

Outcome-Aware Memory Infrastructure for Reliable Agentic AI Deployment

Kiran Vadagam

Researcher in Applied Meta-Cognition & Decision Science in Humans & Machines/AI

Contexton.ai - ODEFTO Tech - Learning & AI Labs

kiranvadagam@odefto.com

August 2026

Abstract

AI agents in 2026 can reason, plan, and execute — but they struggle to learn from what happened. The dominant agent memory architectures (Mem0, Graphiti/Zep, Cognie, Letta) provide increasingly capable storage, retrieval, temporal reasoning, and feedback mechanisms. However, an open problem remains: To our knowledge, we did not identify an existing open-source system that combines explicit, auditable, per-memory and per-tool reliability scoring with direct retrieval/control semantics in the manner proposed by ContextOn.AI_OSS. When an agent gives a wrong answer, retrieves stale knowledge, or uses a failing tool, existing memory systems treat that knowledge with the same trust as verified truth.

We present ContextOn.AI OSS, an open-source outcome-aware memory infrastructure that addresses this gap. The system provides three capabilities: (1) bounded failure attribution — identifying candidate memories that contributed to a wrong answer using answer-content matching, with blast-radius control to limit collateral damage to correct facts; (2) outcome-aware reliability scoring — per-fact confidence that decreases with verified failure, recovers with verified success, and decays over time; and (3) tool reliability memory — tracking which tools succeed for which tasks across sessions. We formalize the reliability model, describe the knowledge graph architecture, present 33 test cases demonstrating the failure-learning loop, and compare against Mem0, Graphiti, Cognie, and Letta. We position ContextOn as a reliability layer that can sit above existing memory providers — not a replacement, but an infrastructure that increases the value of existing memory investments by adding outcome-aware trust intelligence. We further propose Trust-Bounded Context as a combined architecture with ScopeTree, where authorization (what may this agent access?) and reliability (should it trust this?) form complementary layers of a complete AI deployment governance framework.

Keywords: agent memory, bounded failure attribution, reliability scoring, outcome-aware memory, knowledge graphs, agentic AI, confidence decay, tool reliability, trust-bounded context

I. INTRODUCTION

A. The Problem: Agents That Cannot Learn From What Happened

The 2025–2026 wave of AI agents has demonstrated remarkable capability in reasoning, planning, and tool use. Yet a fundamental limitation persists: agents repeat mistakes.

When an agent answers "PM-JAY is a housing scheme" instead of "PM-JAY is health insurance," the memory system stores this incorrect retrieval with the same confidence as verified knowledge. When the same agent uses a tool that fails 30% of the time, it typically lacks a persistent, explicit reliability signal about those failures that can directly inform future tool selection. When one agent in a multi-agent system makes an error, existing memory mechanisms do not necessarily propagate that failure as an explicit reliability signal to other agents.

The ICML 2025 spotlight paper on failure attribution [1] demonstrated the severity: even state-of-the-art reasoning models achieve only 53.5% accuracy in identifying which agent caused a failure, and only 14.2% accuracy in pinpointing when the critical error occurred. The "attribution gap" means agents cannot learn from their failures because they cannot identify what went wrong.

The HaluMem benchmark [2] revealed that memory systems "generate and accumulate hallucinations during the extraction and updating stages, which subsequently propagate errors to the question answering stage." Memory hallucinations are not just retrieval errors — they compound through the memory pipeline.

More recently, research on LLM-agent memory poisoning [3] demonstrated that attackers can manipulate persistent memory and later influence consequential actions, with attacks succeeding even when systems trust content, lineage, and corroboration. This establishes that memory itself is a security and trust boundary — not merely a storage layer.

B. Authorization vs. Reliability: Two Distinct Problems

A critical distinction emerges from the 2025–2026 research landscape:

Authorization asks: Is this information allowed to enter the agent's context? Reliability asks: Once information is legitimately available, how much should the agent trust it?

These are orthogonal dimensions. A memory can be:

	High reliability	Low reliability
Authorized	Use	Verify / quarantine
Unauthorized	Never retrieve	Never retrieve
Unknown	Verify	Block

ScopeTree [4] addresses the authorization boundary through structural entitlement: out-of-scope content is made structurally unreachable through the runtime load path. ContextOn addresses the reliability dimension: once information passes the authorization boundary, how much should the agent trust it based on observed outcomes?

Neither system alone is sufficient. ScopeTree protects against unauthorized information entering context, but cannot prevent authorized-but-unreliable information from influencing decisions. ContextOn tracks reliability of authorized information, but cannot enforce access boundaries. Together, they form a complete governance framework: Scope $\rightarrow$ Trust $\rightarrow$ Outcome $\rightarrow$ Learning.

C. The Market Gap

The current agent memory landscape has converged on multiple architectural approaches, with rapid evolution in feedback mechanisms:

- 1) Vector memory (Mem0 [5], 59K+ GitHub stars): Extract facts, embed, retrieve by cosine similarity. Mem0 now provides POSITIVE, NEGATIVE, and VERY_NEGATIVE feedback with optional reasons [6]. However, feedback is a binary/trinary signal, not a persistent reliability score that influences retrieval ranking.
- 2) Graph memory (Graphiti/Zep [7], temporal knowledge graphs): Track evolving entity relationships with bi-temporal modeling. Best for temporal reasoning. The graph tracks when facts changed, but does not attach explicit reliability scores that agents can directly reason about for trust-weighted retrieval.
- 3) Feedback-aware memory (Cognee [8], \$7.5M seed): The Memify pipeline refines knowledge graph edge weights through feedback loops. Closer to outcome-aware memory, but operates at the graph structure level rather than maintaining per-fact reliability state designed for retrieval control.
- 4) Experience-learning memory (Acontext [9]): Captures learnings from agent runs, including what worked and what failed, then updates reusable skills. Demonstrates that closed-loop experiential learning is an active architecture — but focuses on skill distillation rather than per-memory reliability scoring.

Recent systems increasingly incorporate feedback, trajectory learning, and memory improvement. However, an open problem remains: how to transform execution outcomes into auditable, per-memory and per-tool reliability signals that control future context admission and action selection.

D. Our Contribution

We present ContextOn.AI OSS as an outcome-aware memory infrastructure. Our contributions:

- 1) Bounded failure attribution with blast-radius control (Section III): When an agent fails, we identify candidate memories that contributed to the failure using answer-content matching, and penalize only those memories with bounded collateral damage. We define this as bounded attribution — it does not claim to identify the causally responsible memory, but rather the set of candidate memories whose content overlaps with the wrong answer.
- 2) Outcome-aware reliability scoring (Section IV): Per-fact confidence that decreases with verified failure, recovers with verified success, and decays over time — giving agents an explicit signal for memory trust. We note that the current model uses heuristic parameters and do not claim Bayesian inference; the model is better described as outcome-conditioned heuristic scoring pending empirical calibration.
- 3) Tool reliability memory (Section V): Tracking success/failure rates per tool, per task type, per agent, enabling agents to predict tool reliability before execution. Research shows tool selection is unreliable — ToolTweak [10] demonstrated that manipulating tool names/descriptions can change selection probability from 20% to 81%, and recent attention studies [11] show tool-selection failures are not merely visibility issues.
- 4) Trust-Bounded Context architecture (Section IX): Combining ContextOn's reliability layer with ScopeTree's authorization layer to form a complete governance framework where Context = Authorized $\cap$ Reliable.
- 5) Open-source implementation with 33 test cases (Section VII), available at <https://github.com/uma-dv/contexton-ai-oss>.

II. BACKGROUND AND RELATED WORK

A. Agent Memory Systems

The 2026 agent memory landscape is characterized by rapid convergence on multiple models [12]:

Semantic memory stores facts as embeddings and retrieves by similarity. Mem0 [5] is the leading implementation, with a multi-stage pipeline: fact extraction $\rightarrow$ consolidation/update $\rightarrow$ embedding/storage $\rightarrow$ retrieval. Mem0 reports sub-second lookup over tens of thousands of facts. However, Zhang et al. [13] showed 26.5% cross-domain failure and 34.5% sycophancy rates when Mem0 memories are injected into queries without trust filtering, establishing that relevant memory $\neq$ trustworthy memory.

Temporal graph memory tracks how facts change over time. Graphiti [7] implements bi-temporal modeling (valid-time and transaction-time), enabling agents to reason about when facts were true. Zep reports 94.7% accuracy on the LoCoMo benchmark [14]. However, temporal graphs track validity (was this fact true at time t?) and provenance, but do not attach explicit reliability scores that agents can directly reason about for trust-weighted retrieval.

Episodic memory stores conversation history. Letta (formerly MemGPT) [15] treats memory as an operating system, with the agent managing its own memory through explicit function calls. Elegant for autonomous agents, but places the reliability judgment on the agent itself — the same agent that made the mistake.

Experience-learning memory captures execution outcomes and converts them into reusable knowledge. Acontext [9] distills session outcomes into agent skills. IBM Research's trajectory-informed memory [16] converts execution traces into recovery strategies, reporting up to 28.5 percentage-point improvement on complex tasks. These demonstrate that agents can improve when execution outcomes become memory — but the question of how outcome evidence should alter per-memory reliability remains open.

B. Failure Attribution

The ICML 2025 paper "Which Agent Causes Task Failures and When?" [1] introduced the Who&When benchmark: 184 annotated failure tasks from 127 multi-agent systems. The best method achieved 53.5% agent-identification accuracy and 14.2% step-identification accuracy. Even OpenAI o1 and DeepSeek R1 failed to achieve practical usability.

The "Where LLM Agents Fail" paper [17] introduced AgentErrorTaxonomy and AgentDebug, reporting 24% higher all-correct accuracy and up to 26% relative improvement in task success from targeted feedback/recovery. This confirms that failure diagnosis + corrective feedback can materially improve agent performance.

MemTrace [18] represents memory pipelines as executable memory-evolution graphs and traces operations to find root causes, reporting that memory failures are systematic and arise from information loss, retrieval misalignment, and memory evolution problems. Attribution-guided optimization improved end-task performance by up to 7.62%.

C. Memory Hallucinations and Trust

HaluMem [2] introduced the first operation-level hallucination benchmark for memory systems, with ~15K memory points and ~3.5K questions. Key findings: memory systems generate hallucinations during extraction; hallucinations accumulate during memory updates; errors propagate to question answering. The paper concludes: "Future research should focus on developing interpretable and constrained memory operation mechanisms that systematically suppress hallucinations and improve memory reliability."

Zhang et al. [13] further established that memory retrieval itself is a trust boundary, demonstrating cross-domain leakage, sycophancy, tool-call drift, and memory-induced jailbreaks when memories are injected without trust filtering. Their MemGate mechanism — a neural gate between memory retrieval and the LLM — reduced cross-domain failure from 26.5% to 3.0%.

D. Tool Reliability

Tool selection is a major failure point. ToolTweak [10] demonstrated that manipulating tool names/descriptions can change tool-selection probability from ~20% to as high as 81%. Recent attention studies [11] show that many tool-selection failures are not because the model failed to see the correct tool, but because it selected incorrectly even after attending to the right one.

E. Memory Poisoning

Research on LLM-agent long-term memory poisoning [3] demonstrated that attackers can manipulate persistent memory and later influence consequential actions. Critically, the paper showed that trusting content, lineage, and corroboration alone is insufficient. This establishes that reliability score alone is not enough — provenance + authority + trust + outcome are all required.

F. The Capability Gap

Table I summarizes the capability landscape.

Capability	Mem0	Graphiti	Cognee	Letta	Acontext	ContextOn
Fact storage	✓	✓	✓	✓	✓	✓
Temporal reasoning	✓	✓	✓	✓	Basic	Basic
Entity resolution	✓	✓	✓	✓	✓	✓
Hybrid retrieval	✓	✓	✓	✓	Varies	Basic
Feedback mechanism	✓ trinary	Partial	✓ Memify	Agent-mgd	✓ skills	✓ per-fact
Failure attribution	✗	✗	✗	✗	✗	✓ bounded
Reliability scoring	✗	Implicit	Partial	✗	✗	✓ explicit
Tool reliability	✗	✗	✗	✗	Partial	✓
Outcome-aware retrieval	✗	✗	✗	✗	Partial	✓

III. BOUNDED FAILURE ATTRIBUTION WITH BLAST-RADIUS CONTROL

A. The Attribution Problem

When an agent gives a wrong answer, the naive approach is to penalize all memories that share keywords with the wrong answer. This creates a blast radius problem: correct facts about the same topic get penalized alongside the incorrect fact.

B. Bounded Attribution via Answer-Content Matching

ContextOn uses answer-content matching rather than query-content matching. The algorithm tokenizes the wrong answer, then for each fact in the graph: skips entity nodes, requires 3+ shared meaningful tokens, and requires Jaccard overlap ≥ 0.4 . Only facts whose content overlaps with the wrong answer are penalized.

We define this as bounded attribution: it identifies the set of candidate facts whose content overlaps with the wrong answer, not the fact that caused the failure. True causal attribution would require execution provenance — tracking which memories were actually retrieved, ranked, and supplied to the model — which is future work (Section XI).

C. Blast-Radius Properties

- 1) Entity safety: Entity nodes (concepts, organizations) are never penalized by failures.
- 2) Direct matching only: Penalties do not cascade through edges.
- 3) Threshold enforcement: Requires 3+ shared meaningful tokens AND Jaccard overlap ≥ 0.4 .

D. Formal Definition

Let F be the set of facts, a be the wrong answer, and $\text{related}(a) \subseteq F$ be the candidate facts:

$$\text{related}(a) = \{f \in F : |\text{tokens}(f) \cap \text{tokens}(a)| \geq 3 \wedge \frac{|\text{tokens}(f) \cap \text{tokens}(a)|}{|\text{tokens}(f) \cup \text{tokens}(a)|} \geq 0.4\}$$

For each $f \in \text{related}(a)$: $\text{failure_count}(f) \leftarrow \text{failure_count}(f) + 1$

Limitation: When correct and incorrect facts about the same topic naturally share vocabulary, correct facts may still be penalized. Semantic understanding or execution provenance would be required to fully solve it.

IV. OUTCOME-AWARE RELIABILITY SCORING

A. Confidence Model

Each fact f in the knowledge graph has:

- $\text{conf_stored}(f)$: base confidence from ingestion
- $\text{failure_count}(f)$: number of times this fact was a candidate in a failed answer
- $\text{success_count}(f)$: number of times this fact was verified as correct
- $\text{mentions}(f)$: number of times this fact was retrieved
- $\text{created_at}(f)$: timestamp of ingestion
- $\text{last_updated}(f)$: timestamp of last modification

B. Reliability Score

The reliability score $R(f)$ is computed as:

$$R(f) = \text{conf_stored}(f) \times \text{decay}(t) \times \text{failure_penalty}(f)$$

where: Time decay: $\text{decay}(t) = 0.95^{\text{days_since_verified}}$

This implements a 5% daily decay, reflecting the intuition that unverified knowledge becomes less reliable over time. The decay rate is a design parameter — 0.95 was chosen to balance freshness against over-penalization for recent knowledge.

Failure penalty: $\text{failure_penalty}(f) = 0.5^{\text{failure_count}(f)}$

Each failure halves the confidence. After 1 failure: 50%. After 2 failures: 25%. After 5 failures: 3%. This implements the intuition that repeated failures indicate systematic unreliability.

C. Confidence Restoration

Restoration is not part of the scoring formula — it modifies the formula's inputs via $\text{record_success}()$. Two mechanisms operate:

Mechanism 1: Failure count decrement. Each success decrements $\text{failure_count}(f)$ by 1 (minimum 0), directly reducing the failure penalty in the next scoring call.

Mechanism 2: Bounded additive restoration. The stored confidence is increased by 0.3, capped at the node's original confidence:

$$\text{conf_stored}(f) \leftarrow \min(\text{conf_original}(f), \text{conf_stored}(f) + 0.3)$$

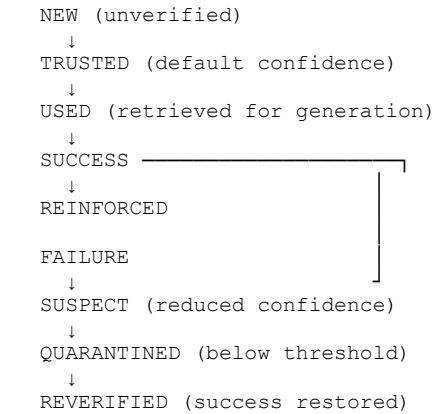
The ceiling $\text{conf_original}(f)$ is recorded at ingestion time and never changes, preventing success from permanently masking past failures. A fact that has been contradicted cannot fully recover to its original confidence through success alone — it requires re-ingestion with a fresh source.

D. Properties

- Property 4 (Monotonic decay): Without explicit success, R(f) monotonically decreases over time.
- Property 5 (Failure dominance): For facts with equal stored confidence, higher failure count always produces lower reliability.
- Property 6 (Bounded recovery): Verified success increases reliability, but never above the original stored confidence.

E. Trust Lifecycle

The reliability of a memory follows a lifecycle:



Memories below a configurable confidence threshold are excluded from retrieval, implementing automatic quarantine of unreliable knowledge.

V. TOOL RELIABILITY MEMORY

Tools are registered with metadata and each execution is recorded. The tool reliability score is:

$$R(t) = \text{success_count}(t) / (\text{success_count}(t) + \text{failure_count}(t) + 1) \times \text{decay}(t)$$

When a tool fails, its reliability score decreases, propagating to all agents using that tool.

VI. ENTERPRISE GRAPH INGEST AND PROVENANCE

ContextOn provides ingest_from_enterprise_graph() to connect external knowledge sources. Ingested knowledge retains its reliability provenance. Future work: implement execution provenance tracking which memories were actually retrieved, ranked, and supplied to the model.

VII. IMPLEMENTATION AND EVALUATION

A. System Architecture

ContextOn.AI OSS is implemented in Python with zero external dependencies:

```
contexton_ai_oss/
├── __init__.py           # Package initialization
├── graph.py              # Core knowledge graph with batch saving
├── entities.py           # Entity extraction and alias resolution
├── confidence.py         # Reliability scoring engine
├── failure_learning.py   # Bounded failure attribution
├── advanced.py           # Confidence-weighted BFS, multi-hop reasoning
├── mcp_server.py         # MCP integration for agent frameworks
└── text_utils.py        # Tokenization and text processing
```

B. Test Suite

The test suite contains 33 test cases covering:

Category	Tests	What It Proves
Graph operations	6	Core CRUD, persistence
Ingestion	4	Entity extraction, deduplication
Querying	5	Token overlap, confidence ranking
Confidence	4	Decay, failure penalty, success restoration
Failure learning	5	Blast radius, entity safety, double-counting fix
Entity resolution	3	Alias detection, false-positive prevention
Procedures	3	Storage, querying, failure attribution
Tools	3	Registration, outcome tracking, failure propagation
Context assembly	3	Session filtering, min-confidence, scoping
Agent capabilities	4	Scoping, hygiene sweep, pruning

C. Key Test: Bounded Blast Radius

```
def test_failure_penalizes_only_wrong_facts():
    """Only facts matching the wrong answer get penalized."""
    graph = ContextGraph()
    graph.ingest("What is PM-JAY?",
                 "PM-JAY is health insurance for poor families")
    graph.ingest("What is PM-JAY?",
                 "PM-JAY is administered by NHA")

    before = graph.query("PM-JAY")[0]["confidence"]

    # Wrong answer shares tokens with the health insurance fact
    graph.record_failure(
        "What is PM-JAY?",
        "PM-JAY is housing scheme for poor families",
        "Wrong domain"
    )

    after = graph.query("PM-JAY")[0]["confidence"]
    assert after < before # Confidence decreased

    # But NHA fact was NOT penalized (doesn't match wrong answer)
    nha_facts = [n for n in graph.nodes.values()
                  if "NHA" in n.get("content", "")]
    for f in nha_facts:
        assert f.get("failure_count", 0) == 0
```

D. Key Test: Success Restores Confidence

```
def test_success_restores_confidence():
    """Verified success undoes the damage of a prior failure."""
    graph = ContextGraph()
    graph.ingest("What is PM-JAY?",
                 "PM-JAY is health insurance for poor families")

    before = graph.query("PM-JAY")[0]["confidence"]

    graph.record_failure(
        "What is PM-JAY?",
        "PM-JAY is housing scheme for poor families",
        "Wrong domain"
    )
    after_failure = graph.query("PM-JAY")[0]["confidence"]
    assert after_failure < before

    graph.record_success(
        "What is PM-JAY?",
        "PM-JAY is health insurance for poor families"
    )
    after_success = graph.query("PM-JAY")[0]["confidence"]
    assert after_success > after_failure
```

E. Key Test: Single-Counting of Failures

```
def test_failure_count_is_single():
    """One failure = 50% drop, not 75%."""
    graph = ContextGraph()
    graph.ingest("What is X?", "X is a tool for testing")

    graph.record_failure("What is X?", "X is a database", "Wrong type")

    for nid, node in graph.nodes.items():
        if "testing" in node.get("content", ""):
            assert node.get("failure_count") == 1 # Not 2
```

F. Formula-to-Code Mapping

Every formula in Section IV maps directly to a specific function in the codebase:

Formula	Code Location	Function
$R(f) = conf_stored \times decay \times failure_penalty$	confidence.py:node_confidence()	Single source of truth for scoring
$decay(t) = 0.95^{days}$	confidence.py:node_confidence()	Time decay computation
$failure_penalty(f) = 0.5^{fc}$	confidence.py:node_confidence()	Exponential failure penalty
$conf_stored \leftarrow \min(ceiling, conf + 0.3)$	failure_learning.py:record_success()	Bounded additive restoration
Failure count decrement	failure_learning.py:record_success()	Success counteracts failure
Blast radius: AND matching	failure_learning.py:_find_related_nodes()	shared ≥ 3 AND overlap ≥ 0.4
original_confidence ceiling	graph.py:add_node()	Recorded at ingestion, never modified

All 33 tests pass and verify these formulas. The test suite (tests/test_core.py) is included in the repository.

VIII. COMPARISON WITH EXISTING METHODS

A. Comparison with Mem0

Aspect	Mem0	ContextOn
Primary function	Memory storage & retrieval	Reliability scoring
Feedback	Trinary (POS/NEG/VERY_NEG)	Per-fact reliability score
Confidence model	Implicit	Explicit (outcome-conditioned)
Tool reliability	✗	✓
Retrieval quality	✓ hybrid	Basic (token overlap)
Positioning	Memory database	Reliability layer

B. Comparison with Graphiti/Zep

Aspect	Graphiti/Zep	ContextOn
Temporal reasoning	✓ bi-temporal	Basic
Reliability scoring	Implicit (provenance)	Explicit (per-fact)
Failure attribution	✗	✓ bounded
Retrieval quality	✓ semantic+keyword+graph	Basic
Infrastructure	Neo4j required	SQLite only

C. Comparison with Cognee

Aspect	Cognee	ContextOn
Feedback mechanism	Memify (edge weights)	Per-fact reliability
Failure attribution	✗	✓ bounded
Enterprise readiness	✓ (\$7.5M, 70+ cos)	Alpha
Tool reliability	✗	✓

IX. TRUST-BOUNDED CONTEXT: COMBINING GOVERNANCE AND RELIABILITY

A. The Combined Architecture

Layer	Question	System
Governance	What may this agent access?	ScopeTree
Context	What information is available?	ScopeTree / memory provider
Trust	What information should it trust?	ContextOn
Execution	What did the agent do?	ContextOn
Outcome	Did it work?	ContextOn
Learning	What should change next time?	ContextOn

B. Trust-Bounded Context Formula

Traditional RAG asks: "Is this relevant?" Trust-Bounded Context asks two questions: "Is this allowed?" (ScopeTree) and "Is this trustworthy?" (ContextOn). Formally: $\text{Context} = \text{Authorized} \cap \text{Reliable}$.

C. Governance × Reliability Matrix

	High reliability	Low reliability
Authorized	✓ Use	⚠ Verify / quarantine
Unauthorized	✗ Never retrieve	✗ Never retrieve
Unknown	⚠ Verify	✗ Block

D. Proposed Experiment

Four-condition experiment: (1) RAG baseline, (2) ScopeTree only, (3) ContextOn only, (4) ScopeTree + ContextOn. Measures: accuracy, hallucination, leakage, repeated failures, recovery, tool failures, token cost, false refusal, stale-memory use.

X. WHY THIS IS ESSENTIAL FOR AGENTIC AI DEPLOYMENT

A. The Reliability Crisis

Metric	Value	Source
AI projects failing	80%+	RAND 2025
GenAI pilots never scale	95%	MIT 2025
Companies abandoning AI	42%	S&P 2025
No tangible AI value	74%	BCG/Stanford 2025
No AI-ready data	60%	Gartner 2025

B. The Three Reliability Gaps

- Gap 1: Bounded "failure signal". When an agent fails, which candidate memories contributed?
- Gap 2: Reliability scoring. How much should an agent trust a given memory?
- Gap 3: Tool reliability memory. Which tools work for which tasks?

C. The Complementarity Argument

ContextOn does not replace Mem0, Graphiti, Cognee, or Letta. It adds the reliability layer that existing memory providers lack, positioning it as infrastructure that increases the value of existing memory investments.

XI. LIMITATIONS AND FUTURE WORK

- A) Retrieval quality: Token-overlap based, not semantic. Future work: optional vector search.
- B) Bounded vs. causal attribution: Uses answer-content matching, not execution provenance.
- C) Blast radius under realistic conditions: Correct facts sharing vocabulary may still be penalized.
- D) Benchmark evidence: No comparative results yet.
- E) Confidence model calibration: Heuristic parameters, not empirically calibrated.
- F) Scalability: JSON persistence, not suitable for millions of facts.
- G) Intra-scope threats: Does not protect against malicious content in authorized scope.

XII. CONCLUSION

We presented ContextOn.AI OSS, an outcome-aware memory infrastructure that addresses three gaps in the agent memory landscape: bounded failure attribution, reliability scoring, and tool reliability memory. We compared against Mem0, Graphiti/Zep, Cognee, Letta, and Acontext, showing that while existing systems increasingly incorporate feedback and learning, ContextOn fills the specific gap of explicit, auditable, per-memory reliability scoring designed for retrieval control.

We proposed Trust-Bounded Context as a combined architecture with ScopeTree, where authorization and reliability form complementary layers. Together, they address both unauthorized-context failures and authorized-but-unreliable-context failures. ContextOn is an early-stage open-source project with significant limitations: no benchmark evidence, basic retrieval quality, heuristic confidence parameters, and bounded (not causal) failure attribution. But it addresses a real and underserved problem. The path forward: prove with benchmarks that feeding agent outcomes back into retrieval materially reduces repeated errors.

XIII. REFERENCES

[1] C. Wang, H. Wang, Y. Chen, and Q. Wu, "Which Agent Causes Task Failures and When? On Automated Failure Attribution of LLM Multi-Agent Systems," in Proc. ICML, 2025. Spotlight paper.

[2] D. Chen et al., "HaluMem: Evaluating Hallucinations in Memory Systems of Agents," arXiv:2511.03506, Nov. 2025.

[3] "Securing LLM-Agent Long-Term Memory Against Poisoning: Non-Malleable, Origin-Bound Authority with Machine-Checked Guarantees," arXiv:2606.24322, Jun. 2026.

[4] K. Vadagam, "Stop Putting Organizations into AI Models: Put AI into Organizational Needs — ScopeTree," ODEFTO Tech, Aug. 2026.

[5] P. Chhikara et al., "Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory," arXiv:2504.19413, 2025.

[6] Mem0, "Feedback API Documentation," 2026. <https://docs.mem0.ai/api-reference/memory/feedback>

[7] P. Rasmussen et al., "Graphiti: Build Real-Time Knowledge Graphs for AI Agents," 2025. <https://github.com/getzep/graphiti>

[8] Topoteretes, "Cognee: Memory Control Plane for AI Agents," 2025. \$7.5M seed, 12K+ stars.

[9] MemDB, "Acontext: Agent Skills as a Memory Layer," 2026. 3.6K GitHub stars.

[10] "ToolTweak: An Attack on Tool Selection in LLM-based Agents," arXiv:2510.02554, 2025.

[11] "Looking Is Not Picking: An Attention-Segment Account of Tool-Selection Failures," arXiv:2606.16364, Jun. 2026.

[12] LLM4Agents, "Agent memory in 2026: Graphiti, Mem0, and the layer we haven't built," May 2026.

- [13] Zhang et al., "Beyond Similarity: Trustworthy Memory Search for Personal AI Agents," arXiv:2606.06054, Jun. 2026.
- [14] Zep, "Research: LoCoMo and LongMemEval Benchmarks," 2026.
- [15] Letta, "Letta: The Platform for Building Stateful Agents," 2025. \$10M seed.
- [16] G. Fang et al., "Trajectory-Informed Memory Generation for Self-Improving Agent Systems," arXiv:2603.10600, Mar. 2026. IBM Research.
- [17] "Where LLM Agents Fail and How They Can Learn From Failures," arXiv:2509.25370, 2025.
- [18] "MemTrace: Tracing and Attributing Errors in LLM Memory Systems," arXiv:2605.28732, 2026.
- [19] RAND Corporation, "The State of AI in 2025," 2025.
- [20] MIT NANDA Initiative, "The GenAI Divide," 2025.
- [21] S&P Global, "AI Project Failure Rates in 2025," 2025.
- [22] BCG / Stanford HAI, "State of AI in the Enterprise 2025," Oct. 2024.
- [23] Gartner, "60% of AI Projects Without AI-Ready Data Will Be Abandoned," 2025.
- [24] Databricks, "2026 State of AI Agents," 2026.
- [25] Deloitte, "State of AI in the Enterprise 2026," 2026.
- [26] McKinsey, "The State of AI 2025," 2025.
- [27] G. Zhou et al., "Memento-Skills: Let Agents Design Agents," arXiv:2603.18743, Mar. 2026.
- [28] S. Lewis, "Structured Distillation for Personalized Agent Memory," arXiv:2603.13017, Mar. 2026.
- [29] "Agentic Memory: Learning Unified Long-Term and Short-Term Memory Management," arXiv:2601.01885, Jan. 2026.
- [30] "Storage Is Not Memory: A Retrieval-Centered Architecture for Agent Recall," arXiv:2605.04897, 2026.
- [31] "LLM-Based Agents for Tool Learning: A Survey," Data Science and Engineering, Springer, 2025.
- [32] Vectorize, "Mem0 vs Zep: AI Agent Memory Compared," 2026.
- [33] 1337skills, "AI Agent Memory in 2026," Jun. 2026.
- [34] Labo LLM, "Agent Memory in 2026," Jul. 2026.
- [35] Blake Crosley, "What I Told NIST About AI Agent Security," Feb. 2026.
- [36] OECD, "AI Adoption by SMEs," Dec. 2025.
- [37] Connected Paths, "AI Project Failure Statistics 2026," Jul. 2026.