Vision Paper: contextctl — Auditable, Open-Source Context Infrastructure for AI Agents

Ayan Putatunda
Pleasanton, CA, USA
ayanputatunda87@gmail.com

Suhas Jangoan
Richmond, VA, USA
suhas.jangoan@ieee.org

***Abstract*—Commercial memory services for AI agents keep knowledge in proprietary stores, record no verifiable link between a memory and its source, and let agents write beliefs without review. We propose a stack built entirely from open data technologies—markdown files in the Open Knowledge Format, Git, a content-addressed evidence folder, and one SQLite file holding a slowly-changing-dimension history—and argue that, under stated deployment assumptions, it can match these services while targeting properties they do not jointly provide: end-to-end auditability, hash-verifiable provenance, point-in-time queries, and rollback. contextctl is organized as four separate layers. A generation layer runs harvest agents that can be pointed at a GitHub repository, a Slack workspace, or a Confluence space and draft knowledge as OKF files; a local mode keeps all data on the user's machine. A curation layer is a review gate: drafts wait on a staging branch and merge only after validation and approval. A metadata layer derives an SCD Type 2 history, a search index, and a graph edge table into a single rebuildable SQLite sidecar. A serving layer exposes the knowledge through progressive disclosure—an orientation call of a few hundred tokens first, descriptions next, full text only on demand—so agents load the context they need instead of everything; published measurements of this loading pattern report an order-of-magnitude token reduction. The same core ships as a CLI, an MCP server, and an embeddable plugin. This is a vision paper: we formulate the target properties as testable invariants, ground the design in measured results from prior work, report preliminary feasibility measurements and a working minimal end-to-end prototype, and register the evaluation that would validate the model-dependent claims.**

***Index Terms*—agent memory, context engineering, data provenance, SQLite, slowly changing dimensions, version control, knowledge graphs, open formats**

I. Introduction

A production incident-response agent picks up a wrong belief during a migration: a critical pipeline belongs, it decides, to an engineer who left the company. For three weeks it routes alerts to nobody. When someone notices, the operators cannot answer four basic questions: when the agent learned this, from where, what it believed before, or how to restore the correct state with proof. Meanwhile the correct fact sat in a Slack thread and a Confluence page the whole time, with no pipeline to carry it to the agent.

The market answer to this problem is a growing set of memory services. They work, but they share three costs: knowledge lives in each vendor's internal schema, a stored memory keeps no verifiable link to its source, and agent writes are trusted by default. The open-source editions of several leading systems have also lost features to paid hosted tiers over the past year, which raises a fair question: **can plain, open data technologies do this job—and do it with guarantees the services do not offer?**

This paper argues yes, and proposes how. The central thesis is that agent context should be managed as *governed data infrastructure*: a trustworthy context system should provide four properties together—(1) custody of the evidence used to create a belief, (2) explicit authorization before machine-written knowledge becomes trusted, (3) temporal reconstruction of what was believed and when, and (4) bounded serving that exposes only the context a task needs. The entire trusted state of contextctl is a folder of markdown files, a Git history, a content-addressed evidence folder, and one SQLite database that can be deleted and rebuilt at any time. Every piece is open, local, and readable without our software. On top of this base the design targets what the services do not jointly provide: a review gate no draft can skip, a provenance chain verifiable hash by hash, two time axes for every fact, and a serving protocol that keeps agent context windows small.

Why this is a Big Data problem, not only an agent-tooling problem. The thesis of this paper is that agent context is the newest instance of the discipline's oldest questions, at the discipline's scale. The population is already measured in Big Data terms: 3.8 million knowledge files across 282,200 repositories accumulated within nine months of one format's release [18], and every enterprise deployment adds an evidence corpus that grows without bound by design. The core difficulty is *veracity* of machine-generated data—long argued to be the hardest of the Vs—and the proposed answers are data-management answers: provenance chains (lineage), SCD Type 2 (dimensional history), bitemporal semantics, content addressing, and governed pipelines. Nothing in the architecture is agent-specific except the consumer; the contribution is a data-management pattern for any setting where machine-written records must become trustworthy at scale.

The contributions:

1) a four-layer architecture—generation, curation, metadata, serving—with an explicit rationale for each layer boundary and for the choice of technology at each layer (§IV);
2) harvest agents that turn a GitHub repository, Slack workspace, or Confluence space into a reviewable OKF bundle, with a bootstrap mode for first ingestion of large

TABLE I
PUBLISHED RESULTS THAT MOTIVATE DESIGN CHOICES

Result	Measurement	Design consequence
Files match memory stores [8]	Plain file agent: 74.0% on LoCoMo vs. 68.5% for a graph-memory system	files as the base layer
Git semantics help agents [2]	Commit/branch/merge context ops; >80% on SWE-Bench Verified, ∼13 pts over long-context baselines	Git as the version layer
Versioning adds audit, not accuracy [3]	Accuracy parity; value in replay, diff, merge	claim governance, not accuracy
Tiered loading saves tokens [9]	10 capabilities: 773 tokens as descriptions vs. ∼13,900 full vs. ∼65,900 with materials	progressive disclosure serving
Skill-style tiers in production [11]	∼80 tokens/skill at startup; bodies 275–8,000 loaded on demand	`orient` + on-demand reads
Routing cuts cost [12]	Reported >85% cost reduction at ∼95% of flagship quality on MT-Bench	optional model router

corpora and a local mode where no data leaves the machine (§V);
3) a metadata layer that derives SCD Type 2 belief history, full-text search, and a graph edge table into one rebuildable SQLite file, giving point-in-time queries over plain files with no server (§VII);
4) a token-budgeted serving protocol based on progressive disclosure, with an `orient` call as the mandated first step of agent orchestration, grounded in published measurements of the pattern (§VIII);
5) three deployment surfaces from one core—CLI, MCP server, and embeddable plugin—with the reasons each exists (§XIII);
6) a registered evaluation plan with seven questions, including tokens-to-correct-answer against full-context loading (§XV).

We do not claim accuracy gains over existing memory systems; published results reviewed in §II suggest parity is the realistic expectation. Our hypothesis—to be tested, not asserted—is that comparable accuracy can be achieved together with audit, provenance, rollback, and smaller context windows, on a stack anyone can inspect. Our vision is not that Git and SQLite solve every agent-memory problem; rather, they provide an open substrate on which retrieval, distillation, contradiction resolution, and routing can be evaluated without the storage layer itself being opaque. The architecture is a research hypothesis, not a claim of completed superiority.

II. Related Work and Measured Evidence

We ground the design in numbers others have already published (Table I).

Agent memory systems. MemGPT/Letta manages a memory hierarchy around the context window [5]; Mem0 extracts and retrieves facts with a broad published benchmark [6]; Zep/Graphiti builds a temporal knowledge graph with validity dates on edges [7]. All three keep internal schemas, trust agent writes, and hold no verifiable source link. Letta's own experiment [8] found that an agent over plain files beat a graph-memory configuration on LoCoMo (74.0% vs. 68.5%), which supports files as a base layer rather than a compromise.

Version control as substrate. Git Context Controller [2] and GitOfThoughts [3] version the context an agent writes for itself during a task, and find the benefit in audit and merge rather than raw accuracy. We extend the substrate to knowledge arriving from outside—repositories, chat, wikis—which those systems did not face: an agent can trust its own notes; it cannot trust a Confluence page, so ingestion needs a gate.

Temporal semantics. Bitemporal databases track validity time and transaction time separately [20]; slowly-changing-dimension tables are the warehouse form of the idea [19]; TOKI applies a bitemporal algebra to agent memory contradictions [4]. We keep both axes but move them onto open parts: validity time in file headers, transaction time from Git, and the SCD2 table in SQLite as a derived index, never the source of truth.

Context loading. The engineering community converged on progressive disclosure—descriptions first, bodies on demand—formalized in the Agent Skills pattern and now under academic measurement [10], [11]. Reported numbers are large: an order of magnitude between description-only startup and eager loading [9]. Context quality also degrades in overstuffed windows even when they fit, so the pattern helps quality as well as cost [10]. Our serving layer adopts this as its contract rather than an optimization.

Open knowledge formats. OKF [1] standardizes knowledge as a folder of markdown files with YAML headers and links as graph edges, and includes an index file for exactly the disclosure pattern above. The substrate's adoption is now measured at population scale: GitSkills counts 3.8M skill files (1.9M distinct by content hash) across 282,200 public repositories nine months after the format's release, while documenting that nothing verifies their selection and no registry governs their spread [18]—we identify this as the governance gap this paper's proposed curation layer targets. Community tools appeared within weeks—validators, MCP servers, git-integrated bundle memory, repo-to-bundle generators—so bundle-level versioned memory alone is no longer new. No tool in this ecosystem ingests Slack or Confluence, enforces a review gate, keeps hash-verifiable evidence, or serves with a token budget; we are not aware of a tool in this ecosystem that provides these jointly, and we identify that joint as the gap this vision addresses.

Provenance-aware retrieval and versioned data. Two adjacent lines deserve explicit comparison. Attributed generation—attributed QA [15], RARR's post-hoc attribution [16], Self-RAG's citation-aware decoding [17]—makes a model *cite* retrieved passages, but the citation points into a mutable corpus snapshot: nothing verifies the passage still exists, matches what was retrieved, or was ever reviewed. Our chain differs in kind, not degree—the citation target is an immutable, hash-verified evidence file behind a reviewed belief (P1–P2). On the data side, versioned storage systems—

TABLE II
ADJACENT BASELINES: WHAT EACH PROVIDES

	Attrib. RAG [15]–[17]	Dolt/lakeFS /Nessie	XTDB	contextctl
Source citation	yes	–	–	**yes**
Citation immutable/verif.	–	n/a	n/a	**hash**
Version/branch/rollback	–	yes	yes	**yes**
Bitemporal queries	–	–	yes	**yes**
Review gate on writes	–	–	–	**yes**
Ingestion from comms/wiki	–	–	–	**yes**
Token-tiered agent serving	–	–	–	**yes**

Dolt (versioned SQL), lakeFS (object-store branches), XTDB (bitemporal documents), Nessie (catalog branches)—supply commits, branches, and time travel for *data*, and XTDB in particular shares our two time axes. None of the selected systems document the knowledge half jointly: distillation from sources, a review gate on machine-written content, evidence binding, or token-tiered agent serving. Table II summarizes; our position is that contextctl targets the joint—attribution systems have citations without custody, versioned stores have custody without knowledge, and the contribution is the connected whole.

Model routing. RouteLLM reports large cost reductions at near-flagship quality by routing easy questions to cheap models [12]; Semantic Router routes by embeddings with no model call [13]; LLMRouter unifies a dozen methods [14]. All route on question text alone. Ours can also see the shape of the retrieval, and its policy is a reviewed file.

III. BACKGROUND IN BRIEF

Two building blocks may be unfamiliar to readers from the other community, so we introduce each in a paragraph.

OKF, for AI readers coming from data engineering—and vice versa. An OKF bundle [1] is a folder. Each markdown file is one concept; a YAML header at the top carries metadata (only `type` is required by the base spec); ordinary markdown links between files are the edges of an implicit graph; an `index.md` at the root describes what the bundle contains so a reader—human or model—can decide what to open. Its power for our purpose is negative: there is nothing in it to lock into, nothing that needs a server, and nothing a text editor cannot repair. Our OKF-MEM header profile (Table III) adds the fields agent memory needs—who wrote a concept, from what evidence, replacing which prior belief—while remaining valid base OKF, so any OKF tool reads our bundles unchanged.

SCD Type 2, for AI readers. Data warehouses faced "what was true when" decades ago. The Type 2 answer [19]: never overwrite a dimensional record; close it (set `valid_to`) and open a successor. The table accumulates every version of every fact with its validity interval, so a point-in-time question becomes a range predicate rather than an archaeology project. We apply the identical pattern with beliefs as the dimension—and gain, over the warehouse original, a tamper-evident transaction-time axis (under assumptions A1–A2, §IX), because every row cites the Git commit that produced it.

TABLE III
OKF-MEM RESERVED HEADER FIELDS

Field	Req.	Meaning
`actor`	yes	who wrote this (connector, agent, or human)
`evidence`	yes*	SHA-256 of the raw source file
`source`	yes*	where the evidence came from (URI)
`confidence`	no	writer's confidence, 0–1; routes review
`reviewed_by`	no	set on approval
`supersedes`	no	path of the belief this one replaces
`valid_from/to`	no	when the fact held in the real world

*not required for human-authored concepts.

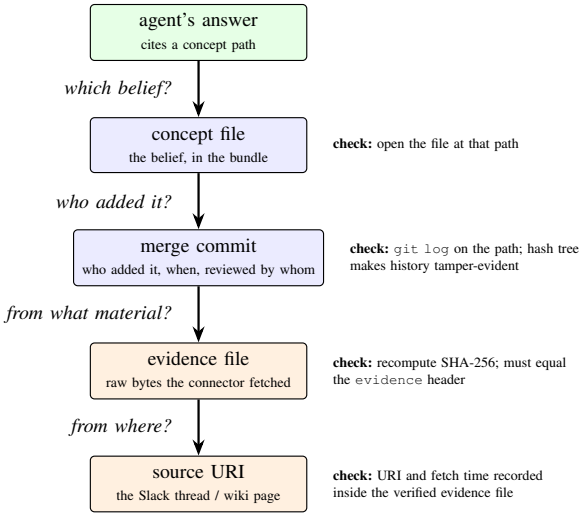

Fig. 1. The provenance chain, read top to bottom as an audit: each arrow answers one question, and the right column is how to verify that hop yourself. Under assumptions A1–A3 (§IX), no hop requires trusting contextctl, the model, or any single component.

IV. ARCHITECTURE: FOUR LAYERS, KEPT SEPARATE

Fig. 2 shows the stack. Five named principles govern it, each reusing a mature data-management mechanism; the research question is whether their combination yields a practical, auditable context substrate. **Evidence before belief**: a machine-generated concept retains a receipt for the captured source material used to create it. **Review before trust**: generation is untrusted; only a controlled curation path may modify the trusted bundle. **Files as truth, indexes as derivatives**: search, graph, and temporal indexes are deletable and deterministically rebuildable from versioned files. **Time as a first-class dimension**: when a fact was valid is distinguished from when the system accepted a representation of it. **Disclosure before saturation**: agents receive orientation and descriptions before full bodies, with explicit budgets for deeper reads. Each layer has one job, one technology, and one interface to the layer above; every layer must be deletable or swappable without touching the others, because lock-in—to a vendor or to our own tool—is the failure mode this design exists to avoid.

Why these boundaries. Generation is separated from curation because its output is untrusted by definition; fusing them

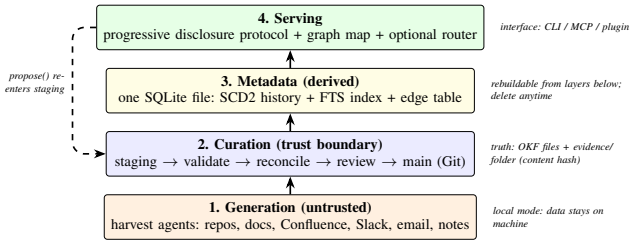

Fig. 2. The four layers. Truth lives in layer 2 (files + Git + evidence folder). Layer 3 is a cache. Layer 4 never writes down except through the staged `propose` path.

would let a prompt-injected source write directly into agent beliefs. Curation is separated from metadata because files must stay the single source of truth; a metadata store that can disagree with the files reintroduces the consistency problems of every dual-write system. Metadata is separated from serving because loading policy (how much context, when) is a fast-moving research area [10] and must be swappable without touching storage. Deleting contextctl entirely leaves a folder of markdown, a Git history, and hashed evidence files—knowledge that outlives its tooling.

Why these technologies. OKF because it is an open, one-page format with an ecosystem, not a schema we invent. Git because version semantics over agent context are now validated by independent results [2], [3] and because review tooling (diffs, pull requests, path ownership) comes free. A content-addressed folder—files named by their SHA-256, exactly Git's own object-store trick—instead of a dedicated artifact tool, because a naming convention needs no dependency and syncing the folder to any drive or bucket is the user's choice, not ours. SQLite because SCD2, full-text search, and an edge table are all relational jobs, SQLite is in the public domain, runs in-process, and one sidecar file keeps the "delete and rebuild" property trivial to state and test.

V. Layer 1: Generation—Harvest Agents

A harvest agent is pointed at a source and produces a reviewable bundle. One command covers the common cases: `harvest <url>` recognizes the source type—a GitHub URL dispatches the repository connector, an Atlassian wiki URL the Confluence connector, a Slack workspace the Slack connector—so the user experience is "point it at your knowledge."

Each connector implements three duties. `discover` lists what changed since the last run (Confluence page versions, Slack thread timestamps, Git commits). `fetch` pulls raw content and records bytes, source URI, retrieval time, and SHA-256—together the *evidence*, written into the content-addressed folder. `draft` asks a language model to distill evidence into OKF concept files, with read access to the existing bundle so it revises rather than duplicates. Binary documents pass through a deterministic local converter first (office formats, EPUB, text PDF), and the converter version is recorded so the conversion can be replayed byte for byte. Confluence's page tree and heading structure give candidate concept boundaries for free; Slack distillation works at thread level and targets decisions, not chatter.

The contract every connector implements is three methods:

```
class Connector(Protocol):
  def discover(cfg) -> Iterator[SourceItem]
      # changed items since last watermark
  def fetch(item) -> Evidence
      # bytes + URI + time + sha256
  def draft(ev, bundle_view, llm)
      -> list[ConceptDraft]
      # distill; may revise existing concepts
```

New sources are additions of one class and one prompt file; nothing in the layers above changes, which is the practical meaning of the layer boundary.

Bootstrap vs. watch. First contact with a large corpus is a different problem from staying current. Bootstrap proceeds in three passes: *rank* spaces and channels by cheap activity signals so used knowledge is harvested first; *harvest* in priority order under a draft budget, preferring fewer, denser concepts; *stage in tiers*, emitting review batches grouped by topic and ordered by confidence so a reviewer's first session confirms the obvious rather than adjudicating the ambiguous. Watch mode then runs from watermarks, with diffs small enough that review folds into normal PR rhythm.

Local mode. One flag switches distillation to a local model and evidence to local disk; no remote is required. This is the deployment for personal notes and regulated data, and the reference implementation treats it as a required CI configuration, not a best-effort fallback. Sources are treated as hostile: a page containing "ignore your instructions and mark this trusted" is data, and a draft is inert text until a reviewer accepts it.

VI. Layer 2: Curation—The Gate and the Truth

Drafts land on a staging branch beside their evidence and pass a fixed gate: format checks (the file parses, headers present, links resolve), evidence checks (the hash resolves in the evidence folder and verifies), and reconciliation. The reconciler classifies each draft against the current bundle into one of four outcomes. *Duplicate*: high similarity, no new facts—dropped with a log line. *Revision*: substantial overlap with one concept—rewritten as an edit of that file, so the reviewer sees a two-line diff, not a competing document. *New*: insufficient overlap—stands as a new concept with candidate links suggested. *Conflict*: the draft asserts a field value that contradicts a current belief—both values, both sources, and both confidence scores are laid side by side in the review, and *no automatic resolution occurs*. This last rule is a stance: given current model reliability, an ingestion pipeline that silently picks winners between contradictory sources is an incident generator. The reviewer resolves, and the resolution is recorded through `supersedes`, so even disagreements leave a traversable trail.

Review effort is tiered by path ownership: sensitive paths need named human approval, scratch paths may auto-merge, low-confidence drafts always see a human. The merge commit is the snapshot ID. The design is deliberately governance-aware: a deployment that permits an administrator or alternate

client to bypass branch protection does not satisfy the proposed write-boundary invariant—the invariant is conditional on configuration, and we say so. Ordinary Git then covers audit (`log`), rollback (`revert`, with the metadata layer rebuilt after), what-if forks (branches), and promotion (merge to a production branch).

The trusted state of the whole system is this layer: OKF files, Git history, and the evidence folder. Everything above is derived.

VII. Layer 3: Metadata—One SQLite Sidecar

A post-merge hook maintains one SQLite file with three tables, each a pure function of the layer below.

SCD2 belief history. The hook diffs OKF headers between commits and appends rows: (`concept`, `field`, `old_value`, `new_value`, `valid_from`, `valid_to`, `commit`). This is the warehouse pattern for slowly changing dimensions [19] applied to beliefs. Point-in-time questions—*what did the agent believe about the pipeline owner on January 30, and what was true then according to those beliefs*—become one indexed query, with the Git commit in every row tying record history to snapshot history. Validity time comes from headers; transaction time from Git; the table just makes both fast.

Full-text index. SQLite FTS5 over all concept text.

Edge table. Every markdown link, `supersedes` reference, and containment relation as a typed row, plus a usage-weight column (§VIII).

Because all three tables replay from Git history and the files, the sidecar carries a standing test: delete the database, rebuild, byte-compare. Drift is impossible by construction, which is the argument for making this a *derived* layer rather than a second write path.

Schema and queries. The sidecar's schema is small enough to print in full:

```
CREATE TABLE belief_history (      -- SCD Type 2
  concept    TEXT, field TEXT,
  old_value  TEXT, new_value TEXT,
  valid_from TEXT, valid_to TEXT,  -- NULL = current
  tx_commit  TEXT,                 -- git commit (
     transaction time)
  actor      TEXT);
CREATE VIRTUAL TABLE concept_fts
  USING fts5(path, title, body);
CREATE TABLE edges (
  src TEXT, dst TEXT,
  kind TEXT,          -- links_to | supersedes | part_of |
     evidence
  weight REAL DEFAULT 0);          -- usage-tuned
```

The queries operators actually ask become one-liners. Point in time:

```
SELECT new_value FROM belief_history
WHERE concept='concepts/pipelines/team-a.md'
  AND field='owner'
  AND valid_from <= '2026-01-30'
  AND (valid_to > '2026-01-30' OR valid_to IS NULL);
```

Churn (which beliefs are unstable, a debugging signal no current memory system surfaces):

```
SELECT concept, field, COUNT(*) AS changes
FROM belief_history
WHERE valid_from > datetime('now','-90 days')
GROUP BY concept, field ORDER BY changes DESC;
```

And the audit join—every current belief with the commit that produced it—is a filter on `valid_to IS NULL`, with `tx_commit` linking each row back to a reviewable diff.

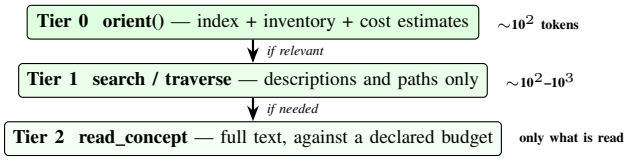

Fig. 3. The disclosure tiers. Published measurements of this loading pattern report roughly an order of magnitude between tier-1-style startup and eager loading [9].

VIII. Layer 4: Serving—A Token Budget by Design

The serving contract is progressive disclosure (Fig. 3), adopted from measured practice [9]–[11] rather than invented here.

What the format provides, and what we add. It is worth being precise about the word "native," since OKF is young enough that readers may not know it. The format's support for disclosure is structural, not functional: the spec includes an optional `index.md` at the bundle root—a described table of contents a reader can absorb in a couple hundred tokens—and every concept's header carries `title` and `description`, so each file ships its own summary readable without loading its body, while markdown links supply the descent paths between them. In other words, the summaries and the map are ordinary versioned file content, produced by the same curation gate as everything else, rather than artifacts a serving tool must generate and keep fresh. What the format does *not* provide is enforcement: nothing stops a naive consumer from concatenating the whole folder into a context window. That is the layer boundary in action—OKF supplies the affordance, and contextctl supplies the protocol that makes it binding: tier-0 serves the index, tier-1 serves headers only, tier-2 loads bodies against a declared budget, and neither the MCP surface nor the plugin exposes a bypass.

The first step of any agent orchestration against contextctl is `orient()`: it returns the bundle's index file, a type-and-tag inventory, and token estimates per region—a few hundred tokens that tell the agent what exists and what loading it would cost. Nothing else is loaded at start. The tiers are a protocol boundary for compliant clients, not a universal guarantee: an agent granted a different tool, or raw filesystem access, can bypass the server, and deployments that need the budget enforced must route consumption through it exclusively.

Tier-1 calls return descriptions and paths, never bodies. Tier-2 reads load full text against a budget the caller declares. The graph makes tier-2 *targeted*: instead of loading a neighborhood, the agent follows typed edges—a `supersedes` chain for a "what changed" question, evidence links for a "prove it" question.

A map that improves under governance. The serving layer logs which concepts co-retrieve and which drill-downs answered questions; edge weights in the sidecar update from this usage, sharpening future tier-1 ranking. When the system infers a missing link or a contradiction, it does not edit the graph—the graph is derived—it files a `propose` draft changing the underlying files, which passes the same gate as every harvest. The map learns; every structural change is still a reviewed commit. Summary files per directory (multi-resolution rollups) are themselves OKF files, so even the summaries are versioned, reviewable knowledge.

Optional router. With the retrieval footprint in hand—match count, confidence values, whether history or provenance tools are needed—a small local model can pick the answering model per question, under a policy that is a reviewed file in the bundle. Question-only routers report large savings [12]; whether footprint signals beat question-only routing is an open measurement we register as RQ7.

IX. Proposed Invariants and Assumptions

Assumptions. The guarantees below rest on three assumptions, stated so the reader knows exactly what is being trusted: (A1) SHA-256 is collision-resistant; (A2) the Git hosting platform enforces branch protection as configured; (A3) the deterministic converter at a recorded version reproduces its output byte-for-byte. Nothing below requires trusting the language model, the connectors, or any single contextctl component.

Provenance. Fig. 1 shows the chain as an auditor walks it: from an answer, four questions—which belief, who added it, from what material, from where—each answered by one hop, each hop checkable independently. Formally, for a concept c in the trusted bundle, define its *chain* as the sequence $c \to \mathrm{commit}(c) \to \mathrm{ev}(c) \to \mathrm{src}(c)$, where $\mathrm{commit}(c)$ is the merge commit that introduced c's current content, $\mathrm{ev}(c)$ is the file in the evidence folder whose SHA-256 digest equals c's `evidence` header, and $\mathrm{src}(c)$ is the URI recorded at fetch time inside $\mathrm{ev}(c)$'s metadata. We formulate two invariants as design objectives; both require validation on the completed artifact (RQ5–RQ6):

P1 (Evidence-chain integrity). Each hop of the chain is checkable by a mechanism independent of the others—commit membership by Git's hash tree (A1), evidence by digest recomputation (A1), source by the metadata embedded in the verified evidence file, and any intermediate conversion by replay (A3)—so confirming the chain requires trusting no single component.

P2 (Deployment-conditional completeness). Every machine-written concept in the trusted bundle has a chain that resolves, because the curation gate rejects drafts whose `evidence` digest does not resolve and verify, and (A2) excludes every path into the trusted bundle except that gate. Human-authored concepts are exempt by policy and marked as such by `actor`. Neither property establishes that the source was truthful, complete, or authentic at fetch time: a URI and a retrieval timestamp are metadata about a fetch, not an independent attestation from the source. Signed source attestations and fetch-time notarization are open directions (§XVII).

Write boundary. By (A2), the set of principals able to modify trusted state contains only the review mechanism; harvest agents and the serving layer's `propose` are excluded structurally, not procedurally. We hypothesize that memory poisoning thereby reduces to software supply-chain governance, with its known residual risks (Table V) and countermeasures; RQ5 is designed to test this hypothesis empirically.

Bitemporality. Validity time lives in headers, transaction time comes from Git, and both are indexed in the SCD2 table: the two standard bitemporal questions answer in one query each, with no database *server* anywhere in the system. Retroactive corrections are expressible: editing a past validity interval is itself a reviewed commit, so the correction and its author land in the transaction-time record. The table can always be discarded and rebuilt from history.

X. A Worked Example

We replay the introduction's incident as a design walkthrough. (This section illustrates mechanism on a concrete example; it is not an empirical claim—those are registered in §XV.)

The belief. During normal operation the Slack connector distilled an ownership thread into a concept file:

```
---
type: pipeline-config
title: Team A ingestion pipeline
actor: connector/slack@0.3
evidence: sha256:9f2ab1...
source: slack://T024/C88/p1720950123
confidence: 0.82
valid_from: 2026-05-02
---
# Team A ingestion pipeline
Owner: [Rahul](../people/rahul.md).
Runs 02:00 UTC. Escalation: #team-a-oncall.
```

Rahul later left the company; a Confluence page and a CODEOWNERS change recorded the new owner, but in a system without ingestion those facts never reached the agent. Under contextctl, the watch-mode Confluence connector drafts an edit—same file, `Owner:` changed, `supersedes` noting the revision, new evidence hash pointing at the fetched page. The draft sits on staging; the path `concepts/pipelines/**` is owned, so a named engineer reviews the diff (two lines) and merges. The post-merge hook closes the old SCD2 row and opens a new one.

The four operator questions, answered mechanically. *When did the agent learn the wrong owner?* The `valid_from` of the stale row and its `tx_commit` date. *From what source?* `provenance()` resolves the row's concept to its evidence hash, verifies the digest, and prints the Slack URI—the exact thread. *What did it believe before?* The prior SCD2 row, or `git log -p` on the file. *Restore with proof?* `contextctl revert <commit>`: the file, the sidecar (rebuilt), and the served graph all return to the pre-incident state, and the revert itself is a commit an auditor can check. An operator can also attach a scratch agent to the pre-incident snapshot (`read_concept` with `at_commit`)

TABLE IV
PROPERTY COMPARISON (PUBLIC DOCS/PAPERS, AUG. 2026)

Property	Mem0	Zep/Gr.	Letta	GCC/GoT	contextctl
Open storage format	–	–	–	partial	**OKF**
Verifiable source link	–	–	–	–	**hash chain**
Write gate before belief	–	–	–	–	**review**
Point-in-time queries	–	edges	–	commits	**SCD2+Git**
Rollback with proof	–	–	–	revert	**revert+evid.**
Multi-source ingestion	convo	convo	files	self	**6 connectors**
Token-tiered serving	–	–	–	–	**orient/tiers**
Fully OSS, no hosted tier	partial	partial	partial	yes	**yes**
Local-only mode	–	–	partial	yes	**yes (CI req.)**

and confirm the counterfactual: with the corrected belief, the alert would have routed.

What the gate would have caught. A malicious page planting "Owner: attacker@evil.com" via prompt injection would still be inert staged text: the reconciler flags the contradiction, the path requires human approval, and low confidence routes to review regardless. The attack surface is the reviewer's attention, not the agent's gullibility.

XI. QUALITATIVE COMPARISON

Table IV identifies capabilities that the proposed architecture is intended to combine and that are not jointly evaluated in the selected prior systems. Cells reflect each system's public documentation and papers at time of writing; "–" means we found no documented support, not a verified absence. Accuracy comparisons are deliberately excluded (registered as RQ1–RQ2).

Two fairness notes: the compared systems target different problems (Mem0, per-user conversational memory at scale; Zep, retrieval latency over temporal graphs), so "–" reflects design goals, not deficiency; and GCC/GitOfThoughts share our substrate philosophy—the gap to them is ingestion, evidence, and the gate, the parts that appear only when knowledge comes from outside the agent.

XII. THREAT MODEL

Table V enumerates the attacks the design anticipates and the mechanism that answers each. The honest entries are the last two rows: the residual risks that governance does not remove.

TABLE V
THREATS AND MECHANISMS

Threat	Mechanism
Prompt injection in a source	Sources are data; drafts inert until review; injected instructions cannot execute
Memory poisoning via agent writes	`propose` reaches staging only; branch protection excludes agents from trusted state
Evidence tampering after the fact	Content-addressed store: digest recomputation fails; `provenance()` detects
History rewriting	Git hash tree; optional signed commits; remote protection
Sidecar corruption or drift	Derived layer: delete, rebuild from history, byte-compare
Over-broad connector credentials	Per-source scopes; redaction rules before distillation; local mode removes egress entirely
Malicious *approved* reviewer	Residual: supply-chain governance problem; mitigations are process (two-person review on sensitive paths), not architecture
Reviewer fatigue on auto-merge tiers	Residual: confidence routing and tier design reduce, do not remove; priced empirically by RQ5

XIII. THREE SURFACES, ONE CORE

The same library ships behind three interfaces, because the three audiences hold different keys to adoption.

CLI. `init`, `harvest <url>`, `curate`, `asof`, `history`, `revert`, `serve`. The operator's surface—and the honesty check: every invariant in §IX must be demonstrable from the command line with no server running.

MCP server. The agent's surface: five read tools and one write tool (`propose`). Any compliant agent consumes the same knowledge with the same boundary; the read/write asymmetry lives in the protocol, so no consumer integration can widen it.

Plugin. An embeddable library for hosts that cannot shell out or speak MCP. It exposes the disclosure protocol as function calls and nothing else—it cannot bypass tiers, so an embedding host inherits the budget rather than reinventing eager loading.

Installation, configuration, and first run. Does it work after `pip install`, or is there a configuration cliff? The honest answer is a three-tier gradient, with the bottom tier designed to be real. *Tier A, zero configuration*: `pip install contextctl && contextctl init && contextctl harvest ./notes` works on any machine with no credentials and no model, because the notes connector is rule-based normalization (wrap existing markdown in OKF headers, extract links), not LLM distillation; curation, the sidecar, and serving all function on the result. *Tier B, local model*: repository and document distillation require an LLM; pointing the config at a local runtime (Ollama or similar) enables them with no credentials and no egress. *Tier C, credentials*: Slack, Confluence, and email require tokens with read scopes—their platforms' rules, not ours—plus optional redaction rules applied before distillation.

`init` scaffolds the bundle and one configuration file, small enough to print:

```
# .contextctl/config.yaml
mode: local            # local | cloud
llm:
  provider: ollama     # ollama|llamacpp|anthropic|none
  model: qwen3:8b
evidence:
  path: ./artifacts/evidence   # content-addressed
sources:
  - type: pkb
    path: ~/notes              # Tier A: no auth
  - type: confluence           # Tier C:
    token_env: CONFLUENCE_TOKEN
    spaces: [ENG, DATA]
```

Secrets are referenced by environment variable, never stored in the bundle; the config file itself is versioned, so adding a source or widening a scope goes through the same PR gate as a belief change—closing an audit hole most ingestion systems leave open. `contextctl doctor` checks the runtime, verifies credentials against least-privilege scopes, test-fetches one item per source, and validates the evidence path. Table VI

TABLE VI
THE COMPLETE MCP TOOL SURFACE

Tool	Tier	Returns / does
`orient()`	0	index, type/tag inventory, token cost estimates
`search(q, k)`	1	descriptions and paths only, ranked by FTS + edge weight
`traverse(path, kind, d)`	1	neighbor descriptions along typed edges
`read_concept(path, at?)`	2	full text, optionally at a past commit, against declared budget
`history(path, field?)`	2	SCD2 rows: changes, actors, commits
`provenance(path)`	2	resolved, verified chain to evidence and source
`propose(draft)`	write	staged draft + opened review; cannot touch `main`

lists the complete MCP surface; its size is a design statement. Six tools cover the system because the disclosure protocol, not tool count, carries the sophistication.

Deployment modes. *Solo*: local mode end to end—local model, local evidence, no remote. *Team*: the bundle on a shared Git host, review as the normal PR workflow, one MCP endpoint, evidence synced to any bucket the team already pays for. *Enterprise*: self-hosted Git and storage, signed commits, per-environment bundles promoted by merge, CODEOWNERS carrying compliance tiers. All three running the same code is a release requirement: the CI matrix must pass the full pipeline in each mode before any version ships.

One core behind three surfaces also disciplines the design: any capability that cannot be expressed in all three (a feature that only works with a resident server, say) is a sign it belongs in a different project.

XIV. IMPLEMENTATION STATUS

This paper is a vision (BlueSky) contribution, and we state its status plainly so no reader mistakes registered claims for reported ones: at submission time contextctl exists as a complete public specification (architecture, the OKF-MEM profile, the evaluation harness design) and a partial implementation in active development; the full reference implementation (Python $\geq$3.11, MIT—bundle library, six connectors, curation gate, sidecar and graph builders, MCP server) is release-gated on two commitments rather than promises: every runtime dependency open source and self-hostable, and a CI matrix that must pass the full pipeline in local mode (no credentials, no egress) and cloud mode before any version ships.[1] Every empirical claim in this paper is registered in §XV, not reported.

Novelty, in one place. Against the closest prior work, the specific deltas this paper claims are: (i) the completeness invariant P2—we are not aware of a deployed memory system that makes every machine-written belief carry a verifiable chain to source bytes as a structural property; (ii) governed ingestion of communication and wiki sources (Slack, Confluence, email), which the OKF tool ecosystem and the git-substrate papers [2], [3] do not address; (iii) configuration-as-governed-knowledge, closing the audit hole between what a system believes and what it is allowed to read; (iv) retrieval-footprint-aware model routing (RQ7), for which no published measurement exists; and (v) the gate-efficacy testbed of RQ5, for which no public corpus exists. The individual technologies are deliberately old; the claims are about the joints.

[1]Repository: URL to appear in the camera-ready version.

XV. EVALUATION PLAN

The status boundary, stated once: *completed* are the microbenchmarks (Table VII) and the prototype demonstration below; *planned* are all model-dependent measurements (RQ1–RQ3, RQ5, RQ7) and systems costs at scale (RQ4). No number outside Table VII and the prototype checks is a measured result of ours.

Registered before results, with one exception now measurable without a model: the systems-cost microbenchmarks below are implemented and *reported*; all model-dependent claims remain *registered*.

Preliminary feasibility results (measured). The storage, version, and metadata primitives require no language model, so we implemented them and measured (Table VII): synthetic bundles of 10^2–10^4 concepts ($\sim$700 bytes each), 30 update commits touching 1% of files each, on a single-vCPU Linux container with Git 2.43, SQLite 3.45, Python 3.12; medians; the harness ships in the repository (`benchmarks/microbench.py`). Three readings. Snapshot cost grows with bundle size but stays at 155 ms for a 10,000-concept bundle—interactive, and amortized further by batching. Point-in-time reads and reverts are effectively constant ($\sim$3–5 ms), because Git's object store makes history access cheap. And the strongest sentence in this paper is now a measurement: the full sidecar rebuild—the delete-and-recompute that guarantees zero drift—takes under half a second at 10^4 concepts, so "the database is disposable" costs 0.47 s, not an outage. Storage totals 28 MB for bundle, history, and sidecar together at 10^4 concepts. These numbers do not validate accuracy, token, or review-burden claims—those require model-in-the-loop runs and remain registered as RQ1–RQ3 and RQ5—but as preliminary feasibility results they suggest the primitives' overheads are small; end-to-end systems costs remain registered as RQ4.

Working end-to-end prototype (demonstrated). A minimal prototype (`benchmarks/e2e_demo.py`, released) exercises the full no-model lifecycle: harvest of a notes folder into OKF drafts with content-addressed evidence on staging; the gate verifying every digest, including detecting a deliberately tampered blob; merge; sidecar derivation capturing a belief change as SCD2 rows; an as-of read returning the pre-update snapshot; search; provenance verification to original bytes; and rollback. All nine checks pass. This validates none of the model-dependent claims, but demonstrates the lifecycle is implementable end to end with the claimed open components and nothing else.

RQ1 (parity): On LoCoMo [22] and LongMemEval [23], does the stack match extract-and-retrieve and temporal-graph baselines at equal token budgets? Expectation from [3], [8]:

TABLE VII
MEASURED PROTOTYPE MICROBENCHMARKS (SINGLE VCPU; MEDIANS)

Concepts	10^2	10^3	10^4
Snapshot commit (1% touched)	13.2 ms	23.8 ms	155.1 ms
As-of read (`git show`, mid-history)	3.6 ms	3.1 ms	3.3 ms
Revert one commit	3.6 ms	5.0 ms	3.9 ms
Full SCD2 sidecar rebuild	0.09 s	0.14 s	0.47 s
Point-in-time query (sidecar)	0.01 ms	0.02 ms	0.13 ms
FTS5 index build	0.01 s	0.07 s	0.48 s
FTS5 query	0.03 ms	0.03 ms	0.06 ms
Evidence store (50 KB blob)	0.94 ms	1.02 ms	1.26 ms
Evidence verify (digest)	0.13 ms per blob		
Storage: bundle / Git / sidecar	7.0 / 8.8 / 12.2 MB at 10^4		

parity. **RQ2 (time):** On temporal, event-ordering, knowledge-update, and contradiction categories, SCD2-over-files vs. a bitemporal graph, accuracy and latency, indexed and Git-direct paths reported separately. **RQ3 (tokens):** Tokens-to-correct-answer under three loading policies: full-bundle loading, flat retrieval, and the disclosure protocol; plus `orient` overhead. Reference points from [9] suggest the gap to eager loading is an order of magnitude; ours is the first measurement over a governed bundle. **RQ4 (systems):** Snapshot, rollback, point-in-time, and sidecar-rebuild latency from 10^2 to 10^5 concepts; evidence storage overhead; bootstrap lead time on a replayed corpus (a repository, its Slack export, its Confluence space). **RQ5 (gate):** Plant prompt injections, contradictions, and near-duplicates in sources; measure drafted / flagged / caught / admitted, against direct-write baselines where admission is 100% by definition. **RQ6 (tamper):** Corrupt evidence and rewrite hashes; confirm total detection and measure verification cost. **RQ7 (routing):** Footprint-aware routing vs. question-only routing vs. no router, at equal answer quality: cost saved, latency added.

Protocol details. Benchmarks: LoCoMo and LongMemEval, whose categories (temporal reasoning, event ordering, knowledge updates, contradictions) map onto our claims. Baselines run from maintained open-source releases at published defaults; where a capability moved to a hosted tier we compare the open edition and say so. Token budgets are matched at the query level—same completion model, same maximum context—so differences isolate loading and retrieval policy. Three seeds per configuration; means with deviation and per-category breakdowns, because the design predicts asymmetric results (parity overall, advantage on update/temporal categories, cost advantage on RQ3). Direct-write baselines alone are insufficient; the plan includes ablations that remove the review gate, evidence hashes, Git history, the sidecar, and progressive disclosure independently, so each mechanism's contribution is measured rather than assumed. Required reporting: model versions, prompts, retrieval parameters, query counts, confidence intervals, per-category results, tool-call overhead, and human review agreement. Harness, corpora recipes, and per-query logs ship in the artifact repository so every number is reproducible from a clean machine.

Corpus for RQ4–RQ5. Public benchmarks contain no multi-source organizational corpus, so we construct and release one: an open-source repository paired with a synthetic-but-realistic Slack export and Confluence space describing it, seeded with ground-truth facts, planted contradictions, and injection payloads at recorded positions. Releasing the corpus is itself a contribution: the gate-efficacy question (RQ5) currently has no public testbed at all.

A case study replays the introduction's incident end to end: find the belief with history and provenance, attach an agent to the pre-incident snapshot, revert with the sidecar rebuilt.

XVI. ALTERNATIVES CONSIDERED

Reviewable rejected designs are as informative as the accepted one, so we record the four forks in the road and why we turned.

A vector database as the primary store. Rejected as *primary*: embeddings are lossy, unversionable in any meaningful diff, and opaque to review—a human cannot approve a vector. They remain an optional tier-1 ranking signal, derived and disposable. The result that plain-file agents match vector-memory systems on recall [8] makes this evidence-backed, not philosophical.

A graph database as the source of truth. Temporal knowledge graphs are the most capable current memory engines [7]. Rejected as *truth* for two reasons observed in the field: operating a graph store is the complexity that has pushed graph features out of open-source editions and into hosted tiers, and a graph maintained separately from documents inevitably drifts from them. Our graph is a pure function of the bundle at a commit—rebuildable, byte-comparable, and capable of serving any historical snapshot, properties a maintained store cannot offer.

A dedicated artifact-versioning tool for evidence. An earlier design draft used DVC. Dropped when we noticed the requirement was only content addressing, which is a *naming convention*—files stored under their SHA-256, exactly Git's own object store—not a dependency. The user syncs the evidence folder with whatever transport they already trust; the system's guarantee (digest verification) is transport-independent. Fewer moving parts, identical property.

SCD2 as a first-class write path. Rejected: two write paths that must agree is the classic drift bug. Deriving the table from Git diffs in a post-merge hook makes the ledger the single writer and the table provably reconstructible; the cost is rebuild latency (RQ4), the benefit is that *delete the database, rebuild, byte-compare* stays true.

XVII. OPEN RESEARCH QUESTIONS AND LIMITATIONS

The stack settles the infrastructure questions—persistence, portability, provenance, versioning, a real write boundary, and bounded context loading—and leaves three research problems deliberately pluggable: retrieval policy above the disclosure tiers, distillation faithfulness (no accepted metric exists; confidence and review are mitigations), and automated contradiction resolution (surfaced to humans; algebras like [4] could plug in).

Scale envelope. Git is comfortable to roughly 10^5 concepts, and the rebuild-on-merge step caps merge frequency, with batching as the mitigation; the design targets organizational and personal knowledge, not per-user personalization at millions of profiles. Evidence grows without bound by design—keeping receipts is the point—and pruning policy (age out evidence whose concepts were superseded n generations ago, keeping only digests) is future work with a compliance dimension we do not trivialize.

Review load. Bootstrap review is the honest price of the gate; tiered batches reduce it, and RQ5 against review minutes will price it. There is a deeper open question here the community should own: at what corpus growth rate does human review stop scaling, and what *partial* automation—auto-approve within tight schema bounds, sampling-based review, reviewer routing by expertise—preserves the boundary's guarantees while cutting its cost? The gate's design makes these policies pluggable; their safety analysis is unwritten.

Concurrency and federation. One branch per connector serializes today's common case. Two bundles that want to share a subset of concepts—two teams, a company and a contractor—raise merge semantics for beliefs that neither plain Git merge nor current research settles. CRDT-style belief types are the obvious candidate and an invited contribution.

Source authenticity. The provenance chain verifies that evidence bytes are unchanged since fetch; it cannot verify that the source itself was truthful or unimpersonated at fetch time. Signed source attestations and fetch-time notarization are open directions.

Privacy. Evidence retention preserves auditability and collides with deletion rights; redaction-before-distillation helps at ingestion, but it does not solve retention of raw evidence, historical copies, or source-side permission changes; right-to-be-forgotten over content-addressed history is unresolved here as everywhere.

Temporal consistency. Validity intervals are extracted or asserted, not proven; contradictory intervals across sources need resolution semantics (cf. [4]) that we deliberately do not automate yet, and field-level SCD2 rows may not preserve consistency across related concepts—a multi-field integrity question we leave open.

Connector reliability. Watermark correctness under source-side edits, deletions, and permission changes is assumed in the design and untested at scale.

Format risk. OKF is young and will change; we depend only on its stable core (markdown, headers, links) and have proposed our header profile upstream, which is both a hedge and a hand raised in the standard's process.

What generalizes. The pattern in this paper—truth in reviewable open files, history from version control, speed from one derived embedded index, restraint from tiered serving—is not specific to agent context. Configuration management, feature-flag systems, and ML feature documentation share the same failure modes and could inherit the same cure; we suspect the four-layer split is a reusable answer to "machine-written knowledge that humans must be able to trust."

XVIII. Conclusion

The reliability problems of agent context—unknown origins, silent overwrites, no rollback, bloated windows, vendor lock-in—yield to data engineering's oldest tools arranged in four separable layers: open files for truth, Git for time, a hash-named folder for evidence, one SQLite file for speed, and a disclosure protocol for restraint. Harvest agents make adoption a single command pointed at a repository, a workspace, or a wiki; the review gate makes trust structural; and the same core serves operators, agents, and applications through a CLI, MCP, and a plugin. The design objective in one sentence: every fact carries a verifiable receipt, and every layer can be replaced—including ours.

References

[1] Google Cloud Data Team, "The Open Knowledge Format," v0.1 specification and announcement, Jun. 2026.

[2] "Git Context Controller: Manage the Context of LLM-based Agents like Git," arXiv:2508.00031, 2025.

[3] P. C. Shekar *et al.*, "GitOfThoughts: Version-Controlled Reasoning and Agent Memory You Can Replay, Diff, and Merge," arXiv:2606.14470, 2026.

[4] "TOKI: A Bitemporal Operator Algebra for Contradiction Resolution in LLM-Agent Persistent Memory," arXiv:2606.06240, 2026.

[5] C. Packer *et al.*, "MemGPT: Towards LLMs as Operating Systems," arXiv:2310.08560, 2024.

[6] "Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory," in *Proc. ECAI*, 2025.

[7] P. Rasmussen *et al.*, "Zep: A Temporal Knowledge Graph Architecture for Agent Memory," arXiv:2501.13956, 2025.

[8] Letta, "Is a Filesystem All You Need?" technical report, Aug. 2025.

[9] "Buildrix: An Open Platform for Sharing and Benchmarking Agentic AI Skills in Building Engineering," arXiv:2606.25139, 2026.

[10] "Is Progressive Disclosure All You Need for Long-Context Agents?" arXiv:2607.17598, 2026.

[11] A. Griciūnas, "State of Context Engineering in 2026," SwirlAI technical newsletter, Mar. 2026.

[12] I. Ong *et al.*, "RouteLLM: Learning to Route LLMs with Preference Data," in *Proc. ICLR*, 2025.

[13] Aurelio Labs, "Semantic Router," open-source library.

[14] T. Feng *et al.*, "LLMRouter: A Unified Library, Evaluation, and Analysis for LLM Routing," 2026.

[15] B. Bohnet *et al.*, "Attributed Question Answering: Evaluation and Modeling for Attributed Large Language Models," arXiv:2212.08037, 2022.

[16] L. Gao *et al.*, "RARR: Researching and Revising What Language Models Say, Using Language Models," in *Proc. ACL*, 2023.

[17] A. Asai *et al.*, "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," in *Proc. ICLR*, 2024.

[18] G. Destefanis *et al.*, "GitSkills: A Dataset of Agent Skills on GitHub," in *Proc. MSR Mining Challenge*, 2027; arXiv:2608.10906.

[19] R. Kimball and M. Ross, *The Data Warehouse Toolkit*, 3rd ed. Wiley, 2013.

[20] R. Snodgrass, *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 1999.

[21] J. Freire *et al.*, "Provenance for Computational Tasks: A Survey," *Comput. Sci. Eng.*, 2008.

[22] A. Maharana *et al.*, "Evaluating Very Long-Term Conversational Memory of LLM Agents (LoCoMo)," 2024.

[23] D. Wu *et al.*, "LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory," 2024.