

Activity Frames: Deterministic Compilation of Screen Capture into Episodic Memory for LLM Agents

Nossa Iyamu

Index Terms—LLM agents, episodic memory, context engineering, screen capture, Model Context Protocol, personal AI

Abstract—Large language model (LLM) agents can read code, search the web, and call tools, yet they remain blind to the one context stream that most defines a user’s day: what the user actually did on their computer. Continuous screen capture is now commodity technology, but it produces instants, not episodes; thousands of snapshot rows per day that an agent cannot reason over efficiently. Current consumption strategies either expose raw text search, which pushes segmentation onto the agent at high token cost, or summarize with an LLM, which is expensive, non-reproducible, and able to hallucinate. We present activity frames, a schema and compilation procedure that turns a local capture database into bounded, typed episodes of user activity using deterministic code alone. The schema enforces a two-tier contract: a measured tier containing only fields derivable mechanically from capture data, and an optional inferred tier where any interpretation must be namespaced, confidence-tagged, and evidence-linked. We release an open-source reference implementation with a built-in capture engine and a Model Context Protocol server. On a 61-day live corpus (109,735 frames, 214,360 input events), compiling a full day takes 68ms, output is byte-identical across runs, and the resulting context block uses 86× fewer tokens than the raw rows it replaces, at zero inference cost. In a downstream question-answering test against an independent oracle, an agent reading the compiled block answers correctly 98.4% of the time at both model tiers we test, whereas an LLM summary of the same capture yields 66.1% to 80.4% and mis-states activity durations by 25 to 136%. A stronger model narrows this gap but never closes it, and only the compiled block is deterministic across runs, fits the busiest day’s context window, and lets a cheaper model match a costlier one. We argue that deterministic compilation is the missing layer between capture and agent memory, and that separating measurement from inference is a prerequisite for trustworthy episodic memory.

I. Introduction

An LLM agent asked “what should I prioritize today?” answers from what it can see: the conversation, some files, perhaps a calendar. It cannot see that its user spent the morning inside a pull request, switched contexts forty times after lunch, or abandoned a draft email at 16:40. Position work has argued that episodic memory, a record of experience situated in time, is the missing piece for long-term LLM agents [1], and surveys of agent externalization treat memory as a first-class architectural concern [2]. Yet in practice, agent memory today means conversation memory: what the user told the model, not what the user did [3]–[5].

The raw material for behavioral episodic memory already exists. Automated time trackers have recorded application focus for a decade [6]; operating systems now ship continuous capture, with Microsoft Recall storing periodic snapshots on Copilot+ PCs [7]; and OpenAI’s Chronicle preview builds memories from recent screen content for its Codex assistant [8]. Capture, in other words, is commodity. Consumption is not. A day of event-driven capture on our corpus yields roughly two thousand snapshot rows, each one asserting only that at time t , application a displayed window w at URL u . Two consumption strategies dominate. Raw search hands the agent a flat result list and leaves sessionization, deduplication, and duration accounting to the model at inference time; we measure this cost at 126,812 tokens for a single day (Section VI). LLM summarization compresses well but imports the weaknesses of its summarizer: per-run cost, non-determinism, degraded reliability over long inputs [9], and the possibility of inventing activity that never happened. Neither yields memory an agent can cache, audit, or trust.

This paper proposes the missing middle: deterministic compilation. In the same way a compiler turns instructions into structured artifacts without opinion, we turn snapshot streams into activity frames: bounded episodes with an application and site, a start and end, dwell-based active time, typed page references, input volume, and evidence pointers back to the raw rows. No model participates. The same database and window always produce the same document, so episodic memory becomes free to rebuild, safe to cache, and mechanically auditable. Interpretation is not banned but quarantined: a second schema tier accepts inferred labels only when they are namespaced, confidence-tagged, and linked to the measured evidence that supports them.

Our contributions are:

- A two-tier schema for representing human computer activity to agents, separating measured fact from tagged inference, with coverage gaps and blind spots as mandatory document elements (Section III).
- Deterministic compilation rules (dwell crediting, session gaps, flicker merging, nearest-frame attribution, coordinate-based click resolution, total URL-to-entity typing) that require no learned components (Section IV).

- An open reference implementation: a dependency-free Python compiler with a provisioned local capture engine, a CLI, and a Model Context Protocol (MCP) server exposing four tools to any MCP-capable agent [10], [11] (Section V).
- An empirical characterization on a 61-day live corpus: token cost against raw rows ($86\times$ reduction for prompt-ready context), 68ms full-day compilation, byte-identical reproducibility, entity-typing coverage, and a duration distribution that separates genuine short attention bouts from single-snapshot transits and multi-monitor effects (Section VI).
- A downstream question-answering benchmark against an independent oracle, run at two model tiers, showing that an agent answers more accurately from the compiled block than from raw rows or an LLM summary of the same capture at both tiers, that the block lets a mid-tier model match a frontier one, and that a stronger model narrows but does not close the gap; the harness is released for rerunning against any model (Section VI-B).

II. Related Work

A. Memory Systems for LLM Agents

MemGPT introduced operating-system-style virtualization of context, paging conversation history in and out of a bounded window [3]. Production memory layers extract and consolidate salient facts from dialogue [4]; Zep organizes memory as a temporal knowledge graph over conversation and business data [5]; E-mem reconstructs episodic context from multi-agent execution traces [12]; and hierarchical procedural memory distills reusable skills from agent trajectories [13]. A recent review unifies these threads as externalization of memory, skills, and protocols [2]. These systems ingest what flowed through the agent: messages, tool calls, task traces. Studies of memory construction further show that granularity and structure materially affect downstream retrieval quality [14], supporting typed episodes over flat logs. Agent Workflow Memory induces reusable routines, but from agent rollouts rather than human activity [15].

B. Screen Capture as Agent Memory

A small 2025–2026 wave targets our exact goal with the opposite method. MIRIX attaches a multi-agent memory system to continuous screenshots, extracting six memory types with cloud model calls over accumulated images [16]. FOCAL filters on-device desktop capture and invokes a local vision-language model to write task-organized session summaries, reporting a 60% token reduction [17]. ProAgentBench collects 500+ hours of desktop capture for proactive-assistance evaluation and segments events on application switches before model-based annotation [18]. SummAct summarizes interaction traces into user intentions with an LLM [19], and OmniQuery augments captured personal media for question answering [20]. In every case a model sits inside the memory-construction loop, so

the memory inherits per-run cost, non-determinism, and the possibility of hallucinated episodes. Activity frames are the deterministic counterpoint: the compile path contains no model at all, output is byte-identical across runs, and interpretation is quarantined in a separate, evidence-linked tier. We regard this as the first deterministic desktop instantiation of the acquisition layer that episodic-memory advocates identify as missing [1].

C. Desktop Activity Understanding Before LLMs

Deriving task structure from interaction logs is a two-decade-old ambition. TaskTracer instrumented window focus, file, and clipboard events and associated them with user-declared tasks [21]; SWISH clustered windows into tasks from title semantics and switching patterns [22]. Interruption science established the fragmentation our compiler measures: knowledge workers average about three minutes per working-sphere event [23], and laptop screen content switches roughly every 19 seconds, with 75% of content segments viewed under one minute [24]; our Section VI distribution is consistent with these findings once capture mechanics are separated from behavior. Deterministic threshold-based segmentation of capture into episodes also has lifelogging precedent [25], and lifelogging’s central critique, that total capture without structure serves nobody [26], is the failure mode activity frames exist to prevent. On the methods side, our pipeline descends from web sessionization’s inactivity-timeout heuristics [27], event abstraction in process mining [28], and robotic process mining’s UI-log segmentation and frequent-routine identification [29]; deterministic data-to-episode compilation for faithfulness echoes the data-to-text tradition [30]. Those pipelines terminate in process models, automation scripts, or prose for humans. Ours terminates in typed episodic memory for an LLM agent, with the measured/inferred boundary as a portable schema contract rather than an internal implementation detail.

D. GUI Agents, Capture Systems, and Memory Trust

A large body of work gives agents screens as an action surface: GUI-agent surveys catalog systems that perceive interfaces via accessibility trees, OCR, and vision models to click, type, and navigate [31], [32], and generative agents showed that a remembered observation stream enables long-horizon behavior in simulation [33]. We compile the inverse direction: what the human did, for the agent to read; personal-agent surveys identify exactly this acquisition problem as open [34]. Deployed capture systems leave the same gap: ActivityWatch is content-blind by design [6], Recall exposes snapshots to the user but no agent API [7], and Chronicle feeds a single walled assistant via background model passes [8]. Separately, agent memory is emerging as an attack surface: stored memories can be poisoned to steer future behavior [35], and membership inference can probe what a store contains [36]. These results motivate two properties we make

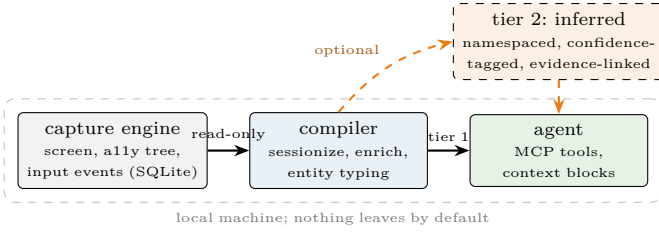


Fig. 1. The two-tier architecture. The measured tier (blue) is produced entirely by deterministic code reading the capture database. Interpretation (orange) is an optional extension that must be namespaced, confidence-tagged, and evidence-linked; stripping it always leaves a valid measured document.

structural: evidence pointers from every episode back to raw rows, and a hard boundary between measurement and inference, so a poisoned or hallucinated label can never masquerade as observed fact. Context-engineering surveys [37] and long-context degradation results [9] both argue for compact, structured context over raw dumps.

III. The Activity Frames Schema

A. Design Principles

Four principles fix the schema’s character. Measured, not guessed: every field in a standard document is derivable by deterministic code from capture data; there are no intent labels, because code cannot observe intent. Reproducible: identical inputs must yield identical documents, making memory cacheable and diffable. Evidenced: every frame carries pointers to the raw rows it was compiled from. Honest about absence: periods without capture are reported as gaps, and every document carries a `blind_spots` list stating what the pipeline systematically cannot see. Consumers must treat uncovered time as unknown, never as inactivity.

B. Document Structure

A document describes one query window and contains four parts: a coverage section (first and last activity, active minutes, span, capture gaps over five minutes), a chronological list of frames, a `blind_spots` list, and provenance metadata (schema_version, generation time, source recorder). A frame is one bounded stretch of attention in a single context, keyed by the pair (application, site), where site is the URL host for browser activity and absent otherwise. Figure 2 shows a representative frame.

C. Typed Page References

Browser frames carry pages: typed references produced by deterministic URL parsing. A reference has a kind (what sort of page), an optional entity (the human-relevant identifier), and a view count. Standard kinds include profile, company, people_search, search, repo, pull_request, issue, doc, email, video, post, ai_chat, and local_dev. The mapping is total: a URL matched by no site parser falls back to a generic page reference with its domain, so typing never loses data. Kinds are open for

```

- id: f-0007
  app: "Google Chrome"
  site: "linkedin.com"
  start: "20:24:04" end: "20:42:11"
  duration_min: 18.0 wall_min: 21.5
  pages:
    - {kind: people_search, entity: "cto paris", count: 2}
    - {kind: profile, entity: "jane-doe"}
    - {kind: company, entity: "acme"}
  input: {keys: 214, clicks: 31}
  interruptions: [{app: "Slack", seconds: 12}]
  evidence: {frame_ids: "99871..100147"}

```

Fig. 2. A single activity frame (YAML, entities anonymized). Every field is computed by code; the evidence pointer names the raw snapshot rows the frame was compiled from.

extension but must remain deterministic functions of the URL.

D. Tier 2: The Inferred Extension

Tools may add interpretation, such as task labels or project clusters, under three schema-enforced rules: inferred content lives in a namespaced inferred block, carries a confidence tag drawn from {high, medium, speculative}, and names its evidence: the measured fields or raw rows supporting it. A consumer can always strip the inferred block and be left with a purely measured document. The reference implementation emits tier 1 only; we consider the boundary itself, not any particular inference method, to be the contribution.

E. Privacy Rule

Input volume (keystroke, click, and copy counts) is part of the standard document. Input content (typed text) must be excluded by default and included only on an explicit operator opt-in. This is a schema requirement, not an implementation courtesy: a conforming producer cannot silently emit content.

IV. Deterministic Compilation

A. Setting

The capture engine is event-driven with a heartbeat: it stores a snapshot row on interaction and on screen change (clicks, application switches, visual changes), plus a periodic row after roughly 30 seconds without input. Heartbeat rows are 19% of our corpus; they matter because they keep the stream alive while the user reads, watches, or steps away from an awake display, and every downstream time measure inherits that. Each monitor records its own stream; within a stream the median inter-frame gap is 6.1s and the 90th percentile is 30.7s (Section VI). A row carries a timestamp, application, window title, and URL when the focused application is a browser. Input events (keystrokes, clicks, clipboard, application switches) arrive in a parallel stream. Compilation consumes these streams for a query window, segmenting each monitor independently, and emits the document of Section III.

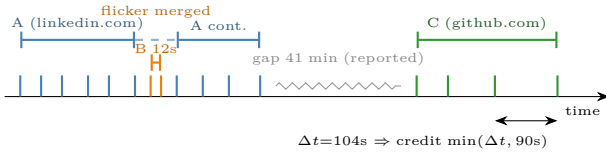


Fig. 3. Segmentation on one timeline. Ticks are snapshot rows. A 12-second detour (B) folds into the surrounding frame as a recorded interruption; a long silence becomes a reported coverage gap; sparse snapshots earn at most the 90s dwell cap.

B. Sessionization

Three constants govern segmentation, chosen from the capture cadence rather than tuned on outcomes. Dwell: a frame contributes $\min(\Delta t, 90\text{s})$ of active time, where Δt is the gap to the next frame in its monitor’s stream. Because the engine emits a $\sim 30\text{s}$ heartbeat during input-free stretches, dwell measures screen tenure: how long a context stayed frontmost on an awake display. That includes reading and watching, which produce no input, but also stretches where the user has stepped away while the display stays on; Section VII quantifies this. The cap, roughly three heartbeat periods, bounds the credit a single frame can earn when capture stalls or the display sleeps; it does not, and cannot, distinguish attention from presence. Consumers who need interaction-gated time can compare each frame’s reported input volume against its duration. Segmentation runs per monitor, so a context visible on two monitors at once earns tenure on both; the schema discloses this as a blind spot and Section VI-B shows the consequence. Session gap: a gap above 300s closes the current frame and becomes a candidate coverage gap; no dwell is credited across it. Flicker merge: the pattern $A \rightarrow B \rightarrow A$, where B lasts at most 20s of wall time and no session gap intervenes, collapses into a single A frame. Crucially, B is not discarded: it is recorded on the merged frame as an interruption with its measured seconds, and its time is not added to A ’s active duration. Nothing is hidden and nothing is double-counted. Algorithm 1 summarizes the procedure; Figure 3 illustrates all three rules on one timeline.

C. Enrichment

Raw input events have three reliability defects that code can repair. First, stale attribution: the recorder sometimes tags an event with the previously focused application. Each event is therefore re-attributed to the temporally nearest snapshot (binary search over the frame stream), whose application, window, and URL are authoritative; the attribution distance is kept in milliseconds so downstream consumers can judge it. Second, anonymous clicks: many click events carry coordinates but no element name. We resolve them against the snapshot’s recorded element tree: exact containment first (smallest containing element wins), then a $\pm 40\text{px}$ tolerance ring, then a coarse screen-zone fallback; every resolution is tagged exact, tolerance,

Algorithm 1 Frame segmentation (one pass, then flicker merge)

Require: snapshots $f_1 \dots f_n$ of one monitor’s stream, sorted by time; constants $D=90$, $G=300$, $F=20$ (seconds)

```

1:  $S \leftarrow []$ ;  $cur \leftarrow \perp$ 
2: for  $i = 1$  to  $n$  do
3:    $k_i \leftarrow (\text{app}(f_i), \text{site}(f_i))$ ;  $\Delta \leftarrow t(f_{i+1}) - t(f_i)$ 
4:   if  $cur = \perp$  or  $k_i \neq \text{key}(cur)$  then
5:     append new segment  $cur$  with key  $k_i$  to  $S$ 
6:   end if
7:   extend  $cur$  with  $f_i$ 
8:   if  $\Delta \leq G$  then
9:      $\text{active}(cur) += \min(\Delta, D)$ 
10:  else
11:     $cur \leftarrow \perp$  {session break; candidate gap}
12:  end if
13: end for
14: for consecutive  $A, B, A'$  in  $S$  with  $\text{key}(A) = \text{key}(A')$  do
15:   if  $\text{wall}(B) \leq F$  and no session break around  $B$  then
16:     merge  $A'$  into  $A$ ; record  $B$  as interruption of  $A$ 
17:   end if
18: end for
19: return  $S$ 
```

or zone, and unresolvable clicks stay unresolved rather than being guessed. Third, keyboard-layout mismatch: some capture stacks record physical key positions decoded as QWERTY while the user types another layout. An explicit, operator-supplied translation map repairs this; it is identity by default and never inferred.

D. Entity Typing

Site parsers map URLs to the typed references of Section III: path and query parsing only, no fetching, no models. Resolution proceeds in layers: a bespoke parser for the host (the reference implementation ships more than twenty, covering professional networks, code hosting, search, documents, mail, maps, video, social, events, dashboards, AI chat, and local development), then a generic search-parameter detector, then a subdomain-and-path heuristic that types common infrastructure pages (sign-in, dashboard, email, calendar, meeting) even for sites without a bespoke parser, and finally a total fallback that guarantees every URL maps to something. Because every layer is a pure function of the URL, adding coverage is a contribution reviewable line by line.

V. Reference Implementation

The reference implementation [10] is a Python package (MIT license) with three parts. The capture engine is provisioned on demand by aframes record: a pinned, MIT-licensed, open-source build that records application focus, window titles, URLs, the accessibility tree, and input events into a local SQLite database, entirely on-device,

TABLE I
MCP tool surface of the reference implementation.

Tool	Returns
get_context	Compact chronological context block for the last N hours, sized for inclusion in a system prompt
get_activity	Full schema-v1 document (JSON) for a day or window
get_day_summary	Coverage plus per-app ledger (minutes, sessions, longest session)
get_patterns	Repetitive workflows over the last N days

TABLE II
Live capture corpus used throughout Section VI.

Calendar span	61 days
Days with activity	46
Snapshot rows (frames)	109,735
with URL	59,458
idle-heartbeat rows	20,429 (19%)
Input events	214,360
Element-tree rows	8,395,885
Median intra-monitor gap	6.1 s
90th-percentile gap	30.7 s
Distinct applications	54

with audio capture off by default. Operators already running a compatible recorder can skip it and point the compiler at any existing database. The compiler has zero runtime dependencies, opens the database read-only, and exposes the document builders plus emitters for JSON, YAML, Markdown, and a compact plaintext context block designed for system prompts. The MCP server [11] is a hand-rolled stdio JSON-RPC implementation, also dependency-free, exposing the four tools of Table I to any MCP client. A workflow-pattern detector (repeated clicks, URL loops, action sequences, application-switching habits, daily habits) rounds out the surface.

VI. Empirical Characterization

We characterize the system on the author’s own live corpus: 61 calendar days of capture (46 days with activity) comprising 109,735 snapshot rows, 214,360 input events, and 8.4M element-tree rows across 54 applications (Table II). The corpus is frozen: the recorder was migrated to a new database after the last captured day, so every number below is reproducible against an immutable file. This is a single-user corpus; we report it as a characterization of the mechanism, not a user study (Section VII).

A. Token Cost

For one representative full day (2,066 snapshot rows), we compare three representations an agent could receive, tokenized with the `cl100k_base` encoding. The raw rows, serialized as the JSON a search API would return, cost 126,812 tokens. The compiled schema-v1 document (222 frames at a 0.5-minute floor) costs 34,815 tokens, a $3.6\times$ reduction that also relieves the agent of segmentation work. The compact context block costs 1,469 tokens, an

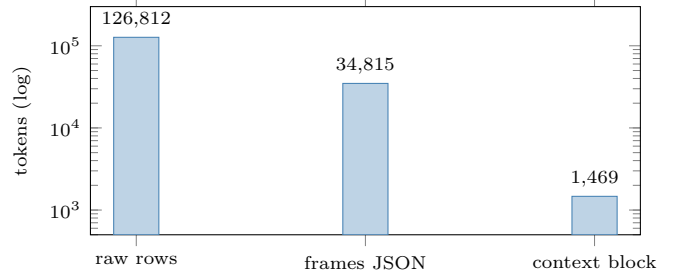


Fig. 4. Token cost of one full day under three representations (`cl100k_base`). Deterministic compilation yields an $86\times$ reduction for prompt-ready context, at zero inference cost.

$86\times$ reduction, small enough to include in every system prompt (Figure 4). Both compiled forms are produced without any model call, so the reduction is free, and long-context results suggest the smaller representation is not merely cheaper but better used by the model [9].

B. Downstream Question Answering

Token cost is a means; the end is whether an agent can answer questions about the user’s day. We test this directly. We evaluate on eight days chosen by a fixed rule: the seven most recent consecutive active days plus the nearest preceding day whose raw serialization overflows the model’s context window (included to exercise that regime). For each day, ground-truth answers are computed by an independent SQL oracle over the raw tables: a documented inactivity-timeout dwell (credit each frame the gap to the next, capped at 60s), deliberately not the activity-frames compiler, so the reference is not circular. Oracle and compiler measure the same construct, screen tenure: the capture heartbeat (Section IV) keeps input-free stretches credited, so “active minutes” throughout this section means time a context was frontmost on an awake display, not interaction time. Every representation is graded against that one construct, which keeps the comparison fair, but absolute magnitudes should be read as tenure. We validate the compiler against the oracle before using the oracle to grade anyone. On the seven benchmark days whose capture was complete when the oracle was frozen, the compiler’s covered active minutes agree with the oracle’s total dwell to a median of 0.9 minutes (all within 2.3); the eighth day’s capture continued after the freeze, so it is compared only at its snapshot. Distinct-application counts agree exactly or within one on every day. Two caveats keep this agreement honest. First, it compares aggregates that both approximate covered wall time, so it validates coverage rather than per-application arithmetic. Second, the two systems intentionally differ on multi-monitor days: the compiler credits each monitor’s stream (Section IV), while the oracle interleaves all monitors into one. On the context-overflow day, which has 133 dual-monitor minutes, the dominant application earns 435.8 minutes by the compiler’s ledger but 346.1 by

TABLE III

Downstream QA across two model tiers (8 days, 64 ground-truth questions, graded against the independent SQL oracle). “Acc.” is overall accuracy; “Dur. err.” is the mean absolute error on the dominant application’s active minutes. The compiled block scores identically at both tiers, so a mid-tier model reads it as well as a frontier one; the raw-row and summary baselines improve with model strength but never catch it. Only the block is deterministic across runs and fits the context window on every day; on the busiest day the raw rows are 257k tokens, so both baselines are infeasible and report on 7 of 8 days (the block, on all 8).

Representation	Sonnet 4.5		Opus 4.5	
	Acc.	Dur. err.	Acc.	Dur. err.
Raw rows	82.1%	39.7%	91.1%	25.7%
LLM summary	66.1%	135.7%	80.4%	25.2%
Activity frames	98.4%	7.3%	98.4%	7.3%

the oracle, a 26% divergence that the grading tolerance below happens to contain; switching the oracle’s cap from 60 to 90s moves its day total by under 6 minutes, so the divergence is the monitor convention, not the dwell cap. We report this rather than average over it. The generator emits 64 questions across five categories: which application dominated, how many active minutes in it, how many distinct applications, pairwise time ranking, whether a given domain was visited, and starting time, plus 16 absent-fact probes (“did the user open Photoshop / visit netflix.com?”) that a faithful system answers negatively. We deliberately exclude two further question types. Longest-session length depends on the session-gap convention rather than a fact, and per-profile recall exceeds the compact block’s token budget on busy days; each would penalize the compact block for a convention or a compression choice rather than for faithfulness, so we record the exclusion here to keep it on the books.

Two agents at different capability tiers, Claude Sonnet 4.5 and the larger Opus 4.5, each given no tools and answering only from the supplied text, answer every question from each of three representations: the raw rows, an LLM-generated summary of those rows written by that same agent, and the activity-frames output (the deterministic per-application ledger plus the compact context block, together ≈ 2 k tokens). We note plainly that this hands the activity-frames agent measured per-application durations the other two must derive for themselves: that is exactly what the compiler is for, but it means the quantitative questions test whether the deterministic arithmetic was already done, not only whether the agent can read. Answers are graded against the oracle with fixed tolerances (numeric within 30%; times within 45 minutes), and the ranking does not depend on them: at the Sonnet tier, a strict 10%/15-minute band leaves the three representations at 95.3%, 80.4%, and 66.1%, and the summary is so far off that widening the band to 50% lifts it only to 67.9%. Table III reports both tiers at the default tolerance.

Four findings. First, the gap is quantitative, not categorical. At both tiers all three representations score perfectly on the absent-fact probes: the models do not invent applications or websites (hallucination rate 0% throughout). The summary fails instead on magnitudes. At the Sonnet tier it answers time questions at 7.1% accuracy and mis-states the dominant application’s active minutes by 135.7% on average, against 7.3% for activity frames (that residual is dwell-method rounding, not error). Fluent prose hides the damage: on 2026-07-05 the measured record (the oracle, with the compiler’s ledger agreeing) is Google Chrome first at 161.6 dwell minutes and Cursor second at 143.9, over a day that ran 11:40 to 22:26, yet a Sonnet summary of that day names Cursor the primary application at “ ~ 7 hours,” inflating its 144 measured minutes by $2.9\times$ and fabricating a nocturnal “6:40 PM–5:26 AM” session that spills into the next day. The activity-frames agent answered 160 minutes. Every day, summary, and graded answer is in the released harness output.

Second, a stronger model narrows the gap but does not close it, and the compiled block erases the difference between the two. The frontier Opus model reconstructs durations far better from prose: its summary reaches 80.4% overall and cuts the duration error to 25.2%. But that is still $3.4\times$ the block’s 7.3%, and reading the block the two models are indistinguishable (98.4% each), whereas reading raw rows or a summary the frontier model runs 9 to 14 points ahead of the mid-tier one. The block lets the cheaper model answer as well as the expensive one; the baselines make capability matter. The benefit of deterministic compilation is therefore largest exactly where compute is cheapest to deploy.

Third, the summary is non-deterministic: regenerating it three times per day produced three distinct texts on every day at both tiers, whereas the activity-frames block was byte-identical across three regenerations, so the summary’s errors are not even stable enough to correct for. Fourth, the baselines do not always run. The busiest day serializes to 257k raw tokens, exceeding the context window, so both raw-row and summary consumption are infeasible; the ≈ 2 k block is unaffected, which is why it alone reports on all eight days. Activity frames are not flawless: their one miss (98.4%, not 100, at both tiers) is a domain-recall question on the busiest evaluated day, where the compact block’s budget drops a visited domain; that is the expected price of a bounded representation, and a recall question rather than a magnitude one. These accuracies are a single answering pass per representation per tier; we release the full harness (oracle, question generator, and grader) so they can be repeated, with confidence intervals and further models, on any corpus.

C. Latency and Reproducibility

Compiling the same full day end to end (segmentation, enrichment-backed input accounting, entity typing,

emission) takes a median of 68 ms over five runs on a consumer laptop (Apple Silicon), with a 65 to 72 ms range. Episodic memory at this cost can simply be rebuilt on every query. Reproducibility holds by construction and we verify it empirically: two independent compilations of the same window produce byte-identical documents once the generation timestamp is excluded. Determinism is what makes the memory cacheable, diffable across code versions, and testable in continuous integration.

D. Entity Typing Coverage

Across all 5,120 distinct URLs in the corpus, the layered parsers produce a non-generic typed reference for 81.3%, spanning 46 kinds; the remaining 18.7% fall back to the generic page reference with domain. The most frequent kinds are profile (1,106 distinct URLs), search (397), email (278), event (209), and messaging (200), reflecting that typed coverage concentrates precisely on the high-signal pages an agent most benefits from resolving. Coverage grows one pure function per site, so the long tail is closed incrementally by contribution rather than by any learned component.

E. How Fragmented Is a Day?

Compiling all 43 active days yields 17,514 frames (median 361 per day) with a median active duration of 0.5 minutes: 68% of frames last under one minute (Figure 5). A number this stark demands decomposition before interpretation, because two mechanical effects sit inside it. First, 52% of the sub-minute frames (6,177) contain a single snapshot, with a median credit of 6.1 seconds: these are transits, the application an operator passes through on the way somewhere else, faithfully recorded but not attention episodes. Second, a further 20% (2,361) lie inside dual-monitor stretches, where each monitor earns its own stream by construction; 27% of captured minutes on this corpus have two monitors active. Restricting to frames with at least two snapshots, the median rises to 0.9 minutes and 52% remain under one minute; that residue is genuine switching. Read this way, the distribution is consistent with what interruption science has measured with dedicated instrumentation: three-minute working spheres [23] and 75% of laptop content segments under one minute [24]. We deliberately do not present the raw histogram as an independent replication: an earlier draft of this analysis segmented all monitors as one interleaved stream, which shreds dual-monitor stretches and pushed the median to 0.3 minutes, and strict segmentation always overstates fragmentation until transits are separated out. The lesson is the schema’s, not just ours: consumers get a `min_minutes` floor (frames below it are omitted with the omission disclosed, never silently merged), the flicker-merge rule keeps sub-20-second detours from shredding genuine focus blocks while still recording them, and single-snapshot transits remain visible in the document precisely

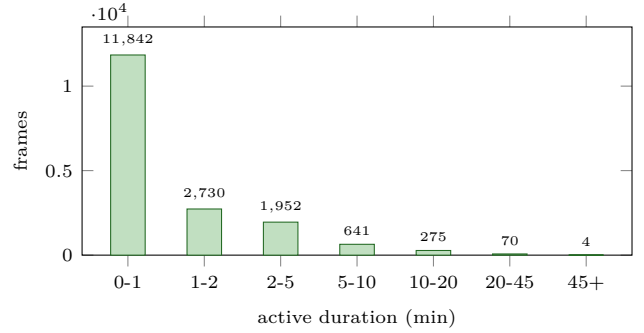


Fig. 5. Active-duration distribution of all 17,514 frames across 43 days (per-monitor segmentation). Median 0.5 min, but 52% of the sub-minute bar is single-snapshot transits rather than attention episodes; excluding them the median is 0.9 min (see text).

so that no compiler-side heuristic has to decide what counts as attention.

VII. Privacy, Trust, and Limitations

A. Privacy Model

The entire pipeline is local: capture, storage, and compilation happen on the user’s machine, the compiler opens the capture database read-only, and nothing is transmitted anywhere by the system itself. The operator chooses what leaves, and when, by handing a compiled artifact to an agent. The schema-level content rule (Section III) keeps typed text out of documents by default; audio capture is off by default in the provisioned engine. Compilation itself involves no model of any kind; the capture engine does run on-device OCR to read screen content, but that output never leaves the machine, and no language model, local or remote, participates in producing memory. The capture database itself remains sensitive at rest, as membership-inference work on memory stores reminds us [36]; we treat device-level encryption as the appropriate control and note that this risk is shared by every capture system rather than introduced by compilation.

B. Trust Properties

Two structural properties address emerging attacks on agent memory. Evidence pointers make every episode mechanically auditable: a verifier can re-read the named raw rows and recompute the frame, which raises the bar for poisoning attacks that rely on unverifiable memories [35]. The measured/inferred boundary ensures that even a compromised or careless tier-2 tool cannot inject interpretation disguised as observation; consumers can always strip to the measured core. We do not claim these properties defeat a compromised capture engine, which can fabricate raw rows; provenance begins at the database.

C. Limitations

Four limitations bound our claims. First, the empirical characterization is a single-user corpus (one professional, one machine, 61 days); cadence, fragmentation, and entity

coverage will differ across roles and platforms, and a multi-user study is future work. Second, the reference implementation reads one capture engine’s database layout; the schema is engine-agnostic but each new source needs an adapter. Third, the measured tier is structurally silent on intent: it reports two profile views and a people search, never “prospecting,” and consumers who need intent must add tier-2 inference and accept its tags. Fourth, screen presence is an imperfect proxy for attention, and on this corpus the gap is quantifiable rather than hypothetical. The capture heartbeat keeps input-free stretches credited: intervals that terminate in a heartbeat row carry 45% of all credited active time, and uninterrupted heartbeat runs reach 42 minutes, so dwell includes reading and watching but also time the user had stepped away from an awake display. The dwell cap does not bound this; it only bounds credit across capture stalls and display sleep. Frames report input volume precisely so consumers can gate on interaction, and a tier-2 tool may label presence-only stretches, but the measured tier reports tenure, not attention, and consumers must read it that way. Monitors compound the proxy error in the other direction: each records its own stream, so a minute with two active monitors earns credit twice in per-application ledgers (27% of captured minutes here; Section VI-B). Off-screen work (paper, phone, conversation) appears only as gaps. Relatedly, frames measure attention episodes, not tasks: interruption research shows interrupted tasks are resumed across much longer horizons than any single frame [23], so task-level structure belongs to tier-2 inference. Finally, the downstream benchmark (Section VI-B) evaluates two agent tiers (Claude Sonnet 4.5 and Opus 4.5) on this single-user corpus with one answering pass each. A stronger model narrows the summary’s gap, so the magnitude of the effect is not model-independent; but the block’s determinism, its context-fit, and its being read equally well by both tiers are structural properties no model strength supplies, and we release the harness so the comparison can be rerun with confidence intervals, against other model families such as GPT and Gemini, and, as multi-user capture becomes available, other users.

VIII. Conclusion

Agents are blind to the activity stream that most defines their user’s day, not because capture is missing but because nothing turns capture into memory an agent can afford, reproduce, and trust. Activity frames fill that seam with the least interesting tool available, deterministic code, and we argue that this dullness is the point: at 68ms and zero tokens per day, episodic memory becomes infrastructure rather than inference, and the measured/inferred boundary gives interpretation a place to live without contaminating fact. The schema, compilation rules, and implementation are open [10]; we hope the format outlives the reference code, and that capture systems, memory layers, and agents converge on

a shared, honest representation of what a person actually did.

References

- [1] M. Pink, Q. Wu, V. A. Vo, J. Turek, J. Mu, A. Huth, and M. Toneva, “Position: Episodic memory is the missing piece for long-term LLM agents,” 2025.
- [2] C. Zhou, H. Chai, W. Chen, Z. Guo, R. Shan, Y. Song, T. Xu, Y. Yang, A. Yu, W. Zhang, C. Zheng, J. Zhu, Z. Zheng, Z. Zhang, X. Lou, C. Zhang, Z. Fu, J. Wang, W. Liu, J. Lin, and W. Zhang, “Externalization in LLM agents: A unified review of memory, skills, protocols and harness engineering,” 2026.
- [3] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez, “MemGPT: Towards LLMs as operating systems,” 2024.
- [4] P. Chhikara, D. Khant, S. Aryan, T. Singh, and D. Yadav, “Mem0: Building production-ready AI agents with scalable long-term memory,” 2025.
- [5] P. Rasmussen, P. Paliychuk, T. Beauvais, J. Ryan, and D. Chalef, “Zep: A temporal knowledge graph architecture for agent memory,” 2025.
- [6] E. Bjäreholt and J. Nilsson, “ActivityWatch: open-source automated time tracker.” <https://activitywatch.net>, 2016. Accessed 2026-07-04.
- [7] Microsoft, “Recall: retrace your steps on Copilot+ PCs.” <https://learn.microsoft.com/en-us/windows/apps/develop/windows-integration/recall/>, 2025. Accessed 2026-07-04.
- [8] OpenAI, “Chronicle: memories from recent screen content in Codex for macOS.” <https://developers.openai.com/codex/memories/chronicle>, 2026. Research preview, accessed 2026-07-04.
- [9] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, “Lost in the middle: How language models use long contexts,” 2023.
- [10] N. Iyamu, “activity-frames: episodic memory for AI agents.” <https://github.com/nossa-y/activity-frames>, 2026. MIT license, accessed 2026-07-04.
- [11] Anthropic, “Model Context Protocol.” <https://modelcontextprotocol.io>, 2024. Open protocol specification, accessed 2026-07-04.
- [12] K. Wang, Y. Lin, J. Lou, Z. Zhou, B. Suvonov, and J. Li, “E-mem: Multi-agent based episodic context reconstruction for LLM agent memory,” 2026.
- [13] S. Forouzandeh, W. Peng, P. Moradi, X. Yu, and M. Jalili, “Learning hierarchical procedural memory for LLM agents through bayesian selection and contrastive refinement,” 2025.
- [14] Z. Pan, Q. Wu, H. Jiang, X. Luo, H. Cheng, D. Li, Y. Yang, C.-Y. Lin, H. V. Zhao, L. Qiu, and J. Gao, “On memory construction and retrieval for personalized conversational agents,” 2025.
- [15] Z. Z. Wang, J. Mao, D. Fried, and G. Neubig, “Agent workflow memory,” 2024.
- [16] Y. Wang and X. Chen, “Mirix: Multi-agent memory system for llm-based agents,” 2025.
- [17] H. Yin, Z. Wen, J. Cao, B. Yuan, and R. Yang, “Focal: Filtered on-device continuous activity logging for efficient personal desktop summarization,” 2026.
- [18] Y. Tang, H. Tang, T. Cao, L. Nguyen, A. Zhang, X. Cao, C. Liu, W. Ding, and Y. Li, “Proagentbench: Evaluating llm agents for proactive assistance with real-world data,” 2026.
- [19] G. Zhang, M. Ahmed, Z. Hu, and A. Bulling, “Summact: Uncovering user intentions through interactive behaviour summarisation,” 2024.
- [20] J. N. Li, Z. J. Zhang, and J. Ma, “Omniquery: Contextually augmenting captured multimodal memory to enable personal question answering,” 2025.
- [21] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker, “TaskTracer: A desktop environment to support multi-tasking knowledge workers,” in *Proc. IUI*, pp. 75–82, 2005.
- [22] N. Oliver, G. Smith, C. Thakkar, and A. C. Surendran, “SWISH: Semantic analysis of window titles and switching history,” in *Proc. IUI*, pp. 92–99, 2006.

- [23] V. M. González and G. Mark, ““constant, constant, multitasking craziness”: Managing multiple working spheres,” in *Proc. CHI*, pp. 113–120, 2004.
- [24] L. Yeykelis, J. J. Cummings, and B. Reeves, “Multitasking on a single device: Arousal and the frequency, anticipation, and prediction of switching between media content on a computer,” *Journal of Communication*, vol. 64, no. 1, pp. 167–192, 2014.
- [25] A. R. Doherty and A. F. Smeaton, “Automatically segmenting LifeLog data into events,” in *Proc. WIAMIS*, pp. 20–23, 2008.
- [26] A. J. Sellen and S. Whittaker, “Beyond total capture: A constructive critique of lifelogging,” *Communications of the ACM*, vol. 53, no. 5, pp. 70–77, 2010.
- [27] R. Cooley, B. Mobasher, and J. Srivastava, “Data preparation for mining world wide web browsing patterns,” *Knowledge and Information Systems*, vol. 1, no. 1, pp. 5–32, 1999.
- [28] S. J. van Zelst, F. Mannhardt, M. de Leoni, and A. Koschmider, “Event abstraction in process mining: Literature review and taxonomy,” *Granular Computing*, vol. 6, pp. 719–736, 2021.
- [29] V. Leno, A. Polyvyanyy, M. Dumas, M. La Rosa, and F. M. Maggi, “Robotic process mining: Vision and challenges,” *Business & Information Systems Engineering*, vol. 63, no. 3, pp. 301–314, 2021.
- [30] F. Portet, E. Reiter, A. Gatt, J. Hunter, S. Sripada, Y. Freer, and C. Sykes, “Automatic generation of textual summaries from neonatal intensive care data,” *Artificial Intelligence*, vol. 173, no. 7–8, pp. 789–816, 2009.
- [31] C. Zhang, S. He, J. Qian, B. Li, L. Li, S. Qin, Y. Kang, M. Ma, G. Liu, Q. Lin, S. Rajmohan, D. Zhang, and Q. Zhang, “Large language model-brained GUI agents: A survey,” 2025.
- [32] G. Liu, P. Zhao, Y. Liang, L. Liu, Y. Guo, H. Xiao, W. Lin, Y. Chai, Y. Han, S. Ren, H. Wang, X. Liang, W. Wang, T. Wu, Z. Lu, S. Chen, LiLinghao, H. Wang, G. Xiong, Y. Liu, and H. Li, “LLM-powered GUI agents in phone automation: Surveying progress and prospects,” 2025.
- [33] J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” 2023.
- [34] Y. Li, H. Wen, W. Wang, X. Li, Y. Yuan, G. Liu, J. Liu, W. Xu, X. Wang, Y. Sun, R. Kong, Y. Wang, H. Geng, J. Luan, X. Jin, Z. Ye, G. Xiong, F. Zhang, X. Li, M. Xu, Z. Li, P. Li, Y. Liu, Y.-Q. Zhang, and Y. Liu, “Personal LLM agents: Insights and survey about the capability, efficiency and security,” 2024.
- [35] Y. Louck, “Securing LLM-agent long-term memory against poisoning: Non-malleable, origin-bound authority with machine-checked guarantees,” 2026.
- [36] K. Chen, Y. Pang, and T. Wang, “MRMMIA: Membership inference attacks on memory in chat agents,” 2026.
- [37] L. Mei, J. Yao, Y. Ge, Y. Wang, B. Bi, Y. Cai, J. Liu, M. Li, Z.-Z. Li, D. Zhang, C. Zhou, J. Mao, T. Xia, J. Guo, and S. Liu, “A survey of context engineering for large language models,” 2025.