Context, Memory, and Governed Data

Part V of *The End of Software Engineering and the Rise of Agentic Engineering*

Matthew Long
The YonedaAI Collaboration, YonedaAI Research Collective
Chicago, IL
matthew@yonedaai.com

September 2026

Abstract

A context window is a bounded token sequence whose accuracy falls with length and varies with position, and an agent's reliable task length is bounded. Two questions follow: what infrastructure those facts require underneath agentic engineering, and what the engineer's role becomes. Five objects are defined: context engineering, persistent memory, the governed data layer, provenance per field, and refusal over plausible error. Persistent memory is a fold over an immutable event history, with retrieval a separate selection under a token budget. A governed data layer evaluates a typed query at a boundary before any source is read, and its refusals are usable only when emptiness and failure are structurally distinct. Four first-party code bases are examined at pinned commits with their documented implementation limits. One merges zero of 2,224 known-true record pairs while refusing correctly, pays 52% of query time for provenance, and records twelve internal defects of the class it exists to prevent. Another validates on ingestion and applies no token budget on retrieval. Of 13 models claiming at least 128K context, 11 fall below half their short-context baseline at 32K. An independent audit finds 6.4% of a widely used memory benchmark's answer key wrong. Of the packages recommended by 16 code-generating models, 19.7% do not exist, and 43% of the fabricated names recur on all ten repeats. These results locate the agentic engineer's control-layer responsibility: allocate context, preserve state, constrain data access, retain provenance, and require typed refusal. The four code bases share one author and form one lineage rather than independent confirmation.

1 Introduction

1.1 The research question

Parts I to IV described a change in what an engineer does and the machinery that change runs on. Part I established the central claim and defined the practice, Part II gave the primitives and the layered architecture, and Parts III and IV gave the loop and the team.

Each of those Parts assumed three things about the layer beneath it. An agent can be given the right material to work with. What it learns in one session reaches the next. A query to a system of record returns either an answer the agent can rely on or a refusal it can act on. None of the three is free and none is solved.

Software engineering is being redefined as agentic engineering. Its unit of work is moving from hand-authored code to specifications, contracts, orchestrated agent activity, and verification of machine-produced artifacts (Part I). The practitioner's work therefore concentrates in three places: the token window, the state that survives between sessions, and what a system may say when it does

not know. What infrastructure sits underneath agentic engineering, and how should the engineer control it?

1.2 The measured constraints

Two measured facts constrain everything below. The context window is smaller in practice than on the specification sheet, and where material sits inside it changes whether the model uses it. The length of task an agent completes reliably is bounded, and it is improving on a measured trend rather than in a step change (Section 3).

Long work therefore has to be decomposed. Most of what an agent knows at any moment was put there deliberately. What it needs later has to be written down somewhere it can find again.

1.3 Scope

Five objects carry the argument: context engineering, persistent memory, the governed data layer, provenance per field, and refusal over plausible error. Two of them, the governed data layer and refusal over plausible error, have no settled operational form in any located source and are stipulated here with the other three (Section 2), with labels for the earlier Parts and the synthesis to cite.

Everything else is borrowed and used without re-explanation. Agent, autonomy level, unit of work, and agentic engineering are Part I's (Definitions 4.2, 4.3, 4.5 and 4.7). Tool, context window, skill, contract, sandbox, orchestrator, verification, evidence class, and the seven compositional terms are Part II's (Definitions 3.1 to 3.9 and 3.10 to 3.16), as is the layered reference architecture that Section 8 instantiates. The loop, the gate, and bounded iteration are Part III's (Definitions 2.2, 2.1 and 2.4), along with the failure catalogue, which is cited once and not restated. Team, team contract, shared board, and the ownership rule are Part IV's (Definitions 2.1 to 2.4).

1.4 Contributions

Persistent memory is a fold over an immutable event history. That framing splits a design question about what to remember into two: what the step function keeps, and what the retrieval function selects under a budget. A memory that also mutates in place gives up replayability, and no inspection of the history detects that it did.

A governed data layer is a typed query evaluated at a boundary before any source is read. Per-field provenance and typed refusal are the two properties that make its answers usable to an agent, and the case study for both is reported with its failures in the same place as its design.

A refusal is actionable only if the outcome type has no variant meaning both "nothing matched" and "I could not determine this." One such variant destroys the property everywhere, which makes it a property of the type rather than of any call site.

The engineer's role follows from the Part I shift. Practitioner evidence places security requirements and other constraints at review time rather than at task specification, which makes system-enforced boundaries necessary.

Three missing measurements would quantify the transition's verification cost, intervention rate, and security exposure. They govern deployment decisions rather than the observed redefinition of engineering work.

1.5 Disclosure

Four code bases are examined here at pinned commits: an agent operating system, an agent platform, a governed data layer, and a memory layer. They share one author, who is the author of this

series, so they are one closely examined lineage rather than four independent confirmations, and the selection rationale is given with the case studies (Section 8).

2 Definitions

The five definitions below are operational in the sense the series uses throughout. Each names something an observer or a program could check on a running system, and no definition leans on an object whose own definition leans back on it.

Two of the five are this series' own stipulations: the governed data layer and refusal over plausible error. No located source gives either a settled operational form, and "governed data layer" is used here as a proposed abstraction drawn from one code base, not as industry consensus. The vocabulary of the area is unsettled: the nearest academic treatment proposes splitting software engineering into practices for humans and practices for agents rather than offering a settled architecture [23]. The other three definitions draw on an existing literature and are written to be compatible with it.

Everything below sits underneath the reference architecture of Part II. The context window is Definition 3.2 there and is used here without restatement, as are typed boundary (3.11), pure step (3.12), effectful step (3.13), and immutable artifact (3.14).

Definition 2.1 (Context engineering)**.** The practice of deciding, for each inference call, which tokens occupy the context window and in what order, given that the window is bounded and that measured accuracy varies with content and with position inside the window. It is distinguished from prompt engineering by its object: context engineering operates over the whole assembled context, including retrieved material, tool results, system instructions, and conversation history, not only over the instruction the practitioner typed.

Definition 2.2 (Persistent memory)**.** Stored state that outlives a single agent session and that a later session can bring into its own context window by issuing a query, rather than by a practitioner pasting it. State is persistent memory only if the retrieval is selective. A file that is always loaded in full is configuration, not memory, and the test that separates them is whether two different queries against the same store can return different subsets of it.

Definition 2.3 (Governed data layer)**.** A component that sits between agents and systems of record and enforces three properties on every answer it returns. First, the caller was authorized before the query was planned. Second, every returned value carries the source and the version of the mapping that produced it. Third, a query the component cannot execute soundly returns a typed refusal rather than a partial, estimated, or silently narrowed answer. A component that enforces two of the three is not a governed data layer under this definition, and the third property is the one implementations most often omit.

Definition 2.4 (Provenance per field)**.** A record attached to each returned value naming which source produced it, under which mapping version, at what time, together with the alternatives that were considered and not selected. Provenance is per field, rather than per row or per query, exactly when two fields of the same row can carry different sources and a caller can read that difference off the response.

Definition 2.5 (Refusal over plausible error)**.** A property of a system rather than of a single call: when the system cannot establish that an answer is correct, it returns a typed, machine-readable statement of why it cannot answer, and that statement is structurally distinct from an empty or zero result. A system has this property only if a caller can tell "nothing matched" apart from "I could not determine this" without parsing prose.

2.1 Notation

Three pieces of notation recur. An event history is a finite sequence $h = \langle e_1, \ldots, e_n \rangle$ of recorded events, appending an event is written $h \cdot e$, and the memory state derived from a history is written $M(h) = \text{fold}(f, m_0, h)$ for a step function f and an initial state m_0. Selective retrieval is written $\text{recall}(M(h), q, b)$, with query q and token budget b. A field observation is a tuple (v, s, ν, t) of value, source, mapping version, and observation time; a field provenance is a selected observation together with the set of observations considered. An outcome is written $\text{Ok}(x)$ or $\text{Decline}(r)$ with a typed reason r. An empty answer is $\text{Ok}(\emptyset)$ and is never a decline.

Repository material is cited in the form "ContextFS at a93035de04, `src/contextfs/core.py`, lines 673 to 861".

3 The substrate as constraint

3.1 Effective context

Vendors quote a context window in tokens. That number is the input the model will accept, not the length over which it holds accuracy. Four sources measure the gap by different methods and agree on its direction.

Position matters as much as length. Varying where the gold document sits in a multi-document question-answering task produces a U-shaped accuracy curve: performance is highest when the relevant passage is near the start or the end, and degrades when the model must reach for it in the middle [28]. The finding held for models built specifically for long contexts. The tasks are partly synthetic, the evaluation is English only, and the models tested are now several generations old.

The sharpest measurement of the gap is more recent. NoLiMa builds needles that share no literal wording with the question, so retrieval requires an associative step rather than a string match. Across 13 models each claiming at least 128K context, 11 fall below 50% of their own short-context baseline once the input reaches 32K tokens [33]. One named model falls from 99.3% to 69.7%. The result is a preprint, and its authors are industrial researchers evaluating third-party models.

RULER attacks the same question from the benchmark-design side. It spans 13 task types across 17 models, covering needle variants, multi-hop tracing, and aggregation. Almost all models show large performance drops as context length increases, and only about half of those claiming 32K or more remain satisfactory at that length [25]. Its authors state that the needle-in-a-haystack format under-tests what long-context work requires.

A vendor technical report on 18 frontier models finds the same effect on deliberately trivial retrieval and replication tasks, and finds that a single distractor reduces performance relative to baseline [24]. It is not peer-reviewed. Its publisher sells a vector database and so has a commercial interest in the conclusion that curating context matters more than enlarging it. It is cited here as corroborating direction, not as independent confirmation.

One technique often offered as a remedy does not address this problem. Retaining the initial-token key-value cache as attention sinks recovers what naive window attention loses in long-running streaming inference, with up to a 22.2 times speedup over sliding-window recomputation [52]. That result is about stability and efficiency in streaming settings. It does not make a model attend better to material buried in the middle of a long input.

3.2 The task horizon

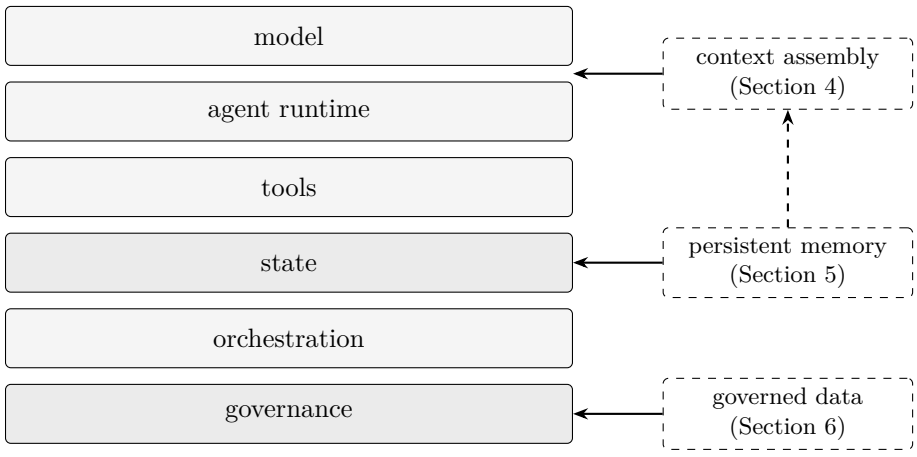
The second constraint is temporal. Regressing the length of task a model completes at a 50% success rate against release date, across six years of models, gives a doubling every 212 days with a

95% bootstrapped confidence interval of 171 to 249 days [26]. That is roughly plus or minus 19%. The same work puts the 50% time horizon of current frontier models at around 50 minutes. Its authors caveat that the trend may have accelerated in 2024, and the measurement is of one research organization's own task suite, so it inherits that suite's task distribution.

The horizon is therefore a moving bound with a known slope. A system designed around a fixed assumption about how much an agent carries in one pass will be wrong in both directions over a two-year window. Designs that keep the decomposition explicit, and keep what an agent learned outside the agent, survive the movement.

3.3 Consequences

The three subjects of this Part sit underneath the layered architecture of Part II (Figure 1). Context assembly sits between the state layer and the model on every call. Persistent memory sits in the state layer and outlives the session. The governed data layer sits in the governance layer and stands between the agent and any system of record.



Solid arrows: the subject of this Part sits in that layer.
Dashed arrow: retrieval from memory feeds context assembly under a token budget.

Figure 1: The three subjects of this Part are placed against the layered reference architecture proposed in Part II, whose second layer is the program Part II defines in its Definition 3.6. Part V adds no layer and develops what sits inside three of them.

4 Context engineering

4.1 Origins of the term

Deciding what an inference call sees is as old as prompting. The name is recent, and its emergence is unusually easy to date: the sequence spans a fifteen-week window in 2025 (Table 1).

The account in which one person popularized the term single-handedly does not survive the timeline. The vendor blog post predates the most widely circulated social post by two days.[1] The

[1]Neither cited page prints the date of the 25 June post. The weblog links it by its status identifier, 1937902205765607626, which decodes to 25 June 2025 at 15:54 UTC.

Date	Source	Contribution
19 Jun 2025	Lutke, a post on X	Preferred "context engineering" to "prompt engineering" and glossed it as providing all the context for the task. The primary post could not be retrieved directly and is quoted through [47].
23 Jun 2025	Chase, a vendor blog	Gave the earliest definition that could be fetched directly: building dynamic systems to provide the right information and tools in the right format so the model can plausibly accomplish the task [13].
25 Jun 2025	Karpathy, a post on X	Called it the art and science of filling the context window. The primary post could not be retrieved directly and is quoted through [47].
27 Jun 2025	Willison, a weblog	Synthesized the exchange and became the citable record for the two posts above [47].
17 Jul 2025	Mei et al., a survey	Formalized the practice academically over more than 1,400 papers, with a taxonomy of retrieval and generation, processing, and management [32].
29 Sep 2025	Anthropic, an engineering blog	Gave the vendor definition: the strategies for curating and maintaining the optimal set of tokens during inference [4].

Table 1: "Context engineering" emerged as a named practice over fifteen weeks in 2025. Two of the six primary sources are social posts that could not be fetched directly and are cited through a practitioner weblog that quotes them.

academic survey then appeared within a month of the coinage, which says more about how fast a label spreads than about how much was new in it.

Definition 2.1 is compatible with all six sources and adds what makes the activity engineering rather than writing. It operates over the whole assembled context, and it operates under a measured constraint. The vendor definition's own page states that as the number of tokens in the context window increases, the model's ability to accurately recall information from that context decreases [4].

4.2 Deployed mechanisms

Deployed systems use three mechanisms to keep an assembled context inside the region where the model still performs. Each solves a different problem. Conflating them produces designs that solve none of them.

Compaction. When a conversation approaches the window limit, older context is summarized and the conversation continues from the summary. One vendor implementation triggers by default at 150,000 input tokens, with a configurable minimum of 50,000 [3]. Compaction addresses length. It does nothing about position sensitivity, and it is lossy by construction: what the summarizer did not judge important is gone, and no later step can recover it. A system that relies on compaction alone has no mechanism for a fact that mattered only in hindsight.

Just-in-time retrieval. Rather than loading everything relevant at the start, the agent records what it learns in files and reads them back when needed. A vendor tool for this exposes six file operations over a dedicated directory and is documented as complementary to compaction rather than a replacement for it [5]. This addresses both length and the hindsight problem, at the cost of requiring the agent to know what to look for. A figure for token savings from this tool circulates widely; it does not appear on the primary documentation page, and it is therefore not used anywhere in this series.

Instruction files with a load hierarchy. Deployed agent memory in 2026 is closer to files than to vector databases. One vendor documents four scopes of instruction file, loaded from broadest to most specific. Alongside them sits an automatically written memory directory whose index loads at the start of every session, up to the first 200 lines or 25 KB, whichever comes first; the topic files behind it are read on demand [6]. A cross-vendor convention for the same purpose is now stewarded by a foundation under the Linux Foundation and describes itself as a README for agents [1].

That size limit is a token budget applied to the one artifact loaded unconditionally. The principle generalizes.

Proposition 4.1 (Selective retrieval without a budget is not context engineering)**.** *Let* $\mathrm{recall}(M(h), q, b)$ *denote selective retrieval from a memory state under query* q *and token budget* b*. If a system implements the query* q *but leaves* b *unbounded, the assembled context length becomes a function of how much matched rather than of what the model can use, and the position and length effects of Section 3 return in full. Selectivity without a budget relocates the problem from the store to the window; it does not remove it.*

Retrieval reduces the candidate set. Only a budget reduces the assembled context. A system whose retrieval returns k results, and whose formatter truncates each result to a fixed number of characters, has bounded the context in characters rather than tokens, and has no bound at all once the results are numerous. The memory-layer case study in Section 5.3 is exactly that system.

4.3 Assembly as an enforced boundary

Context assembly is easier to reason about when it is a named step behind a boundary than when it is string concatenation spread through a code base. One first-party implementation shows the shape.

In the agent operating system examined here, a module called `ContextBridge` sits between the memory store and the agent. Before an agent starts, it queries the memory layer over a command-line interface on the host, renders the results as plain Markdown files into a context directory, and mounts that directory read-only into the agent's microVM. After the agent finishes, it scans the agent's output directory and ingests what was written back into the memory layer with identifiers and lineage tags. Its module documentation states the rule directly: agents never call the memory store, they read files and write files, and the orchestrating module does all store interaction on the host (agent-os at 61cb3994da, `src/agent_os/lib/agent_os/context_bridge.ex`, lines 1 to 60).

Three properties of that arrangement have names in the vocabulary of Part II. The rendering step is pure for the agent, because the same query results produce the same files. The mount is a typed boundary in a weak sense: the agent can receive only Markdown files at known paths, so a violation is detectable without running it. The read-only mount stops the agent altering its own context store, which leaves ingestion as the only write path.

5 Persistent memory as a fold

5.1 Lineage

Agent memory has three visible ancestors, and practitioners tend to inherit one without noticing the others.

The operating system supplies the founding analogy. MemGPT proposed virtual context management drawn from hierarchical memory systems in traditional operating systems, treating the context window as main memory and an external store as disk, with the model issuing the paging

calls itself [36]. Several later systems in this review use the same external-memory analogy, whether or not they cite MemGPT; the located sources do not support a prevalence claim about the field as a whole.

The simulation literature supplies the retrieval function. Generative Agents introduced a memory stream scored by recency, importance, and relevance, each min-max scaled and summed with equal weights; recency decays exponentially over simulated hours since last retrieval, and importance is a model-rated score [37]. Later systems reuse related weighted retrieval schemes, but the located sources do not establish that every such system descends from this one. Its evidence is thin: twenty-five simulated agents, no held-out ground truth for what should have been retrieved, and evaluation by human believability ratings rather than task accuracy.

Information retrieval supplies the machinery but not the framing. Retrieval-augmented generation pairs a parametric model with a non-parametric index [27]. Repository-level code completion improves an in-file baseline by over 10% through iterative retrieve-then-generate [56]. A graph-plus-PageRank scheme beats standard retrieval-augmented generation on multi-hop question answering by up to 20% [22]. A survey systematizes the fragmented field [57], and a later system organizes memories into a linked network in the style of a Zettelkasten, letting a new memory update the representations of older ones [53].

The retrieval function, not the storage format, is where a memory system's design lives (Table 2).

System	Retrieval function	Source
Generative Agents	The sum of recency, importance, and relevance, each min-max scaled with every weight equal to 1. Recency decays at 0.99 per simulated hour since last retrieval, importance is a model-rated score from 1 to 10, and relevance is embedding cosine similarity.	[37]
MemGPT	Model-issued paging between an in-context working set and an external store, by analogy with virtual memory.	[36]
HippoRAG	Personalized PageRank over a knowledge graph built from the corpus, in place of flat vector similarity.	[22]
A-MEM	A linked note network in which a new memory both links to and updates the representations of existing ones. The published description does not say whether those updates append a revision or overwrite in place, which is exactly the distinction Proposition 5.1 turns on.	[53]
ContextFS (case study)	Reciprocal rank fusion of keyword and vector results at $k = 60$ with weights 0.4 and 0.6, a 1.5 times multiplier for eight human-authored types, and 40% of result slots reserved for non-code memories.	The repository, Section 5.3

Table 2: Published retrieval functions are compared side by side. Every row is a different answer to the same question: given a query and a store, which memories reach the window.

5.2 The fold

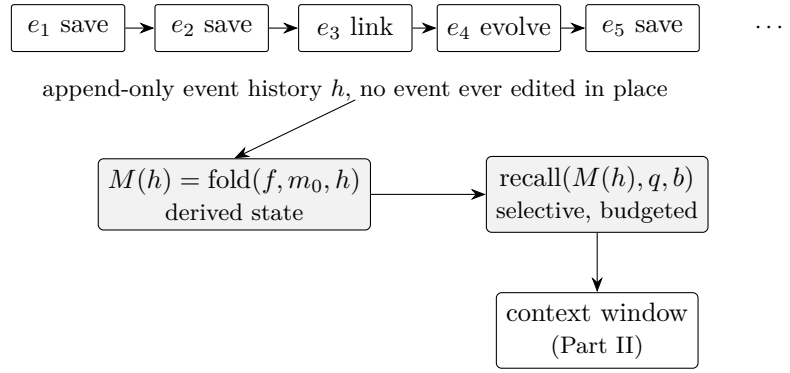
Storage and scoring do not tell a reader what a memory system guarantees. One framing does, and it is borrowed from the compositional vocabulary of Part II, Definitions 3.10 to 3.16.

Write the recorded history as a finite sequence $h = \langle e_1, \ldots, e_n \rangle$ of events. An event is a save, a derivation, a link, or a deletion marker. Each is an immutable artifact in the sense of Part II, Definition 3.14: once written it is not edited, and a correction is a new event that supersedes an old

one while both remain readable. The memory state is then

$$M(h) \;=\; \mathrm{fold}(f, m_0, h),$$

for a step function f that takes a state and an event and returns a new state, and an initial empty state m_0. Retrieval is a separate, pure function $\mathrm{recall}(M(h), q, b)$ that selects from the state under a query and a token budget (Figure 2).



An in-place update writes into $M(h)$ without appending to h.
After one such write, $M(h)$ is no longer recoverable from h alone (Proposition 5.1).

Figure 2: Persistent memory is a fold over an immutable event history, with retrieval as a separate budgeted selection.

Proposition 5.1 (Replayability requires a complete event history)**.** *Assume a deterministic fold and events that record source identity, version, and transformation identity. If every operation that changes what the system remembers appends such an event to h, then the current state and every prefix state are recoverable from h, and element-level provenance can be derived when the fold records which events contribute to each element. If a live state can also be changed without a corresponding event, that live state is not recoverable from h alone. Earlier prefix states and any separately retained provenance remain recoverable.*

Argument. Under the stated determinism assumption, M is a function of h, so replaying a prefix reproduces the corresponding prefix state. Exact element provenance requires the additional contribution relation; append-only storage alone does not supply it. For an unrecorded update, let u be applied after e_n. The history h is unchanged by construction, so $\mathrm{fold}(f, m_0, h)$ still evaluates to the pre-update state while the live state is the post-update one. Since h contains no record of u, no function of h reconstructs the post-update state. This does not erase states already recoverable from retained prefixes. □

The proposition does not say in-place update is wrong. It says that a system offering both operations offers two different guarantees under one name, and that which guarantee a caller gets is decided per call. Callers have to be told which. The case study below is exactly that situation.

5.3 A memory layer read as a fold

ContextFS is a local-first persistent memory layer for coding agents, built on SQLite for structured storage, SQLite FTS5 for keyword search, and a local vector store for semantic search, exposed through a command-line interface, a Python interface, and a tool server. All line references below are to the pinned commit.

The append path. A save computes a 16-character prefix of the SHA-256 of the content. If a memory with that hash already exists in the namespace, the save returns the existing memory instead of inserting a duplicate (ContextFS at a93035de04, `src/contextfs/core.py`, lines 718 to 740). Content-addressed deduplication at the append point makes replay idempotent: the same event twice produces one state change.

The two derivation operations. The system offers both disciplines of Proposition 5.1, and its own documentation is explicit about the difference. The `evolve` operation creates a new memory with the updated content and bidirectional lineage edges to the original, and its docstring states that the original memory remains unchanged (ContextFS at a93035de04, `src/contextfs/core.py`, lines 1515 to 1554, and `src/contextfs/memory_lineage.py`, lines 167 to 180). The `update` operation modifies fields of an existing memory in place (same file, from line 1391). A caller who uses `evolve` gets a fold; a caller who uses `update` does not, and the store cannot tell the difference afterwards.

The retrieval function. Hybrid search runs keyword and vector retrieval in parallel and merges them by reciprocal rank fusion at $k = 60$, with weights 0.4 for keyword and 0.6 for vector (ContextFS at a93035de04, `src/contextfs/fts.py`, lines 741 to 750 and 795). Eight memory types that the code comments describe as typically human-created are then multiplied by a boost factor of 1.5, and 40% of the result slots are reserved for non-code memories before the remainder is filled best-first (same file, lines 519 to 535 and 831 to 859). The code states the reason for both adjustments: automatically indexed code memories would otherwise crowd out the decisions, procedures, and errors a practitioner wrote down.

Written with the case study's own operations, the three pieces fit the framing except for the missing budget argument (Listing 1).

```
# 1. Append to the history. Deduplicated by content hash.
# core.py:718-740 returns the existing memory on a hash collision.
m1 = ctx.save("Retry budget for the payment webhook is 3",
              type=MemoryType.DECISION)

# 2. Derive without mutating: a new memory plus EVOLVED_FROM /
# EVOLVED_INTO edges. core.py:1515-1554. The original stays readable.
m2 = ctx.evolve(m1.id, "Retry budget raised to 5 after the Sept incident")

# Contrast: this writes into the state without appending to the
# history. core.py:1391. After it, m1's old content is gone.
ctx.update(m1.id, content="Retry budget raised to 5")

# 3. Selective retrieval: RRF(k=60, 0.4/0.6), type boost 1.5x,
# 40% of slots reserved for non-code types. fts.py:741-750, 831-859.
hits = ctx.search("payment webhook retry", limit=10)

# 4. The missing argument. There is no token budget on this path;
# formatting truncates each memory to 200 or 300 characters
# (schemas.py:1422-1427), which is a character bound, not a token bound.
context = "\n".join(h.memory.to_context_string() for h in hits)
```

Listing 1: The append, the fold, and the budgeted retrieval are written with the case study's own operations. The budget argument is the one piece the implementation does not have.

No token budget on the retrieval path. The system has no token budget on the way out, though it validates rigorously on the way in. Ingestion checks structured data against per-type

JSON schemas and rejects a save whose payload does not conform (ContextFS at a93035de04, `src/contextfs/schemas.py`, lines 1410 to 1420). Retrieval counts characters instead. The context formatter truncates to the first 200 characters when a summary is present and the first 300 otherwise (same file, lines 1422 to 1427). The session-start hook that injects recalled context runs the command-line auto-recall, which defaults to five memories per type across three types and truncates each to 100 characters (ContextFS at a93035de04, `src/contextfs/cli/memory.py`, lines 795 to 860). A real tokenizer exists in the code base and serves only the ingestion chunker (same commit, `src/contextfs/rag.py`, lines 694 to 712). This is Proposition 4.1 in a shipped system, on exactly the path where the constraint of Section 3 applies.

Other limitations of the case study. Its documentation lists three tool names that do not exist in the tool server and omits seven that do. Its architecture documents describe a sentence-transformer embedding backend, while the code defaults to an ONNX-based one. Its optional cloud service hashes user passwords with unsalted SHA-256 although a password-hashing library is a declared dependency. It declares an MIT license in package metadata and ships no license file at the pinned commit. None of this bears on the fold reading. All of it bears on how much weight a reader should put on any single first-party system.

The temporal model. A companion formal working series by the same author (2026, in preparation) examines this code base and states two results about its lineage. One is positive: a record's ancestry cannot be silently rewritten once written, which is the immutability guarantee the fold reading needs. The other is negative, and sharper than a missing feature. The system records when something was written, not when it was true, and that series argues the distinction is unrecoverable from the stored data rather than merely absent. The stored model carries created-at and updated-at, and version entries carry one timestamp rather than a pair. This Part therefore does not describe the system as implementing independent valid-time and transaction-time axes, and neither should anyone citing it. That companion series contains no experiment, no measurement, and no benchmark; its results are cited here as arguments about structure, never as evidence about behaviour.

5.4 The evidence base for memory systems

A practitioner choosing a memory system in 2026 chooses under worse evidence than the marketing suggests, because the benchmarks the claims rest on are either unaudited or audited and found defective (Table 3).

Both leading commercial memory systems publish benchmark claims, and both are vendor-authored preprints. One reports a 26% relative improvement in an LLM-as-judge metric over a major vendor's memory feature on LoCoMo, a graph variant about 2% above its own base configuration, 91% lower p95 latency, and over 90% token-cost saving against full-context prompting [15]. The other reports 94.8% against 93.4% on DMR and accuracy improvements of up to 18.5% on LongMemEval [43].

Neither claim survives its benchmark. The first rests on the benchmark an independent audit found materially defective [38]. The second's headline margin is 1.4 points on a benchmark built by the team behind the system it is compared against, and 1.4 points is smaller than the error rate the audit found in the other benchmark. Neither should be repeated without the audit alongside it.

Abstention is one of the five abilities LongMemEval measures. A memory system that cannot decline to answer is measured on the same axis as a data system that cannot refuse (Section 6).

Benchmark	Size	Reliability
LoCoMo [30]	Conversations averaging about 300 turns and 9K tokens, over up to 35 sessions	An independent audit found 99 of 1,540 questions (6.4%) with wrong gold answers, and its judge accepting 62.81% of deliberately vague or wrong-but-topical answers, capping a perfect system at about 93.6% [38].
LongMemEval [51]	500 curated questions across five abilities including abstention	Peer-reviewed at a major venue. It reports commercial assistants dropping about 30% in accuracy on sustained-interaction memory. We located no independent audit.
DMR (used by [43])	Not independently characterized here	Created by the team behind the system it is used to compare against, which is a circularity the comparing paper does not resolve.

Table 3: Three memory benchmarks differ in what is known about their reliability. The two most-cited vendor comparisons both rest on the row with the documented defects.

5.5 Retrieval for code

This question is not settled. A comparison across 427 samples in 25 repositories, with relevance defined by what an agent's workflow needed rather than by semantic similarity, finds that no single retrieval family dominates [41]. One embedding model has the best sample-weighted reciprocal rank on positive samples, a larger one the best recall at 20, and a structural repository map the best budgeted context yield at an 8K-token budget. The dispute between the embedding camp [27, 56] and the structural-search camp therefore turns on a metric choice rather than resolving either way. This is one recent paper across 25 repositories with no independent replication, and it should be held loosely.

6 Governed data and typed queries

6.1 Older traditions

An agent querying a system of record has the same three needs a human analyst has, in a sharper form. The query must be authorized. The answer must say where it came from. An answer the system cannot stand behind must be recognizable as such. Three established traditions already supply two of the three, so a governed data layer belongs beside them rather than presented as an invention.

Semantic layers supply centralized meaning. MetricFlow and the dbt Semantic Layer define metrics once in configuration and reuse them across every consumer, so two teams asking the same business question get the same SQL [17]. A commercial semantic layer now describes itself in agent terms, centralizing metric definitions, joins, access rules, and caching upstream of every dashboard, application, and agent, and validating every query against the model with access policies applied before it reaches the warehouse [16]. That description is vendor positioning rather than a measured result.

Provenance standards supply the vocabulary. The W3C PROV data model gives entities, activities, and agents as the conceptual core, and PROV-O gives its OWL2 encoding. Both have been Recommendations since April 2013 [39, 40]. The database literature supplies the finer distinctions. Why-provenance identifies which source data contributed to a result, and where-provenance identifies the location in the source the result came from; both are set out in a

monograph by Cheney, Chiticariu, and Tan [14]. The authorship is worth stating because a widely circulated citation of the same work adds an author who is not on it.

Lineage tooling supplies the operational form. OpenLineage defines datasets, jobs, and runs with extensible facets, and is a graduate project of a foundation [34]. All three traditions predate agents.

None of them supplies the third property. A semantic layer returns a number. A provenance standard describes where that number came from. Neither says what the system must do when it cannot compute the number soundly. That gap is where an agent differs from a dashboard: a dashboard's consumer is a person who will notice an implausible figure, and an agent's consumer is a program that will act on it.

6.2 Authorization and feasibility

The governed data layer examined here compiles a semantic query against the declared capabilities of each connector, pushes down only what a connector can prove it executes correctly, and reconciles the rest. Two of its checks run before any source is contacted.

Authorization runs first. The policy engine's method for it is named and documented as authorizing and filtering a query before planning, and it fails closed on three grounds: a principal from a different deployment, a workspace the principal does not hold, and a catalog whose source pins do not validate (CatDB at 7cc9341a16, `catdb/crates/catdb-policy/src/lib.rs`, lines 971 to 985). Authorization before planning is stronger than authorization before execution, because a plan that was never built cannot leak its own shape.

Feasibility runs second, and it is the more unusual check. The module deciding it opens by stating its purpose: whether a plan's work is bounded by something declared, decided before a source is read (CatDB at 7cc9341a16, `catdb/crates/catdb-algebra/src/feasibility.rs`, lines 1 to 40). Its documentation draws the contrast with cost-based planning. A cost-based planner estimates, and when the estimate is wrong the query still runs, for an hour or until something kills it, with nothing in the answer recording that the estimate was wrong. This system instead asks of each plan node whether the source declared it executes that node, or whether the node is residual under a bound the plan can prove. A node that is neither produces a refusal naming it. The same documentation explains why the bound is asked of the declaration rather than of the data. A statistic goes stale, is often absent, and reading it costs the read the check exists to avoid, while a declared capability is available with no source contacted.

This is the strongest form of boundary checking in the corpus, in the sense of Part II, Definition 3.11. The typed boundary is checked before the effectful step runs at all, not after it returns.

6.3 Provenance per field

Provenance at row or query granularity answers where a result came from. Provenance at field granularity answers where each value came from. An agent needs the second granularity whenever a row was assembled from three systems that disagree.

The implementation examined here builds it from three types. An observation state is a private five-arm enumeration: an observed value, an observed null, a field the source did not see although it saw the record, a record the source never saw, and a source that was not consulted, with a reason (CatDB at 7cc9341a16, `catdb/crates/catdb-provenance/src/lib.rs`, lines 15 to 70). Collapsing those five into "no value" is the defect the design exists to prevent.

A field observation wraps one such state with the source version identifier, the record key, the mapping version identifier, an optional physical attribute name, and an observation timestamp (same file, lines 171 to 205). The version identifiers point at content-addressed catalog versions, so

an observation names an immutable version rather than a mutable source name.

The third type is what distinguishes this design. A field provenance carries the provenance rule version, the field name, an optional index of the selected observation, and a vector named `considered` holding every observation the resolver looked at, not only the one it chose. Its constructor refuses to build the record if the selected index is out of bounds or points at an entry that was not an observed value (same file, lines 405 to 470, and Listing 2).

```
FieldProvenance {
  provenance_rule_version_id: <uuid>, // which rule chose
  field: "annual_revenue",
  selected: Some(0), // index into considered
  considered: [
    // index 0: the value that was returned
    { source_version_id: <uuid>, mapping_version_id: <uuid>,
      record_key: "acct-4471", observed_at_ms: 1756... ,
      state: ObservedValue(3_000_000) },
    // index 1: a source that saw the record but not this field
    { source_version_id: <uuid>, mapping_version_id: <uuid>,
      record_key: "acct-4471", observed_at_ms: 1756... ,
      state: FieldUnseen },
    // index 2: a source deliberately not consulted, with a reason
    { source_version_id: <uuid>, mapping_version_id: <uuid>,
      record_key: "acct-4471", observed_at_ms: None,
      state: NotAsked { reason: "not authorized for principal" } }
  ]
}
// The constructor returns an error if ‘selected‘ is out of bounds,
// or if it points at an entry whose state is not an observed value.
```

Listing 2: The distinguishing feature of per-field provenance in the case study is that the losing observations are retained and an invalid selection cannot be constructed. The shape is paraphrased from CatDB at 7cc9341a16, `catdb/crates/catdb-provenance/src/lib.rs`, lines 171 to 470.

The cost is measured and it is not small. The repository's own architecture notes record provenance at 52% of query time and 96.3% of wire bytes (CatDB at 7cc9341a16, `docs/architecture.md`, line 201). The same documentation records the consequence. Full records with per-field provenance cannot be returned in one logical query beyond roughly 2,140 to 2,156 records for a five-field, two-source entity, and past that the system returns a capacity refusal rather than truncating (CatDB at 7cc9341a16, `docs/semantic-warehouse.md`, lines 123 to 131). Refusing rather than truncating is the correct behaviour. It is also a hard ceiling on how large an answer the design can deliver.

6.4 Refusal

Proposition 6.1 (Refusal and emptiness must remain observably distinct)**.** *Let a call return* $\mathrm{Ok}(x)$ *or* $\mathrm{Decline}(r)$ *for a typed reason* r*, and let the empty answer be* $\mathrm{Ok}(\emptyset)$*, for calls whose success value is a collection or an option. A caller can then distinguish "nothing matched" from "I could not determine this" by inspecting the outcome alone. Encoding emptiness as* $\mathrm{Decline}(\mathrm{NothingMatched})$ *still preserves the distinction if typed reasons remain visible and no failure maps to that reason. Information is lost only when empty success and indeterminate failure map to the same observable value, or when the caller erases the reason. The property therefore belongs to the observable outcome contract, not to the spelling of one variant. A success type with no empty inhabitant does not raise the question.*

The case study states this rule about itself and enforces it mechanically. The connector-level error

type's module documentation opens with it: there is deliberately no variant meaning "nothing matched." Emptiness is a successful answer with zero rows, carrying the count of what was examined, and every variant of the error type means the read did not happen or did not happen completely (CatDB at 7cc9341a16, `catdb/crates/catdb-connectors/src/error.rs`, lines 1 to 14). A caller treating one of those variants as "no results" is discarding a failure. The documentation names that as the bug the crate is shaped to make impossible.

The same discipline appears at the query-service level as a large closed enumeration. Counting the variants at the pinned commit gives 35 (CatDB at 7cc9341a16, `catdb/crates/catdb-federation/src/service.rs`, lines 708 to 1022). The server layer maps each to a stable machine-readable code and an HTTP status, mostly 422 (same commit, `catdb/crates/catdb-server/src/entity.rs`, lines 339 to 650). Seven variants carry design weight, and three of them are decided before any source is contacted (Table 4).

Variant	Decided	The wrong answer it prevents
`PlanInfeasible`	Before any source is read	An unbounded plan that runs until something kills it, returning nothing that says the estimate was wrong.
`Authorization`	Before planning	A plan built over sources the caller may not see, whose shape leaks even if its rows do not.
`GrainUnproven`	Before reading	Keyset paging under a record key not proven unique silently drops rows, so a short answer looks like a complete one.
`AggregateUnsound`	During execution	A sum across two sources that could both hold the same record. The repository records a demonstration where the naive federated sum of 4,140,000 cents against a true 3,000,000 would have overstated by 38%.
`RelationshipUnproven`	During execution	A field reached through a join whose cardinality was declared but not measured safe, returning a fan-out as if it were a lookup.
`IdentityUnevaluable`	During execution	A silent fall back to exact-key matching when a declared crosswalk cannot key the reconciliation.
`UnionDoesNotDescribeTheMerge`	During execution	Rows merged under a description that disagrees with what the sources actually returned.

Table 4: Seven of the 35 typed refusal variants counted at the pinned commit are shown with the point in the pipeline at which each is decided. The first three are decided before any source is contacted.

The repository records a demonstration that the refusal is decided rather than reflexive. Across two engines holding mirrored order data, a federated sum returns 422 with the aggregate-unsound code naming both source identifiers. Minimum and maximum over the same two sources return 200, and a sum over a text field refuses with a different cause (CatDB at 7cc9341a16, `docs/architecture.md`, lines 39 to 58). A positive control that answers is what separates a designed refusal from a system that has simply stopped working.

6.5 Abstention

The rule above is stated over types because types are where it is enforceable. A longer research tradition sits behind the idea, and it consistently finds that knowing when not to answer is harder than answering.

Selective prediction states the trade formally: accept less coverage in exchange for a guaranteed error rate on what remains. The canonical demonstration guarantees 2% top-5 error on ImageNet with probability 99.9% while retaining almost 60% test coverage [20].

The question-answering analogue made the difficulty measurable. Adding more than 50,000 adversarially written unanswerable questions to a reading-comprehension benchmark dropped a system scoring 86% F1 on the answerable-only version to 66% on the version that also requires abstention [42]. Twenty points is the price of the abstention requirement alone.

Language models decline for more reasons than safety. A noncompliance taxonomy adds four: the request is incomplete, unsupported, indeterminate, or asks the model to be something it is not. Evaluated against that taxonomy, models incorrectly complied with as many as 30% of requests in the understudied categories [11]. The two categories that matter for a data layer, unsupported and indeterminate, are among those handled least well.

A model that cannot reliably decide when to abstain is a poor last line of defense. That is the argument for putting the decision in the query planner, where it is decidable, rather than in the model, where it is not.

6.6 Observed implementation limits

On the repository's own measurements the system has largely stopped working, and it says so about itself in stronger terms than an outside critic would use.

On the Leipzig DBLP-ACM entity-resolution benchmark, with 2,616 rows in one engine, 2,294 in another, and 2,224 known-true pairs, the system merges 0 of 2,224. It returns 4,910 records, which is 2,616 plus 2,294 exactly, so not one record key was produced by both engines. Recall is 0.0000. Precision is undefined because nothing was predicted. A positive control with a synthetic key merges 2,224 of 2,224, so the machinery works and the identity rule is what is missing (CatDB at 7cc9341a16, `docs/semantic-warehouse.md`, lines 83 to 95). The repository's own summary is blunter than the numbers: it merges 0 of 2,224 known-true pairs, and everything else works and returns wrong data because of it (same file, lines 14 to 24).

Three further results sit beside that one. The aggregate, join, and union plan nodes all refuse, so the algebra is defined more widely than it executes (CatDB at 7cc9341a16, `docs/product-plan.md`, line 34). The entity-level authorization method has zero production callers at the pinned commit (same file, line 34). And the repository's contributor instructions record four continuous-integration workflow runs in its entire history, all four failures, with the linting, test, no-default-features, and conformance jobs never having executed on a runner (CatDB at 7cc9341a16, `CLAUDE.md`, lines 168 to 176).

The last of those files carries the result this Part most needs. It records that the single most persistent defect class in the code base is a failure that looks like an empty result, at twelve instances so far, and states the remedy as a type rule: make failure and emptiness structurally different values, and let no error type carry a variant meaning "nothing matched" (CatDB at 7cc9341a16, `CLAUDE.md`, lines 36 to 46). A system built specifically to hold the refusal property, by an author who wrote the property down as a rule, recorded twelve internal violations of it. That is the strongest evidence in this corpus that Proposition 6.1 describes something hard rather than something obvious. It also shows why the redefined discipline puts refusal semantics and verification boundaries under engineering control.

The result separates two properties that a single success rate would conflate. The identity rule fails on the benchmark, while the governed boundary refuses rather than silently returning a false merge. The positive controls answer and the provenance is complete; those guarantees cost 52% of query time, 96.3% of wire bytes, impose a ceiling around 2,150 records, and expose the missing

identity rule as a visible failure. The evidence therefore supports a precise engineering conclusion: governed data prevents one class of silent error, but it cannot supply missing domain identity. What does not fit the evidence is citing the design without the results.

6.7 Limits of typing

Type discipline at one boundary does not survive crossing into an untyped graph runtime. The result comes from the companion formal working series (2026, in preparation). Its witness is an execution manifest whose edges carry node identifiers and no port type, in the platform code base examined by this series at 1c3ad24011.

A component that validates its inputs and outputs rigorously, wired into a runtime whose edges are untyped, gives a guarantee inside the component and nothing across the wire. A stack's typing is its weakest boundary, not its strongest, and the boundary to check is the one between two systems built by different people.

7 Untrusted content

7.1 The framing

The two preceding sections assume the material entering the window is what the engineer intended to put there. That is the least defensible assumption in the stack.

A system is exploitable if it has three properties: access to private data, exposure to untrusted content, and the ability to communicate externally [48]. A model follows instructions found in content without reliably distinguishing where they came from, so an attacker who can place text where the model will read it can direct the tools the model holds. The rule has outlasted competing framings because it constrains the deployment rather than the model, which lets an engineer check a design against it.

Its academic origin is the indirect prompt injection paper, which demonstrated attacks against deployed applications and named three classes: data theft, worming, and information contamination [21]. The standards followed. The 2025 OWASP list for LLM applications puts prompt injection at LLM01, and adds two entries specific to agents: excessive agency at LLM06 and vector and embedding weaknesses at LLM08 [35]. The tool-protocol side has its own security guidance, covering confused-deputy attacks in OAuth proxies, token passthrough, which it forbids outright, server-side request forgery through metadata discovery, session hijacking, local server compromise, and scope minimization [31].

7.2 The measured rates

Two benchmarks give numbers rather than taxonomies. InjecAgent builds 1,054 test cases over 17 user tools and 62 attacker tools across 30 agents, and reports a ReAct-prompted GPT-4 vulnerable 24% of the time, with the rate roughly doubling under a reinforced-instruction technique [55]. AgentDojo constructs 97 realistic tasks in email, banking, and travel domains with 629 security test cases, and measures both attack success and task success under attack; its finding is that models fail many of the tasks even with no attack present, and that existing defenses break some security properties but not all [18].

Neither number describes a solved problem, and neither benchmark's authors present a general defense. We found no general defense with a measured residual attack rate in the sources reviewed (cutoff 2026-09-01).

7.3 Coding agents

In August 2025 attackers exploited a vulnerable continuous-integration workflow to steal a package-registry publish token, and shipped a malicious post-install script in a widely used build system across versions 20.9.0 to 21.8.0. The supply-chain mechanics are not what makes the incident relevant here. The payload is. The malware invoked the developer's own installed AI command-line tools with permission-bypassing flags to perform reconnaissance and credential collection, which the reporting vendor describes as the first known case of attackers turning developer AI agents into supply-chain attack tooling. The reported scale was over 1,000 valid repository-host tokens, dozens of cloud and registry credentials, and roughly 20,000 files exposed in the first phase, with a later wave affecting more than 400 organizations and making over 5,500 private repositories public [50].

An agent with private data access, a tool surface, and a permission-bypass flag is the third leg of the trifecta, already installed on the developer's machine. Applied to the four code bases this Part examines, the check clears none of them (Table 5).

System	Private data	Untrusted content	External communication	Reading
Agent operating system	yes	yes	yes	All three legs are present. They are mitigated by microVM isolation and a read-only context mount, not by removing a leg.
Agent platform	yes	yes	yes	All three legs are present. Its declared sandbox variants and tool policy bound the third leg rather than removing it.
Governed data layer	yes	partly	yes	It holds authorized access to systems of record. Its query surface is typed and closed, which narrows what injected text can express.
Memory layer	yes	yes	optional	It indexes repository content and agent output, so untrusted text reaches the store. External communication occurs only if the optional sync service is enabled.

Table 5: The three-property check [48] is applied to the four code bases examined in this Part. No row is clear of it, which is the expected result for any useful agent deployment and is why the check is a design prompt rather than a pass or fail.

7.4 The cost of plausible fabrication

The empirical case for refusal over plausible error (Section 6) rests on one large study.

Across 16 code-generating models and 576,000 generated samples in Python and JavaScript, 19.7% of the 2.23 million recommended packages did not exist: 440,445 hallucinations, including 205,474 unique non-existent names. Commercial models averaged at least 5.2% and open-source models 21.7%. Resampling 500 prompts that had produced a hallucination, and repeating each ten times, 43% of hallucinated packages recurred in all ten queries while 39% did not recur at all [44]. The study covers two languages and a mid-2024 model generation.

The 43% figure turns a nuisance into an attack surface. A reproducible fabrication is one an attacker can predict and pre-register in advance. A system that returns a plausible name it cannot stand behind is not merely wrong. It is publishing a target. That is the practical argument for Definition 2.5, and it rests on no first-party code base.

7.5 Limits of recording effects

Fencing protects the record, not the world. The companion formal working series (2026, in preparation) observes that a model of effects has no vocabulary for an effect that was performed but never recorded. Part II's rule, from its Definition 3.13, that effects sit at the edges and every invocation is recorded, gives a complete audit of the invocations the system knows about. An injected instruction causing an unrecorded side effect falls outside that guarantee by construction. Detection has to come from the affected system rather than from the agent's own trace, and no arrangement of the agent's own records closes the gap.

8 The repositories

8.1 Selection rationale

Three criteria selected the four code bases used as case studies. Each is a working system rather than a demonstration, with tests, pinned commits, and a documented design. Each occupies a different layer of the Part II architecture, so together they instantiate the architecture rather than repeating one point in it. Each documents its own failures in a form specific enough to cite, which is what makes the implementation analysis in this Part possible.

agent-os, at 61cb3994da An agent operating system in Elixir. Used in this Part only for where state lives and how it crosses into an agent, through its context-assembly module. Its orchestration behaviour is Part IV's subject.

AgentHero platform, at 1c3ad24011 An agent platform in Rust. Used in this Part only for where session state lives, through the searchable memory projection described below, and for the point at which its manifest edges stop carrying types.

CatDB, at 7cc9341a16 A federated, versioned data layer in Rust. The primary case study for Section 6.

ContextFS, at a93035de04 A local-first memory layer in Python. The primary case study for Section 5.

8.2 A projection

The platform's session memory is a clean instance of the fold reading. A database migration creates a table of session memory chunks keyed by a source event identifier, with a header comment stating the rule: the event log remains canonical, and every memory row cites its exact source event. Each row carries the run identifiers, the provider and model, a content hash, and a generated full-text search vector, indexed (AgentHero at 1c3ad24011, `supabase/migrations/20260824000001_agent_session_memory.sql`, lines 1 to 35). This is $M(h)$ built from h and declared rebuildable from it, which is the property Proposition 5.1 identifies. The load-bearing comment is the one saying the projection is derived, not authoritative.

8.3 Author's relationship to the repositories

The four code bases share one author, who is the author of this series. One of them names another as its reference implementation in its own accompanying paper. A companion formal working series (2026, in preparation), also by the same author, examines three of the same four as its own case

studies. They are one closely examined lineage, and agreement between them corroborates nothing. Every argument in this Part that rests on repository evidence uses it as an existence proof that a design can be built and what it costs when it is, never as evidence about how common the design is or how it performs against alternatives.

The description of that companion series as this series' formal counterpart originates in this series: the companion series nowhere describes a practitioner-facing counterpart, so the relationship is asserted here rather than by it. And three of the four code bases declare a permissive license in package metadata without shipping the license text at the pinned commit. That is a fact about the repositories, recorded here and not a legal conclusion.

9 The engineer's role

9.1 Derivation

Part I's shift is from authoring the artifact to specifying, constraining, and verifying it. Six activities follow from that shift and the substrate above, and a reviewer could check that each was done.

Write contracts a program can decide. Part II's contract, Definition 3.4, is a statement of what an agent must produce and how the result will be checked, evaluable without asking the model that produced the work. Writing one is now a first-class deliverable, and it is the artifact a reviewer rejects.

Design the bounds and the gates. Part III's bounded iteration and gate, Definitions 2.4 and 2.1, are parameters someone chooses. A maximum pass count with no distinct exhaustion outcome is a bound that reports failure as success.

Own the assignment and the record. Part IV's ownership rule and shared board, Definitions 2.4 and 2.3, are the mechanism by which conflicting changes become detectable. Someone assigns each shared object to exactly one writer, and that someone is the engineer.

Budget the context. The practitioner decides what occupies the window and in what order, under Proposition 4.1, against a window whose usable length is measurably shorter than its advertised length.

Curate the memory. Deciding what is worth remembering, what supersedes what, and which derivations append rather than overwrite is a design decision with the consequences Proposition 5.1 sets out.

Decide what the system must refuse. Enumerating the situations in which the system must decline rather than answer, and giving each a distinct type, is the governance work of Section 6, and it is the one item on this list with no pre-agentic equivalent in most code bases.

9.2 Evidence for the gap

The list above says where effort should go. Measured evidence says it does not yet go there.

None of the 15 professional engineers in one study specified security requirements in their initial prompts, even when they had the relevant knowledge. The study combined interviews with observed coding tasks across three experience cohorts, and found that experience level did not predict outcomes; its authors describe a decoupling of security awareness from security behaviour [8].

It is a small qualitative study and should not be read as a rate. It is still a direct observation of the specification step not happening.

The review side shows the same pattern at larger scale. Across 400 repeat reviewers and 11,429 reviews of agent-authored pull requests over seven months, approval rates rose from 30.1% to 36.8%, inline comments fell 22%, and review latency rose 3.5 times [54]. The authors cannot rule out that the code genuinely improved, and say so. The pattern fits habituation and it also fits rational recalibration. Vendor telemetry over 22,000 developers and more than 4,000 teams, in a within-organization design with no randomization, reports pull requests merged without review up 31.3%, median time to first review up 156.6%, and median time in review up 441.5% [19]. Those three review-time figures are distinct sub-metrics reported side by side, not a contradiction.

The most demanding productivity result for this control layer is a randomized trial. Sixteen experienced maintainers worked 246 real issues in repositories they had contributed to for years, and allowing AI increased completion time by 19% [9]. The same developers had forecast a 24% reduction, and after finishing still believed they had been sped up by about 20%. The task class is narrow and the authors say so. It is nonetheless a result about a population that already had the specification skills. Section 5 of Part I sets it beside the other randomized trial, in which the 35 marketplace freelancers who completed one standardized solo task, an HTTP server in JavaScript, were 55.8% faster with a completion assistant; the two trials measured different populations on different task classes. The difference locates where agentic engineering must account for repository context, verification cost, and practitioner expertise; it does not turn direct authorship back into the discipline's organizing center.

Field evidence is thinner but points the same way. An observational study of 302,600 verified AI-authored commits across 6,299 repositories found that 22.7% of tracked AI-introduced issues still survive at the latest version of the repository, with code smells accounting for 89.3% of issues [29]. The study has no control arm.

Specification failures have a documented taxonomy. A study of 150 hand-annotated multi-agent traces, with inter-annotator agreement of $\kappa = 0.88$, produced 14 failure modes in three categories, the first being system design and specification issues [12]. A practitioner who writes a contract a program can decide is acting on that category. The taxonomy classifies traces from open-source frameworks rather than reporting a rate, and its authors note it may not generalize to proprietary production systems.

A conceptual problem sits underneath the measurement problem. A peer-reviewed critical review argues that software-engineering research routinely equates trust with the likelihood of accepting generated content, which does not capture the concept, and that the field rarely embeds its findings in the trust frameworks of neighbouring disciplines [7]. Acceptance rates are therefore weak evidence about calibration, and calibration is what an engineer's judgment has to be.

Two practitioner observations complete the picture, and neither is a measurement. The first is that the binding constraint has moved: 200 lines of working, debugged, production-level code is an incredibly good day, the new limiting factor is cognitive capacity, and the discipline once enforced by how long things took now has to come from somewhere else [49]. The second draws the distinction this Part's account depends on. In vibe coding you do not care about the code, only the behaviour of the system; in augmented coding you care about the code, its complexity, the tests, and their coverage [10]. The role described here is the second.

An industry technology radar names the organizational form of the same problem. Codebase cognitive debt, placed in its caution ring, is the growing gap between a system's implementation and a team's shared understanding of how and why it works [45]. The same volume cautions against coding throughput as a productivity measure, because cycle times increase as engineers raise pull requests filled with insufficiently reviewed AI output, leading to repeated back-and-forth

with reviewers [46]. Both are consultancy judgments rather than studies.

9.3 Functional thinking

Part I names composition and functional thinking as the skill that grows in importance, and Part II defines the seven terms. The demonstration here is that all three of this Part's main results are statements about boundaries rather than about steps.

The fold reading of memory separates an append-only history from a derived state and puts the guarantee in the relation between them, which is why Proposition 5.1 can be argued in four lines. The governed-data reading puts authorization and feasibility at a boundary checked before the effectful step runs, which is what makes a refusal cheap enough to be the default. The refusal reading is a statement about a type rather than a call site, which is why one bad variant destroys the property everywhere.

This vocabulary is a design discipline argued from cases, not an empirically established improvement. The strongest observational study associating functional languages with lower defect rates is substantially deflated by its own peer-reviewed reproduction. The direct evidence on whether typed output boundaries help or hurt model reasoning is an unresolved dispute with a commercial interest on one side. Part II carries both citations where it owns the terms. The tradition these ideas come from is about programs written by people, so its bearing on agent systems is by analogy and by case, which is how it is used here.

10 Open problems

Ten problems remain open, and three of them are measurement gaps bearing directly on the economics and governance of the redefinition (Table 6).

11 Current substrate boundaries

Agentic engineering requires a deliberate substrate choice rather than an agent-facing layer on every system. Four situations call for direct data access, fixed configuration, or coarser provenance inside the larger architecture, and the case studies supply the evidence for two of them.

When the consumer is a person looking at a dashboard. Typed refusal earns its value from the absence of a human plausibility check. The argument for it assumes the consumer is a program that will act on a wrong number without noticing. An analyst reading a chart supplies the check no type system can, and for that reader a semantic layer returning a best effort with a caveat beats one returning 422.

When the reconciliation problem is unsolved. The case study that refuses correctly and merges 0 of 2,224 known-true pairs is the clearest evidence in this Part. If two systems of record share no key and no crosswalk exists, a conventional pipeline with a documented fuzzy match and a human review queue delivers answers today, while the governed layer delivers refusals. The choice turns on whether a wrong merge costs more than no merge. For most reporting workloads it does not.

When the answer size exceeds what provenance can carry. Per-field provenance measured at 96.3% of wire bytes and a capacity refusal above roughly 2,150 records is a hard bound. Bulk

Open problem	The measurement that would close it
Verification cost against authoring cost	Is verification cheaper than authoring at equivalent quality? Closing this needs verification minutes per accepted unit of work against authoring minutes per unit for a matched hand-authored baseline, on the same task. None of the sources we located measures both sides for one task. This is the series' most important gap.
Human intervention count per task	The number of times a human supplies input, correction, or approval between a task's start and its acceptance, divided by tasks, over a stated window. This is the most direct operationalization of autonomy level. None of the studies we located reports it.
Organizational reversion	Organizations that adopted coding agents broadly and later restricted or withdrew them for measured quality or cost reasons, over organizations that adopted. We found no documented case of such a reversion in the sources reviewed (cutoff 2026-09-01).
Memory benchmark validity	An audited benchmark with a corrected answer key and a judge whose false-accept rate is measured. The most-used benchmark has 6.4% wrong gold answers and a judge accepting 62.81% of deliberately wrong answers [38].
Retrieval for code	A comparison with a fixed metric and independent replication. The one direct comparison located finds no dominant method across 427 samples in 25 repositories [41].
Effective context	Per-model effective context published alongside advertised context. At present 11 of 13 models claiming at least 128K fall below half their short-context baseline at 32K [33].
Injection defense	A defense with a measured residual attack rate that does not break task success. Present benchmarks measure susceptibility, not remedy [55, 18].
Cost of provenance at scale	Provenance overhead measured across implementations. The one first-party measurement here is 52% of query time and 96.3% of wire bytes, on one system.
Accountability for agent-authored code	A regulatory or standards provision that names it. A high-level summary of the European AI Act, updated 31 August 2026, describes four risk tiers and no provision specifically naming AI-generated code or coding agents as a risk category [2].
Unrecorded effects	A detection method that does not rely on the agent's own trace. Recording effects at explicit boundaries audits the invocations the system knows about and cannot see one it never recorded.

Table 6: Each open problem is paired with the measurement that would close it.

extraction, model training data preparation, and any workload whose unit is a large result set are better served by a conventional warehouse query with dataset-level lineage [34] than by per-field provenance that cannot be delivered at that size.

When the state is configuration rather than memory. Build commands, conventions, and project layout are configuration, not memory, under Definition 2.2, because they are always loaded in full. A file loaded at session start is the right mechanism for them [6, 1]. Putting them behind a retrieval function adds a failure mode, a latency cost, and the chance that the instruction the engineer most needed did not rank in the top ten.

Three documented restrictions are often cited as reverts, and their actual scope is narrower. A game engine foundation banned AI-generated pull requests in mid-2026, and a language foundation adopted a comparable earlier restriction. The stated reason in the first case was reviewer burden and

mentorship economics rather than code quality, because reviewer feedback on an agent's patch does not train a future human maintainer. A widely used networking library restricted its bug-bounty intake after the confirmed-vulnerability rate fell from above 15% to below 5%, with roughly 20% of submissions being low-quality AI-generated reports by mid-2025.

All three restrict agent-generated *contributions* from outside. None reverses agent *use* inside an organization, and they should not be cited as if they did. We found no documented case of an organization that adopted coding agents broadly and then reverted for measured quality or cost reasons in the sources reviewed (cutoff 2026-09-01).

12 Limitations

The case studies are one lineage. This is the limitation with the widest reach. Four code bases by one author, who is the author of this series, are four looks at one set of design instincts. Every repository claim in this Part is an existence proof and none is evidence about prevalence. Agreement between them should be discounted to zero.

Implementation-level claims are not behavioural claims. Every statement about a code base here comes from reading source at a pinned commit. Each says that the implementation at that commit performs a given operation, never that a running deployment was observed doing so. Where a repository's own documentation reports a measurement, the claim is attributed to that documentation and inherits whatever measurement discipline the repository applied. In at least one case that discipline includes a continuous-integration history of four runs and four failures.

Intervention moves rather than vanishes. Part I's intervention indicator identifies task classes in which human intervention counts do not fall over time. This Part's own evidence supplies a candidate: reconciling records across systems of record that share no key. The governed data layer converts that task from a silent wrong merge into a typed refusal. That improves safety and does nothing for autonomy, because a human must supply the crosswalk every time. Nothing in the substrate described here reduces the number of human decisions in that class, and the case study's 0 of 2,224 is what that looks like in practice. This is redefined engineering work: the human supplies governed mappings and acceptance decisions rather than performing every record operation directly.

The measurement gaps bound efficiency estimates. Three quantities in Table 6 would quantify verification cost, human intervention, and security risk, and none of the sources we located measures them. The shift in engineering responsibility is visible in execution records, adoption volume, review artifacts, and practitioner reports; the missing measurements determine how efficiently organizations can implement it.

The security section reports susceptibility, not risk. The two injection benchmarks measure attack success under constructed conditions. Neither establishes a rate of exploitation in the field, and the one field incident described is a single case. A reader should not convert 24% on a benchmark into an expectation about a deployment.

Two primary sources could not be fetched directly. Two of the six entries in Table 1 are social posts that returned errors on direct retrieval. Both are cited through a practitioner weblog that quotes them [47]. Their dates and wording are consistent across secondary sources.

13 Conclusion

Two bounds explain why the substrate under agentic engineering exists at all: the length of context over which a model holds accuracy, and the length of task it completes reliably. Both are measured, both are moving, and neither is close to irrelevant. Every mechanism in this Part answers one of them.

Context engineering is the decision about which tokens occupy a bounded, position-sensitive window, and it is engineering only when it carries a budget. Persistent memory is a fold over an immutable event history, which splits the design question in two and turns the difference between an appending derivation and an in-place update into a guarantee rather than a preference. Governed data is a typed query evaluated at a boundary before any source is read, useful to an agent only when a refusal is structurally distinguishable from an empty answer.

The first-party evidence records what these designs cost. A memory layer with rigorous typed ingestion has no token budget on retrieval. A data layer with 35 typed refusal variants, whose provenance carries every observation it considered, merges zero of 2,224 known-true pairs, spends 52% of query time on provenance, and records twelve internal instances of the defect its design exists to prevent.

Six activities follow for the engineer: write contracts a program can decide, choose the bounds and the gates, assign ownership of shared state, budget the context, curate the memory, and enumerate what the system must refuse. The evidence that this is the right list is mechanistic. The evidence that practitioners are not yet doing it is measured, and includes the finding that none of fifteen professional engineers specified security requirements in a prompt even when they knew what to ask for.

Three measurements would settle whether the shift is worth it: verification cost against authoring cost, human intervention counts, and organizational reversion. None of the sources we located reports any of them. Until one does, the argument of this series rests on mechanism, on cases, and on being explicit about which is which.

References

[1] AGENTS.md. Open format for agent instruction files, stewarded by the Agentic AI Foundation under the Linux Foundation. `https://agents.md/`, accessed 2026-09-02.

[2] Future of Life Institute. High-level summary of the AI Act. Explainer of primary legislation, not the legislative text. `https://artificialintelligenceact.eu/high-level-summary/`, updated 31 August 2026.

[3] Anthropic. Compaction. Platform documentation, accessed 2026-09-02. `https://platform.claude.com/docs/en/build-with-claude/compaction`.

[4] Anthropic. Effective context engineering for AI agents. Engineering blog, 29 September 2025. `https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents`.

[5] Anthropic. Memory tool. Platform documentation, tool type `memory_20250818`, accessed 2026-09-02. `https://platform.claude.com/docs/en/agents-and-tools/tool-use/memory-tool`.

[6] Anthropic. How Claude remembers your project. Claude Code documentation, `https://code.claude.com/docs/en/memory`. Accessed 2026-09-02.

[7] Baltes, S., Speith, T., Chiteri, B., Mohsenimofidi, S., Chakraborty, S., Buschek, D. On the need to rethink trust in AI assistants for software development: a critical review. *IEEE Transactions on Software Engineering*. arXiv:2504.12461.

[8] Bappy, F.H., Hossain, T., Meheraj, S.M., Akhand, A.S., Tabassum, T., Zaman, T.S., Hasan, R., Islam, T. From preventive to reactive: how AI coding assistants transform developers' security awareness. *Symposium on Usable Privacy and Security (SOUPS)*, 2026. arXiv:2605.23130.

[9] Becker, J., Rush, N., Barnes, E., Rein, D. Measuring the impact of early-2025 AI on experienced open-source developer productivity. METR research report; preprint, arXiv:2507.09089, 2025. $N = 16$ developers, 246 tasks in mature repositories they knew well.

[10] Beck, K. Augmented coding: beyond the vibes. Newsletter, 25 June 2025. `https://newsletter.kentbeck.com/p/augmented-coding-beyond-the-vibes`.

[11] Brahman, F., Kumar, S., Balachandran, V., Dasigi, P., Pyatkin, V., Ravichander, A., Wiegreffe, S., Dziri, N., Chandu, K., Hessel, J., Tsvetkov, Y., Smith, N.A., Choi, Y., Hajishirzi, H. The art of saying no: contextual noncompliance in language models. *NeurIPS Datasets and Benchmarks*, 2024. arXiv:2407.12043.

[12] Cemri, M., Pan, M.Z., Yang, S., Agrawal, L.A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J.E., Stoica, I. Why do multi-agent LLM systems fail? Preprint, arXiv:2503.13657, 2025.

[13] Chase, H. The rise of "context engineering". LangChain blog, 23 June 2025. `https://www.langchain.com/blog/the-rise-of-context-engineering`.

[14] Cheney, J., Chiticariu, L., Tan, W.-C. Provenance in databases: why, where and how. *Foundations and Trends in Databases* 1(4), 2009, pages 379 to 474. DOI 10.1561/1900000004. Print monograph; the DOI did not resolve on the access date and no wording is quoted from it here.

[15] Chhikara, P., Khant, D., Aryan, S., Singh, T., Yadav, D. Mem0: building production-ready AI agents with scalable long-term memory. Vendor-authored preprint, arXiv:2504.19413, 2025.

[16] Cube. Introduction. Product documentation, vendor positioning rather than a measured result. `https://docs.cube.dev/docs/introduction`, accessed 2026-09-02.

[17] dbt Labs. About MetricFlow. Developer documentation, `https://docs.getdbt.com/docs/build/about-metricflow`. Accessed 2026-09-02.

[18] Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., Tramèr, F. AgentDojo: a dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. Preprint, arXiv:2406.13352, 2024.

[19] Faros AI. Ten takeaways from the AI Engineering Report 2026: the acceleration whiplash. Vendor telemetry, 22,000 developers and 4,000+ teams, within-organization design, no randomization. `https://www.faros.ai/blog/ai-acceleration-whiplash-takeaways`.

[20] Geifman, Y., El-Yaniv, R. Selective classification for deep neural networks. Preprint, arXiv:1705.08500, 2017.

[21] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., Fritz, M. Not what you've signed up for: compromising real-world LLM-integrated applications with indirect prompt injection. Preprint, arXiv:2302.12173, 2023. The arXiv record lists no venue.

[22] Gutiérrez, B.J., Shu, Y., Gu, Y., Yasunaga, M., Su, Y. HippoRAG: neurobiologically inspired long-term memory for large language models. *NeurIPS*, 2024. arXiv:2405.14831.

[23] Hassan, A.E., Li, H., Lin, D., Adams, B., Chen, T.-H., Kashiwa, Y., Qiu, D. Agentic software engineering: foundational pillars and a research roadmap. Preprint, arXiv:2509.06216, 2025.

[24] Hong, K., Troynikov, A., Huber, J. Context rot: how increasing input tokens impacts LLM performance. Chroma technical report, 14 July 2025. Not peer-reviewed; the publisher sells a vector database. `https://trychroma.com/research/context-rot`.

[25] Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., Ginsburg, B. RULER: what's the real context size of your long-context language models? *COLM*, 2024. arXiv:2404.06654.

[26] Kwa, T., West, B., Becker, J., Deng, A., Garcia, K., Hasin, M., Jawhar, S., Kinniment, M., Rush, N., Von Arx, S., Bloom, R., Broadley, T., Du, H., Goodrich, B., Jurkovic, N., Miles, L.H., Nix, S., Lin, T.,

Painter, C., Parikh, N., Rein, D., Sato, L.J.K., Wijk, H., Ziegler, D.M., Barnes, E., Chan, L. Measuring AI ability to complete long software tasks. Preprint, arXiv:2503.14499, 2025. The 212-day doubling figure and its interval are taken from the paper body, not the abstract.

[27] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. *NeurIPS*, 2020. arXiv:2005.11401.

[28] Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P. Lost in the middle: how language models use long contexts. *Transactions of the Association for Computational Linguistics*, 2024. arXiv:2307.03172.

[29] Liu, Y., Widyasari, R., Zhao, Y., Irsan, I.C., Chen, J., Lo, D. Debt behind the AI boom: a large-scale empirical study of AI-generated code in the wild. Preprint, arXiv:2603.28592, 2026. Observational, no randomized control.

[30] Maharana, A., Lee, D.-H., Tulyakov, S., Bansal, M., Barbieri, F., Fang, Y. Evaluating very long-term conversational memory of LLM agents. Preprint, arXiv:2402.17753, 2024.

[31] Model Context Protocol. Security best practices. Specification revision 2025-11-25, `https://modelcontextprotocol.io/specification/2025-11-25/basic/security_best_practices`.

[32] Mei, L., Yao, J., Ge, Y., Wang, Y., Bi, B., Cai, Y., Liu, J., Li, M., Li, Z.-Z., Zhang, D., Zhou, C., Mao, J., Xia, T., Guo, J., Liu, S. A survey of context engineering for large language models. Preprint, arXiv:2507.13334, 2025.

[33] Modarressi, A., Deilamsalehy, H., Dernoncourt, F., Bui, T., Rossi, R.A., Yoon, S., Schütze, H. NoLiMa: long-context evaluation beyond literal matching. Preprint, arXiv:2502.05167, 2025. Industrial authors evaluating third-party models.

[34] OpenLineage. Documentation. LF AI and Data Foundation graduate project, version 1.53.0, accessed 2026-09-02. `https://openlineage.io/docs/`.

[35] OWASP Gen AI Security Project. OWASP Top 10 for LLM applications, 2025 edition. Community consensus standard, not peer-reviewed or regulatory. `https://genai.owasp.org/llm-top-10/`.

[36] Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S.G., Stoica, I., Gonzalez, J.E. MemGPT: towards LLMs as operating systems. Preprint, arXiv:2310.08560, 2023.

[37] Park, J.S., O'Brien, J.C., Cai, C.J., Morris, M.R., Liang, P., Bernstein, M.S. Generative agents: interactive simulacra of human behavior. *ACM UIST*, 2023. arXiv:2304.03442. The venue is taken from the ACM record; the arXiv page lists none.

[38] Penfield Labs. We audited LoCoMo: 6.4% of the answer key is wrong and the judge accepts up to 63% of intentionally wrong answers. Independent third-party audit, 4 April 2026. `https://dev.to/penfieldlabs/we-audited-locomo-64-of-the-answer-key-is-wrong-and-the-judge-accepts-up-to-63-of-intentionally-33lg`.

[39] Moreau, L., Missier, P. (eds.). PROV-DM: the PROV data model. W3C Recommendation, 30 April 2013. `https://www.w3.org/TR/prov-dm/`.

[40] Lebo, T., Sahoo, S., McGuinness, D. (eds.). PROV-O: the PROV ontology. W3C Recommendation, 30 April 2013. `https://www.w3.org/TR/prov-o/`.

[41] Qin, B., Xie, Y. Agent Retrieval Bench: evaluating repository context retrieval for coding agents. Preprint, arXiv:2607.24882, 2026. Single paper, no independent replication.

[42] Rajpurkar, P., Jia, R., Liang, P. Know what you don't know: unanswerable questions for SQuAD. *ACL*, 2018. arXiv:1806.03822.

[43] Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., Chalef, D. Zep: a temporal knowledge graph architecture for agent memory. Vendor-authored preprint, arXiv:2501.13956, 2025.

[44] Spracklen, J., Wijewickrama, R., Sakib, A.H.M.N., Maiti, A., Viswanath, B., Jadliwala, M. We have a package for you! A comprehensive analysis of package hallucinations by code generating LLMs. *USENIX Security Symposium*, 2025. arXiv:2406.10279.

[45] Thoughtworks. Codebase cognitive debt. Technology Radar Volume 34, caution ring, April 2026. https://www.thoughtworks.com/radar/techniques/codebase-cognitive-debt.

[46] Thoughtworks. Coding throughput as a measure of productivity. Technology Radar Volume 34, caution ring, April 2026. https://www.thoughtworks.com/radar/techniques/coding-throughput-as-a-measure-of-productivity.

[47] Willison, S. Context engineering. Weblog, 27 June 2025. https://simonwillison.net/2025/Jun/27/context-engineering/.

[48] Willison, S. The lethal trifecta for AI agents: private data, untrusted content, and external communication. Weblog, 16 June 2025. https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/.

[49] Willison, S. Conceptual integrity and counting lines of code. Weblog, 19 August 2026. https://simonwillison.net/2026/Aug/19/conceptual-integrity-and-counting-lines-of-code/.

[50] Wiz Research. s1ngularity: supply chain attack leaks secrets on GitHub. Vendor security research, 26 August 2025. https://www.wiz.io/blog/s1ngularity-supply-chain-attack.

[51] Wu, D., Wang, H., Yu, W., Zhang, Y., Chang, K.-W., Yu, D. LongMemEval: benchmarking chat assistants on long-term interactive memory. *ICLR*, 2025. arXiv:2410.10813.

[52] Xiao, G., Tian, Y., Chen, B., Han, S., Lewis, M. Efficient streaming language models with attention sinks. *ICLR*, 2024. arXiv:2309.17453.

[53] Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., Zhang, Y. A-MEM: agentic memory for LLM agents. Preprint, arXiv:2502.12110, 2025.

[54] Yu, H., Liu, L., Jiang, X., Jia, Y., Wang, S., Qian, P., Chen, Y. Habituation at the gate: rising approval and declining scrutiny in human review of AI agent code. Preprint, arXiv:2606.22721, 2026. The authors disclaim causal interpretation.

[55] Zhan, Q., Liang, Z., Ying, Z., Kang, D. InjecAgent: benchmarking indirect prompt injections in tool-integrated large language model agents. Preprint, arXiv:2403.02691, 2024.

[56] Zhang, F., Chen, B., Zhang, Y., Keung, J., Liu, J., Zan, D., Mao, Y., Lou, J.-G., Chen, W. RepoCoder: repository-level code completion through iterative retrieval and generation. *EMNLP*, 2023. arXiv:2303.12570.

[57] Zhang, Z., Bo, X., Ma, C., Li, R., Chen, X., Dai, Q., Zhu, J., Dong, Z., Wen, J.-R. A survey on the memory mechanism of large language model based agents. Preprint, arXiv:2404.13501, 2024.