and **compaction is destructive** [28, 34, 45].

Existing mitigations (tuned configs, external memory layers [4, 8, 16, 20, 26]) cannot create an enforceable contract: the harness controls prompt assembly, tool mediation, and lifecycle events, yet exercises that control without an enforceable contract.

From these observations we derive six requirements, independent of model choice or retrieval backend:

- **Invariants survive destruction.** The harness must restore instructions and constraints deterministically after compaction and reset.



**Figure 1.** Architecture of a stateful tool-using agent with the CLAWVM layer (shaded). The agent harness manages sessions, assembles prompts, mediates tools, and emits lifecycle events (compaction, pruning, flush, reset). CLAWVM interposes at the harness level: the page table and representation selector feed prompt assembly, the fault observer instruments paging and lifecycle behavior, and the writeback journal enforces validated, non-destructive persistence at all lifecycle boundaries. The context window (right) shows how pages at varying fidelity levels fill the token budget. Letters mark failure points from field reports (Section 2): (a) post-compaction bootstrap loss, (b) flush miss on reset, (c) destructive writeback to durable stores.

- **Capture and recall are policy, not discretion.** For designated state, the harness drives capture and recall instead of relying on the model to flush or search.
- **Durability is lifecycle-complete.** The harness must commit dirty state at every boundary where the runtime would otherwise destroy the only copy.
- **Writeback is validated and non-destructive.** Updates must pass deterministic checks and use append/merge semantics; destructive overwrites must be rejectable.
- **Recall is observable.** The runtime must distinguish “no match,” “denied,” and “backend error” via reason codes.
- **Eviction is cost-aware.** Eviction decisions must account for the cost of re-running tool calls.

CLAWVM is a harness-managed virtual memory layer that satisfies these six requirements. The next section describes its design: typed pages, multi-resolution representations, an observable fault model, and a validated writeback protocol.

### 3 Design

CLAWVM addresses the six requirements from Section 2 through three enforced choices at the harness level. We first formalize the budget constraint, then describe each choice.

We model the agent runtime as an event stream over sessions. Each event may trigger tool execution, memory lookup, and a model call. After reserving tokens for output, system prompts, tool schemas, and a safety margin, the remaining capacity is the *token budget* for paged state. At each call, the memory manager selects a resident set that satisfies minimum-fidelity invariants within this budget and commits dirty state at lifecycle boundaries. CLAWVM makes three choices explicit and enforceable: (i) the unit of residency

is typed pages, (ii) admissible degradations under budget pressure are multi-resolution representations, and (iii) lifecycle points at which state becomes durable use validated writeback. Together these yield deterministic prompt assembly, controlled degradation instead of lossy summarization, observable faults, and lifecycle-complete persistence.

**Pages as the unit of residency.** CLAWVM treats all assistant-relevant state as typed pages. A page is a typed record with a stable identifier, scope, provenance, and a minimum-fidelity invariant. Pages are the unit of selection, eviction, and writeback, turning best-effort prompting into enforceable behavior. Scope (session-private vs. project-shared) and provenance (originating tool call or transcript span) are first-class metadata, required for observability and safe writeback. Table 1 lists six page types that cover failure-prone state, each with a minimum representation level (the four levels are defined below): *Bootstrap/Policy* (system instructions and procedural directives, whose loss causes “forgot its protocol” failures), *Constraint* (hard-pinned, never degraded below structured), *Plan* (goal and current step while active), *Preference* (scoped, may degrade to pointer), *Evidence* (tool outputs with deterministically resolvable pointers), and *Conversation Segment* (span identifiers and timestamp ranges).

**Multi-resolution residency.** Each page can be resident at one of four levels: *full* (verbatim excerpt), *compressed* (token-reduced text, e.g., LLMingua-2 [31]), *structured* (typed fields sufficient to satisfy invariants), and *pointer* (a resolvable handle plus minimal metadata). Under pressure, pages degrade along this chain while preserving invariants: constraints never drop below structured, and evidence pointers must

**Table 1.** Page classes, minimum-fidelity invariants, scope rules, and degradation paths in CLAWVM.

Type	Min. fidelity	Scope	Degradation path
Bootstrap	Struct. after compact, reset	P/S	F→St
Constraint	Hard-pinned at struct.	S	F→St
Plan	Goal + step	S	F→St→Pt
Preference	Scope, prov.	P	F→C→St→Pt
Evidence	Ptr resolves det.	S	F→C→St→Pt
Conversation	Span ID + time	S	F→C→St→Pt

F=full, C=compr., St=struct., Pt=ptr, P=proj., S=sess.

remain resolvable. Representations are generated at page creation time, not on demand under budget pressure: the harness extracts structured fields and computes token-reduced variants when a page is first ingested or updated. The representation selector then chooses among pre-existing variants, so budget-pressure decisions involve only table lookups and token arithmetic, not runtime LLM calls or compression passes. This prevents compaction from being all-or-nothing: structured and pointer representations preserve enough to reconstruct the full page on demand.

**Fault model.** Unlike OS page faults, which the kernel resolves transparently from disk, agent faults are silent: without instrumentation the harness cannot detect missing state, and recovery means regeneration. CLAWVM defines observable faults in two families, operationalizing the residency, durability, and observability failures from Section 2. *Working-set faults* capture missing state (residency failures): *refetch faults* (a tool result re-retrieved after eviction), *duplicate tool faults* (an equivalent tool call runs again because the result was evicted), *pinned invariant misses* (hard-pinned page missing at prompt assembly), and *post-compaction bootstrap faults* (required Bootstrap/Policy pages missing after compaction). *Durability faults* capture state lost at lifecycle boundaries (durability and observability failures): *silent-recall faults* (lookup returns empty when the backend actually denied access or errored) and *flush-miss faults* (dirty pages lost because the runtime destroyed the context before committing them). These are *policy-controllable* faults: a correct policy can prevent them. Physical insufficiency (token budget too small to fit all pinned pages) and semantic errors (model produces factually wrong updates) are outside policy control and evaluated separately. We also log *duplicate-signature alerts* when canonical tool signatures repeat despite a resident result; these are workload signals, not policy failures. We quantify instability with the *thrash index*: the ratio of paging events to hits over the run (Appendix A), adapting the classical thrashing ratio [5]; high values signal a working-set/budget mismatch.

**Selection policy.** Prompt assembly is a multi-choice knapsack with hard constraints. CLAWVM uses a deterministic two-phase policy: *Phase 1* installs all hard-pinned pages and minimum-required representations (surfacing invariant pressure if the minimum set cannot fit); *Phase 2* applies greedy upgrades (pointer→structured→compressed→full) by marginal utility per token until the budget is exhausted. Utility combines pin status, membership, recency, scope, and recompute cost, with deterministic tie-breaks (Appendix A). Listing 1 gives the full algorithm.

**Listing 1.** Deterministic multi-resolution prompt assembly in CLAWVM.

```
function assemblePrompt(session, candidates, budget):
  prompt <- fixedRegion(session)
  prompt <- prompt + minimumRequired(candidates)
  remaining <- budget - tokens(prompt)

  upgrades <- allRepresentationUpgrades(candidates)
  sort upgrades by (deltaUtility(u) / deltaTokens(u))
  descending

  for u in upgrades:
    if deltaTokens(u) <= remaining and respectsInvariants(u)
      and noConflict(u):
      applyUpgrade(prompt, u)
      remaining <- remaining - deltaTokens(u)

  return prompt
```

**Replay oracle.** To separate policy quality from budget insufficiency, CLAWVM supports offline replay-oracle analysis: given a trace and the same budget, an oracle with bounded future knowledge picks representations that minimize faults. The oracle’s *horizon h* is the number of future demand turns visible;  $h = \infty$  denotes unbounded lookahead. The *oracle gap* (online minus oracle fault counts) measures headroom versus unavoidable workload pressure.

**Validated writeback protocol.** CLAWVM treats persistence as a three-phase lifecycle-aware transaction: *structured staging* (the harness requests only typed, append/merge/set-with-version updates at lifecycle boundaries), *deterministic validation* (each staged update is checked for schema correctness, provenance, scope, non-destructive semantics, and policy compliance), and *scoped commit* (validated updates commit via deterministic merge rules at logged commit points). Rejected updates remain in the journal with reason codes; full rules appear in Appendix A.

## 4 Implementation

We implement a CLAWVM prototype in six Python modules (~1300 lines of non-comment code, zero external dependencies) targeting harnesses that expose session metadata, transcript logs, tool mediation, and lifecycle hooks [29]. Six lifecycle hooks map to engine operations; state is persisted via JSON snapshots between invocations. Pluggable adapters

normalize diverse sources (workspace files, transcript indexes, hybrid retrieval, QMD/Mem0/Cognee [4, 16, 20]) into CLAWVM pages, keeping the contract retrieval-agnostic. The six modules are: SessionPageTable, RepresentationSelector, WritebackJournal, FaultObserver, DecisionTrace (append-only JSON-lines audit log), and ClawVMEngine.

## 5 Evaluation

**Setup.** All experiments drive the CLAWVM engine directly over pre-generated workload specifications, bypassing the harness hook layer to isolate selection and fault-detection logic. Each workload defines pages (types, pin classes, token costs), per-turn demands, tool-call signatures, dirty-page sets, and lifecycle events; the engine runs the full per-turn pipeline and emits JSON-lines decision traces. Token budgets range from 120 to 500 in six steps; all runs are deterministic. Unless noted otherwise, aggregates average over 4 workloads  $\times$  6 budgets = 24 configurations. Each RQ maps to the requirements from Section 2; fault types and workloads are chosen so every requirement is exercised.

**Workloads.** Four synthetic families exercise different stress patterns: *evidence-heavy* (rotating tool signatures, periodic compaction), *interruption-heavy* (parallel tasks with compaction and reset), *lifecycle torture* (compaction every 3 turns, persistent dirty pages), and *multi-session interference* (2 concurrent sessions, interleaved compaction). Six lifecycle regression scenarios serve as a pass/fail gate (Table 3).

**Baselines.** We compare five policies (Table 2): *Retrieval* (no caching or writeback), *Retrieval+Cache* (Retr.+Cache; adds artifact caching), *Compaction-Hybrid* (Comp-Hybrid; prefetch + writeback at compaction but not reset, recency upgrades [26]), *CLAWVM* (full lifecycle completeness), and *Oracle* with horizon  $h=3$  (3-turn lookahead). Comp-Hybrid represents best-case practitioner effort: its configuration (prefetch, compaction writeback, recency upgrades, pointer resolution) is distilled from OpenClaw documentation, community guides, and issue-tracker discussions [8, 26, 34]. A real operator would need to discover and maintain each of these knobs independently. Even so, Comp-Hybrid has structural gaps that no amount of configuration can close: it lacks reset writeback (dirty state lost on /new), hard auto-pinning (bootstrap pages unprotected), and utility-based upgrades (Table 2). No hyperparameter search was performed for any policy; all run under equal token budgets.

We organize the evaluation around three questions:

- RQ1** How effectively does CLAWVM eliminate policy-controllable faults?
- RQ2** Which design features of CLAWVM are necessary?
- RQ3** Does CLAWVM generalize beyond synthetic workloads, and at what cost?

**Table 2.** Policy configuration knobs ( $\checkmark$  = enabled,  $-$  = disabled). Pin: auto-pin bootstrap/constraint pages. Pref.: prefetch before demand. WB-C/WB-R: writeback at compaction/reset. Upgrade: Phase 2 scoring strategy (none, recency, utility, or oracle). Resolve: follow pointer handles to reconstruct evidence content.

Policy	Pin	Pref.	WB-C	WB-R	Upgrade	Resolve
Retrieval	-	-	-	-	none	-
Retr.+Cache	-	-	-	-	none	$\checkmark$
Comp-Hybrid	-	$\checkmark$	$\checkmark$	-	recency	$\checkmark$
<b>CLAWVM</b>	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	<b>utility</b>	$\checkmark$
Oracle ( $h=3$ )	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	oracle	$\checkmark$

**Table 3.** Tier-1 lifecycle regression scenarios and gate assertions.

Scenario	Assertion
Post-compaction bootstrap	Bootstrap fault observable after compaction
Reset dirty-page flush miss	Flush-miss fault at reset boundary
Threshold jump race	Flush-miss fault from skipped pre-compaction commit
Silent recall visibility	Silent-recall fault from deny/error with reason code
Unsafe persistence rejection	Writeback rejected with DESTRUCTIVE_OP reason
Evidence churn duplicate tool	Duplicate-tool fault from repeated equivalent call

### 5.1 RQ1: Fault Elimination

We run all five policies (Table 2) across 4 workloads and 6 token budgets (120–500), measuring explicit faults (refetch, duplicate-tool, bootstrap, flush-miss, silent-recall) and thrash (Section 3). Six Tier-1 regression scenarios (Table 3) serve as a pass/fail gate.

All six Tier-1 regression tests trigger the expected fault or rejection with the correct reason code, validating instrumentation and enforcement. Table 4 presents aggregate reductions across 24 configurations. Against Comp-Hybrid, CLAWVM reduces explicit faults by 100% (1.5 mean to zero) and thrash by 11.4%. The 1.5 mean understates the gap: it averages over six budgets including loose ones (360–500) where all policies approach zero. At the tightest budget (120 tokens), Comp-Hybrid incurs 26 faults across 4 workloads, including 7 bootstrap and 5 flush-miss, while CLAWVM produces zero. These are not tuning failures; they are structural: Comp-Hybrid lacks reset writeback and hard pinning, so no configuration change can prevent them. Against retrieval, mean explicit faults drop from 67.8 to zero; thrash drops by 77.4%. Residual thrash comes from duplicate-signature alerts (inherent tool-signature reuse, not policy failures). These fault classes map directly to field-reported symptoms: refetch and duplicate-tool faults produce redundant API

**Table 4.** Aggregate reductions relative to baselines, averaged over 4 workloads  $\times$  6 budgets (120–500 tokens). *Explicit faults*: policy-controllable events (refetch, duplicate-tool, bootstrap, flush-miss, silent-recall). *Thrash*: paging instability (faults + duplicate-signature alerts over hits); residual thrash at zero faults reflects workload repetition.

Baseline	Explicit	$\Delta\%$	Thrash	$\Delta\%$
Retrieval	67.8	-100%	3.993	-77.4%
Retr.+Cache	10.8	-100%	1.642	-45.2%
Comp-Hybrid	1.5	-100%	1.017	-11.4%
<b>CLAWVM</b>	<b>0.0</b>	—	<b>0.901</b>	—
Oracle ( $h=3$ )	0.0	0.0%	0.901	0.0%

**Table 5.** Additive ablation at budget 180: explicit faults when enabling one feature on the bare baseline. Summed across 4 workloads. Boot. = bootstrap, Flush = flush-miss, D+R = duplicate-tool + refetch.

Variant	Boot.	Flush	D+R	Total
baseline	20	23	228	271
+pin	0	23	228	251
+resolve	20	23	0	43
+WB-C	20	3	228	251
+WB-R	20	20	228	268
+upgrade	7	23	42	72
<b>CLAWVM</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

calls [40], bootstrap faults cause the agent to “forget its protocol” [9, 21, 23, 39], and flush-miss faults mean decisions silently vanish on reset [1, 14, 22, 24]. Section 7 discusses the remaining semantic-correctness gap.

**Takeaway.** CLAWVM eliminates all policy-controllable faults at every budget level, matching an offline oracle ( $h=3$ ) on fault count.

## 5.2 RQ2: Feature Necessity

We assess which design features are necessary for fault elimination through subtractive and additive ablation. We remove one feature at a time from full CLAWVM at budget 180 (the tightest non-trivial budget) and measure explicit faults summed across all 4 workloads.

Table 6 presents the subtractive ablation results. Removing pointer resolution causes the most damage (126 faults across refetch and duplicate-tool), followed by writeback at compaction (20 flush-miss faults) and auto-pinning (9 bootstrap faults). Removing upgrade scoring or prefetch yields zero faults; other features compensate, demonstrating redundancy and no single point of failure. The additive ablation (Table 5) confirms: pointer resolution alone eliminates 84% of faults (271 $\rightarrow$ 43), auto-pinning eliminates all bootstrap faults, and compaction writeback cuts flush-miss by 87%.

**Table 6.** Subtractive ablation at budget 180: explicit faults when removing one feature from full CLAWVM. Summed across 4 workloads. Boot. = bootstrap, Flush = flush-miss, D+R = duplicate-tool + refetch.

Variant	Boot.	Flush	D+R	Total
<b>CLAWVM</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
-pin	9	0	0	9
-resolve	0	0	126	126
-WB-C	0	20	0	20
-WB-R	0	3	0	3
-upgrade	0	0	0	0
-prefetch	0	0	0	0

**LRU baseline.** To test whether the utility-based knapsack is necessary or a simpler heuristic suffices, we add a least-recently-used (LRU) variant that keeps all structural features (auto-pinning, writeback, prefetch, pointer resolution) but replaces utility scoring with pure recency-ordered upgrades. Across all 4 workloads and budgets 120–300, LRU achieves zero explicit faults and identical thrash to CLAWVM. This result reinforces the architectural claim: CLAWVM’s fault elimination is robust by construction, guaranteed by the Phase 1 structural constraints (hard-pinning, lifecycle writeback, pointer resolution) regardless of the Phase 2 upgrade heuristic. Utility scoring operates in the quality regime *above* the fault-free floor, determining which pages are upgraded to higher-fidelity representations when budget permits. We retain utility scoring by design because LRU makes strictly worse upgrade decisions in predictable situations: it spends budget upgrading recently touched but cheap-to-recompute evidence over stale but expensive-to-recompute results, and downgrades infrequently accessed preferences to pointer even when their base utility is high. Under tight budgets with bursty access patterns, this means the model sees more pointers and fewer full representations for high-value state, degrading downstream output quality without triggering a fault. Utility scoring accounts for recompute cost, pin class, and scope, preferring upgrades with the highest marginal value per token. Evaluating this quality gap requires end-to-end task metrics with a live model, which we leave to future work; for the structural safety properties evaluated here, any reasonable upgrade heuristic suffices once the enforcement layer is in place.

**Weight sensitivity.** We sweep each of the four utility scoring weights (pin boost hard  $\in [0.5, 4.0]$ , pin boost soft  $\in [0.0, 1.5]$ , recency  $\in [0.1, 1.2]$ , recompute cost  $\in [0.0, 1.2]$ ) independently across all 4 workloads at budget 180 (21 configurations, 84 runs). All configurations produce zero explicit faults and identical thrash (0.901). The weights are not brittle: any setting within the tested ranges achieves fault elimination, because the structural constraints prevent faults independent of upgrade ordering. Weights will differentiate

policies in prompt-quality evaluations, where the fidelity mix of resident pages affects downstream model behavior.

**Takeaway.** Pointer resolution, auto-pinning, and lifecycle writeback are the critical features; fault elimination is robust to the choice of upgrade heuristic and weight parameterization. Utility scoring provides a principled extension point for optimizing prompt quality above the fault-free floor.

### 5.3 RQ3: Generalization and Cost

We evaluate generalization and runtime cost using real-trace replay, task-granularity replay, adversarial stress tests, and overhead measurement.

We validate on 12 trace-derived replay workloads from real Claude Code sessions spanning coding, research, DevOps, trading, ML ops, web scraping, long-form writing, email writing, and product design (100-turn truncations; Appendix B). Conversion is deterministic (tool signatures canonicalized by name and argument schema, dirty pages inferred from tool outputs, lifecycle events preserved from transcript metadata); no traces were used to tune policy parameters. CLAWVM produces zero explicit faults across all 12 traces; Retrieval produces a median of 51 faults (range 49–77). Comp-Hybrid leaks exactly one flush-miss per trace—dirty state silently lost on every session because it lacks reset writeback. At 200 turns the pattern holds: zero faults for CLAWVM, while retrieval-only faults scale to a median of 83 (range 71–109; Appendix Table 10). Sensitivity analysis (50–400 turns) confirms CLAWVM maintains zero faults at all lengths, while retrieval-only faults grow near-linearly.

To move beyond per-turn aggregates, we construct 30 synthetic task workloads across four categories (coding, debugging, writing, ops; 5–14 turns each) and replay them at budgets 180 and 300. A task “succeeds” if it completes with zero explicit faults (controlled replays, not live execution). This criterion is structural (schema violations, missing pages, lost commits), independent of model output quality; human inspection confirmed zero-fault runs preserved state needed for correct completion. Table 8 shows the results: CLAWVM achieves 100% task success (zero explicit faults) at both budgets. Comp-Hybrid drops to 76.7% at budget 180; all 7 failures are bootstrap faults in debugging tasks where mid-task compaction evicts unprotected bootstrap pages (Table 7). At budget 300 the failures vanish—not because the policy improves, but because the larger budget avoids triggering compaction. This brittleness is the core issue: Comp-Hybrid’s success depends on having enough headroom to sidestep its structural gaps, and tighter budgets expose them. The identical thrash across all rows (1.36) is a workload floor: residual thrash at zero faults reflects inherent tool-signature repetition, not policy differences.

**Live hook validation.** Twenty single-session tasks against a production agent (Table 11) show 20/20 success for both policies with zero faults and comparable overhead

**Table 7.** Task success by type and policy at two budgets. All 7 Comp-Hybrid failures are bootstrap faults in debugging tasks.

Type	<i>n</i>	Comp-Hybrid		CLAWVM	
		180	300	180	300
coding	9	9/9	9/9	9/9	9/9
debugging	8	1/8	8/8	8/8	8/8
writing	2	2/2	2/2	2/2	2/2
ops	11	11/11	11/11	11/11	11/11
total	30	23/30	30/30	30/30	30/30

**Table 8.** Task-granularity replay: 30 synthetic tasks at two budgets. Success = zero explicit faults for the task.

Policy	Budget	Success	Faults	Tok/task	Thrash	$p_{50}$ ( $\mu$ s)
Comp-Hybrid	180	76.7%	0.23	1607	1.36	170
Comp-Hybrid	300	100.0%	0.00	2649	1.36	164
<b>CLAWVM</b>	<b>180</b>	<b>100.0%</b>	<b>0.00</b>	<b>1605</b>	<b>1.36</b>	<b>167</b>
<b>CLAWVM</b>	<b>300</b>	<b>100.0%</b>	<b>0.00</b>	<b>2639</b>	<b>1.36</b>	<b>166</b>

**Table 9.** Adversarial explicit faults by policy. Boot. = bootstrap, Pin = pinned-invariant-miss, Flush = flush-miss, D+R = duplicate-tool + refetch.

Scenario	Policy	Boot.	Pin	Flush	D+R	Total
starvation	Retrieval	0	10	0	0	10
	Comp-Hybrid	0	10	0	0	10
	<b>CLAWVM</b>	<b>0</b>	<b>10</b>	<b>0</b>	<b>0</b>	<b>10</b>
churn	Retrieval	9	0	9	280	298
	Comp-Hybrid	9	0	0	0	9
	<b>CLAWVM</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
cascade	Retrieval	4	0	13	60	77
	Comp-Hybrid	0	0	7	0	7
	<b>CLAWVM</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

(16.6 s vs. 16.9 s). The two policies tie because single-session tasks do not trigger the lifecycle boundaries (compaction, reset) where Comp-Hybrid’s structural gaps manifest.

We stress-test CLAWVM with three adversarial scenarios (Table 9): *budget starvation* (budget 40, three hard-pinned pages requiring 60 tokens total), *extreme churn* (50 unique evidence pages in 50 turns, budget 180), and *cascade reset* (9 resets in 30 turns with persistent dirty pages). Under churn and cascade resets, CLAWVM achieves zero faults where Retrieval incurs up to 298 and Comp-Hybrid up to 9. Under starvation, all policies incur 10 identical pinned-invariant-miss faults: when the budget physically cannot fit all hard-pinned pages, no policy can help, and CLAWVM surfaces this as a diagnosable failure rather than silent degradation.

The policy-engine decision procedure adds median 18–44  $\mu$ s per turn ( $p_{95} < 60 \mu$ s, worst-case 114  $\mu$ s), excluding model and tool latency, and < 83 KB peak memory.

**Takeaway.** CLAWVM generalizes to real traces, task-level workloads, and adversarial scenarios with zero policy-controllable faults and negligible overhead.

**Threats to validity.** Claims are scoped to structural lifecycle faults, not semantic correctness. Deterministic replay isolates policy behavior but not online effects (model nondeterminism, tool latency).

## 6 Related Work

CLAWVM’s contribution is *enforcement*: critical state survives lifecycle transitions, dirty state is committed before destruction, and paging decisions are observable.

**OS abstractions and transactional agents.** Several systems share overlapping abstractions but target different problems: AIOS [19] provides a multi-agent kernel with scheduling and context management; MemOS [13] unifies representation and evolution across heterogeneous memory types; SagaLLM [2] adds Saga-style transaction guarantees; Text2Mem [41] defines a typed memory operation language with schema-level invariants; and Memory OS [12] uses OS-inspired hierarchical storage with segmented page organization. CLAWVM composes enforcement primitives (lifecycle-complete commit, minimum-fidelity invariants, observable faults) into a contract at the harness level; it is a runtime layer that can wrap any of them.

**Agent memory systems.** CoALA [38] and Generative Agents [32] define frameworks for what memory an agent should have; CLAWVM addresses how the harness *enforces* that memory survives lifecycle transitions. MemGPT [30] is the closest predecessor, explicitly framing context management as OS-style virtual memory with model-driven paging. Memory-as-Action [47] similarly lets the model drive paging decisions. A-MEM [44] takes a different approach, organizing long-term memory as a Zettelkasten-like network with dynamic note construction and linking rather than prompt-time paging. These systems improve adaptivity but leave residency and writeback to model discretion; CLAWVM moves both to the harness with replayable enforcement. The two approaches are potentially composable: a MemGPT-style model could serve as the paging heuristic inside CLAWVM’s enforcement layer, gaining lifecycle-complete writeback and observable faults while retaining adaptivity. LLMLingua-2 [31] compresses prompts, RAG-Cache [11] caches retrieval results, and production layers (Mem0 [20], Cognee [4], QMD [16]) improve recall quality. All are orthogonal to enforcement and usable as CLAWVM components.

**Benchmarks.** AgentBench [15], WebArena [48], and OS-World [43] measure end-to-end task success but do not instrument paging or lifecycle failures. LoCoMo [17], Long-MemEval [42], AMemGym [3], Mem2ActBench [35], and

MemoryAgentBench [10] benchmark model memory ability across retrieval, learning, and forgetting. CLAWVM evaluates enforcement at the runtime level: lifecycle faults, commit correctness, and replayable selection.

## 7 Limitations

We identify three limitations that bound the scope of our claims.

**Semantic correctness is out of scope.** CLAWVM validates schema, provenance, scope, and non-destructiveness, but does not verify the semantic truth of model-generated updates. A model can produce a well-formed, properly scoped update that is factually wrong. Semantic verification could be layered on via domain-specific validation predicates on the writeback journal.

**Replay assumes simulatable tools.** The replay engine models tool calls by canonical signatures without executing real tools. Extending to non-deterministic services requires recording and replaying actual tool outputs, which the trace schema supports but the current prototype does not exercise.

**External validity.** Results cover replayed demand streams, 12 converted real-session traces, 30 synthetic task-level workloads, and 20 live single-session tasks. The live experiment confirms hook correctness but exercises only single-session execution; cross-session lifecycle edges (compaction, reset) remain replay-validated only. Multi-session online deployment with user-visible task-success metrics is future work.

## 8 Conclusion

Recurring memory failures in stateful tool-using agents are symptoms of a missing contract between the harness and the state it manages. CLAWVM provides the missing contract: typed pages, minimum-fidelity invariants, multi-resolution representations, and validated writeback at every lifecycle boundary. Across synthetic, real-trace, and adversarial workloads at all budget levels, CLAWVM eliminates all policy-controllable faults with negligible overhead; an offline oracle confirms zero remaining headroom on fault count. Fault elimination is robust by construction: an LRU baseline with the same structural features achieves identical fault counts, and utility scoring weights are insensitive across wide ranges, confirming that safety comes from the enforcement layer rather than heuristic tuning. The fault model and replay oracle are independently useful for evaluating alternative policies. Future work includes end-to-end evaluation with live models, multi-session deployment, and extension to multi-agent orchestrators.

**Artifact availability.** All source code, data, and scripts used in this paper are publicly available at <https://github.com/mp-dsg/clawvm>.

## References

- [1] AmbitiousRealism2025. 2026. Bug: Pre-compaction memory flush uses stale token counts, can be bypassed. GitHub issue #5457, openclaw/openclaw. <https://github.com/openclaw/openclaw/issues/5457> Accessed 2026-02-25.
- [2] Edward Y. Chang and Longling Geng. 2025. SagaLLM: Context Management, Validation, and Transaction Guarantees for Multi-Agent LLM Planning. *Proceedings of the VLDB Endowment* 18, 12 (2025), 4874–4886. doi:10.14778/3750601.3750611
- [3] Jiayang Cheng, Dongyu Ru, Lin Qiu, Yiyang Li, Xuezhi Cao, Yangqiu Song, and Xunliang Cai. 2026. AMemGym: Interactive Memory Benchmarking for Assistants in Long-Horizon Conversations. In *International Conference on Learning Representations (ICLR)*. <https://openreview.net/forum?id=sfrVLzsmf> ICLR 2026 poster; published 2026-01-26; last modified 2026-02-11..
- [4] Cognee. [n. d.]. OpenClaw. Cognee documentation (integration guide). <https://docs.cognee.ai/integrations/openclaw-integration> Cognee documentation page for the OpenClaw integration..
- [5] Peter J. Denning. 1968. The Working Set Model for Program Behavior. *Commun. ACM* 11, 5 (1968), 323–333. doi:10.1145/363095.363141
- [6] Dowser. 2026. Feature Request: Pre-prompt memory injection hook (message:before event). GitHub issue #11910, openclaw/openclaw. <https://github.com/openclaw/openclaw/issues/11910> Opened 2026-02-08; closed as not planned. Request for automatic memory injection before prompt processing..
- [7] EmberCF. 2026. [Bug] Pre-compaction memory flush prompt causes agents to overwrite existing memory files. GitHub issue #6877, openclaw/openclaw. <https://github.com/openclaw/openclaw/issues/6877> Accessed 2026-02-25.
- [8] gavlaahh. 2026. Your agent forgets everything after compaction. Here's a fix (open source, \$0.10/month). Reddit post (r/openclaw). [https://www.reddit.com/r/openclaw/comments/1r3nyro/your\\_agent\\_forgets\\_everything\\_after\\_compaction/](https://www.reddit.com/r/openclaw/comments/1r3nyro/your_agent_forgets_everything_after_compaction/) Posted 2026-02-13 (UTC)..
- [9] haymourt. 2026. OpenClaw “forgot” to run a protocol that we agreed it would. Reddit post (r/AI\_Agents). [https://www.reddit.com/r/AI\\_Agents/comments/1qxo8lw/openclaw\\_forgot\\_to\\_run\\_a\\_protocol\\_that\\_we\\_agreed/](https://www.reddit.com/r/AI_Agents/comments/1qxo8lw/openclaw_forgot_to_run_a_protocol_that_we_agreed/) Posted 2026-02-06 (UTC)..
- [10] Yuanzhe Hu, Yu Wang, and Julian McAuley. 2026. Evaluating Memory in LLM Agents via Incremental Multi-Turn Interactions. In *Proceedings of the Fourteenth International Conference on Learning Representations (ICLR)*. <https://openreview.net/forum?id=DT7JyQC3MR> ICLR 2026..
- [11] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. 2025. RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation. *ACM Transactions on Computer Systems* 44, 1, Article 2 (2025), 27 pages. doi:10.1145/3768628 Preprint available as arXiv:2404.12457..
- [12] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory OS of AI Agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. 25961–25970. doi:10.18653/v1/2025.emnlp-main.1318
- [13] Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Huayi Lai, Hao Wu, Bo Tang, Zhengren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofen Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. 2025. MemOS: A Memory OS for AI System. arXiv:2507.03724 [cs.CL] doi:10.48550/arXiv.2507.03724 Submitted 2025-07-04; last revised 2025-12-03 (v4)..
- [14] Limitless2023. 2026. fix(memory-flush): add softThresholdPercent for context-relative threshold. GitHub pull request #17041, openclaw/openclaw. <https://github.com/openclaw/openclaw/pull/17041> Accessed 2026-02-25.
- [15] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. AgentBench: Evaluating LLMs as Agents. In *The Twelfth International Conference on Learning Representations*. [https://proceedings.iclr.cc/paper\\_files/paper/2024/hash/e9df36b21ff4ee211a8b71ee8b7e9f57-Abstract-Conference.html](https://proceedings.iclr.cc/paper_files/paper/2024/hash/e9df36b21ff4ee211a8b71ee8b7e9f57-Abstract-Conference.html) Also available as arXiv:2308.03688..
- [16] Tobias Lütke. 2026. QMD - Query Markup Documents. GitHub repository. <https://github.com/tobi/qmd> Latest tagged release v1.0.7 dated 2026-02-18..
- [17] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating Very Long-Term Conversational Memory of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 13851–13870. doi:10.18653/v1/2024.acl-long.747 arXiv:2402.17753.
- [18] marcmezy. 2026. Memory flush uses stale token count, fires after compaction instead of before. GitHub issue #2397, openclaw/openclaw. <https://github.com/openclaw/openclaw/issues/2397> Opened 2026-01-26; closed. Issue describes token-count accounting lag (using a prior total token count) causing the flush to occur after compaction rather than before..
- [19] Kai Mei, Xi Zhu, Wujiang Xu, Mingyu Jin, Wenyue Hua, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. 2025. AIOS: LLM Agent Operating System. In *Second Conference on Language Modeling (COLM 2025)*. Montreal, Canada. <https://openreview.net/forum?id=L4HHkCDz2x> Also available as arXiv:2403.16971..
- [20] Mem0. [n. d.]. OpenClaw. Mem0 documentation (integration guide). <https://docs.mem0.ai/integrations/openclaw> Integration documentation; npm package: @mem0/openclaw-mem0..
- [21] mmartoccia. 2026. Feature: Workspace-aware post-compaction bootstrap to prevent amnesia. GitHub issue #20225, openclaw/openclaw. <https://github.com/openclaw/openclaw/issues/20225> Accessed 2026-02-25.
- [22] Moddy14. 2026. Memory flush softThresholdTokens doesn't scale with context window size. GitHub issue #17034, openclaw/openclaw. <https://github.com/openclaw/openclaw/issues/17034> Accessed 2026-02-25.

- [23] nickjlamb. 2026. feat: workspace-aware post-compaction context. GitHub pull request #20267, openclaw/openclaw. <https://github.com/openclaw/openclaw/pull/20267> Accessed 2026-02-25.
- [24] NullSense. 2026. [Feature]: Memory flush on /new and /reset (pre-reset memory save). GitHub issue #8185, openclaw/openclaw. <https://github.com/openclaw/openclaw/issues/8185> Accessed 2026-02-25.
- [25] OpenClaw. 2026. Compaction. OpenClaw Docs. <https://docs.openclaw.ai/concepts/compaction> Accessed 2026-02-25.
- [26] OpenClaw. 2026. Configuration Reference. OpenClaw Docs (Gateway & Ops). <https://docs.openclaw.ai/gateway/configuration-reference> Accessed 2026-02-25.
- [27] OpenClaw. 2026. Context. OpenClaw Docs. <https://docs.openclaw.ai/concepts/context> Accessed 2026-02-25.
- [28] OpenClaw. 2026. Memory Overview. OpenClaw Docs. <https://docs.openclaw.ai/concepts/memory> Accessed 2026-02-25.
- [29] OpenClaw. 2026. Session Management Deep Dive. OpenClaw Docs (Reference). <https://docs.openclaw.ai/reference/session-management-compaction> Accessed 2026-02-25.
- [30] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560 [cs.AI] doi:10.48550/arXiv.2310.08560 arXiv:2310.08560; submitted 2023-10-12; last revised 2024-02-12 (v2)..
- [31] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. LLMingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, Bangkok, Thailand, 963–981. doi:10.18653/v1/2024.findings-acl.57
- [32] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. Article 2, 22 pages. doi:10.1145/3586183.3606763 arXiv:2304.03442.
- [33] Qwibit.ai. 2026. NanoClaw. GitHub repository. <https://github.com/qwibitai/nanoclaw> Accessed 2026-02-25.
- [34] robdih. 2026. OpenClaw Best Practices: What Actually Works After Running It Daily. Reddit post (r/openclaw). [https://www.reddit.com/r/openclaw/comments/1r4t9q8/openclaw\\_best\\_practices\\_what\\_actually\\_works\\_after/](https://www.reddit.com/r/openclaw/comments/1r4t9q8/openclaw_best_practices_what_actually_works_after/) Posted 2026-02-14 (UTC)..
- [35] Yiting Shen, Kun Li, Wei Zhou, and Songlin Hu. 2026. Mem2ActBench: A Benchmark for Evaluating Long-Term Memory Utilization in Task-Oriented Autonomous Agents. arXiv:2601.19935 [cs.CL] doi:10.48550/arXiv.2601.19935 Submitted 2026-01-13..
- [36] Sipeed. 2026. PicoClaw. GitHub repository. <https://github.com/sipeed/picoclaw> Accessed 2026-02-25.
- [37] Skipper-Assistant-1968. 2026. QMD memory\_search silently returns empty in Discord guild channels (default scope denies chatType 'channel'). GitHub issue #10191, openclaw/openclaw. <https://github.com/openclaw/openclaw/issues/10191> Opened Feb 6, 2026; closed. Describes silent empty results when default scope denies chatType "channel".
- [38] Theodore R. Summers, Shunyu Yao, Karthik R. Narasimhan, and Thomas L. Griffiths. 2024. Cognitive Architectures for Language Agents. *Transactions on Machine Learning Research* (Feb. 2024). <https://openreview.net/forum?id=1i6ZCvfIQJ> Accepted by TMLR; published 2024-02-22; also available as arXiv:2309.02427..
- [39] theghostybot. 2026. Feature: Post-compaction system event injection for context continuity. GitHub issue #10524, openclaw/openclaw. <https://github.com/openclaw/openclaw/issues/10524> Accessed 2026-02-25.
- [40] These-Koala9672. 2026. How I solved context loss in long-running Claude agent sessions (OpenClaw). Reddit post (r/ClaudeAI). [https://www.reddit.com/r/ClaudeAI/comments/1r351ho/how\\_i\\_solved\\_context\\_loss\\_in\\_longrunning\\_claude/](https://www.reddit.com/r/ClaudeAI/comments/1r351ho/how_i_solved_context_loss_in_longrunning_claude/) Posted 2026-02-12 (UTC)..
- [41] Yi Wang, Lihai Yang, Boyu Chen, Gongyi Zou, Kerun Xu, Bo Tang, Feiyu Xiong, Siheng Chen, and Zhiyu Li. 2025. Text2Mem: A Unified Memory Operation Language for Memory Operating System. arXiv:2509.11145 [cs.CL] doi:10.48550/arXiv.2509.11145 Submitted 2025-09-14; last revised 2025-10-23 (v2)..
- [42] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. In *Proceedings of the Thirteenth International Conference on Learning Representations (ICLR)*. Singapore. [https://proceedings.iclr.cc/paper\\_files/paper/2025/hash/d813d324dbf0598bbdc9c8e79740ed01-Abstract-Conference.html](https://proceedings.iclr.cc/paper_files/paper/2025/hash/d813d324dbf0598bbdc9c8e79740ed01-Abstract-Conference.html) Poster. Also available as arXiv:2410.10813..
- [43] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems*. doi:10.52202/079017-1650 NeurIPS 2024, Datasets and Benchmarks Track; also available as arXiv:2404.07972..
- [44] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-Mem: Agentic Memory for LLM Agents. In *Advances in Neural Information Processing Systems*. <https://neurips.cc/virtual/2025/poster/119020> NeurIPS 2025 poster. OpenReview: <https://openreview.net/forum?id=FiM0M8gct>. Preprint: arXiv:2502.12110..
- [45] Deshraj Yadav. 2026. We Built Persistent Memory for OpenClaw (FKA Moltbot, ClawdBot) AI Agents. Mem0 blog post. <https://mem0.ai/blog/mem0-memory-for-openclaw> Posted Feb 6, 2026..
- [46] ZeroClaw Labs. 2026. ZeroClaw. GitHub repository. <https://github.com/zeroclaw-labs/zeroclaw> Accessed 2026-02-25.
- [47] Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. 2025. Memory as Action: Autonomous Context Curation for Long-Horizon Agentic Tasks. arXiv:2510.12635 [cs.AI] doi:10.48550/arXiv.2510.12635 Submitted 2025-10-14; last revised 2026-01-10 (v2)..

[48] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A Realistic Web Environment for Building Autonomous

Agents. In *The Twelfth International Conference on Learning Representations*. [https://proceedings.iclr.cc/paper\\_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html](https://proceedings.iclr.cc/paper_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html) Also available as arXiv:2307.13854..

## A Design Details

This appendix collects formal definitions behind the design choices in Section 3, provided for reproducibility and implementation reference.

**Utility scoring.** The greedy upgrade order in Listing 1 depends on a per-page utility function. For page  $p$ , representation  $r$ , and turn  $t$ , the utility is:

$$U_t(p, r) = w_{\text{pin}}I_{\text{pin}}(p) + w_{\text{boot}}I_{\text{boot}}(p) + w_{\text{plan}}I_{\text{plan}}(p, t) \\ + w_{\text{rec}}R_t(p) + w_{\text{scope}}S(p) + w_{\text{rc}}C(p, r),$$

where  $I_{\text{pin}}$ ,  $I_{\text{boot}}$ ,  $I_{\text{plan}}$  are indicator functions for pin status, bootstrap/policy membership, and active-plan membership;  $R_t$  is recency;  $S$  is scope utility; and  $C$  is recompute cost. Each candidate upgrade  $u$  is ranked by  $s(u) = \Delta U_t(u)/\Delta \text{tokens}(u)$ , with deterministic tie-breaks by (page\_id, rep\_level). In our experiments, the CLAWVM score uses pin boost (2.0 hard, 0.6 soft), recency weight 0.6, and recompute-cost weight 0.4; the recency baseline uses (0.9, 0.1). Oracle variants also weight each page by  $2.2 \times$  the number of future demands remaining in the trace.

**Thrash index.** Section 3 defines the thrash index informally; the precise formulation is:

$$\text{thrash} = \frac{F}{H + 1},$$

where  $F$  counts paging events (explicit faults plus duplicate-signature alerts) and  $H$  counts hits over the entire run;  $+1$  prevents division by zero. Because duplicate-signature alerts reflect inherent workload repetition (re-issued tool signatures when the result is already resident), thrash can be non-zero even when all policy-controllable faults are eliminated.

**Writeback protocol details.** Section 3 summarizes the three-phase writeback protocol; this appendix provides the full specification.

*Phase 1 (structured staging):* at eviction/downgrade boundaries or lifecycle hooks (flush/reset/compaction), the harness requests structured updates only: tuples of (field, op, value, scope, evidence\_ref) where ops are append/merge/set-with-version. These are appended to a `WritebackJournal` but not yet applied.

*Phase 2 (deterministic validation):* each staged update is checked against five invariants: (1) schema correctness (field exists, types check, size limits hold), (2) provenance (evidence\_ref resolves; dangling provenance rejected), (3) scope (requested scope allowed for the active principal/session), (4) non-destructive semantics (no overwrite of previously committed content), and (5) policy/safety (no violation of hard-pinned constraints). Rejected updates remain in the journal with reason codes (e.g., `SCOPE_DENIED`, `DESTRUCTIVE_OP`).

*Phase 3 (scoped commit):* validated updates commit into the appropriate scoped store using deterministic merge rules at explicit, logged commit points.

**Non-destructiveness rule.** For each staged update targeting key  $k$ , CLAWVM checks the journal’s last committed version  $v_k$ . Append/merge operations are valid only if they preserve prior committed entries for  $k$ . Set-with-version is valid only when the staged version equals  $v_k$ ; otherwise the update is rejected as `DESTRUCTIVE_OP`.

## B Supplementary Results

This appendix provides supplementary evaluation tables behind the claims in Section 5.

**Real-trace validation and scaling.** To test generalization beyond synthetic workloads, we converted 12 diverse real Claude Code session transcripts (coding, research, DevOps, trading, ML ops, web scraping, long-form writing, email writing, product design) into replay workloads. Table 10 shows aggregate fault counts at 100 and 200 turns, demonstrating how each policy scales with session length. CLAWVM maintains zero explicit faults at both lengths. Comp-Hybrid leaks exactly one flush-miss per trace at 100 turns; at 200 turns the flush-miss no longer appears in the median, though the structural gap (missing reset writeback) remains. Retrieval-only faults grow near-linearly, from a median of 51 at 100 turns to 83 at 200 turns.

**Table 10.** Aggregate explicit faults across 12 real traces at two session lengths (budget 300). Median and range reported.

Policy	Turns	Median (range)	Thrash
Retrieval	100	51 (49–77)	0.677
Retrieval	200	83 (71–109)	0.527
Comp-Hybrid	100	1 (1–1)	0.047
Comp-Hybrid	200	0 (0–0)	0.029
<b>CLAWVM</b>	<b>100</b>	<b>0 (0–0)</b>	<b>0.032</b>
<b>CLAWVM</b>	<b>200</b>	<b>0 (0–0)</b>	<b>0.032</b>

**Live hook validation.** Table 11 summarizes the live single-session validation.

**Table 11.** Live hook validation: 20 single-session tasks with a production agent. Both policies achieve identical success with comparable overhead.

Policy	Success	Faults	Mean time (s)
Comp-Hybrid	20/20	0	16.9
CLAWVM	20/20	0	16.6