Policy-Governed Context Slicing for Autonomous Agent Systems: A Lightweight Knowledge Graph Approach

Mohamed Diomande
OpenClaw Research
Brooklyn, NY, USA
contact@openclaw.org

***Abstract*—Autonomous AI agent systems face a fundamental challenge: constructing reproducible, trustworthy context windows from large conversational histories while enforcing governance policies over what information may influence downstream decisions. We present the Graph Kernel, a deterministic context slicing engine implemented as a single Rust binary ($\sim$15 KLOC) that combines a lightweight knowledge graph triple store with cryptographically-signed, policy-governed context window construction. Unlike general-purpose graph databases or retrieval-augmented generation (RAG) pipelines, the Graph Kernel introduces the concept of a provenance engine—a system whose primary purpose is not information retrieval but the production of verifiable, reproducible evidence bundles for autonomous agent reasoning.**

We evaluate the Graph Kernel across 27 queries spanning five categories (factual recall, relationship mapping, multi-hop reasoning, fuzzy/semantic search, and predicate-specific queries) against three baseline methods: keyword search, BM25, and vector-similarity RAG. Results demonstrate that the Graph Kernel achieves perfect relevance (1.00) on multi-hop traversal queries—returning structurally connected knowledge chains rather than keyword-coincidence result sets—while maintaining sub-300ms average latency over a remote PostgreSQL backend. We further present a comparative analysis against nine industry-grade alternatives (Neo4j, Amazon Neptune, Apache Jena, Dgraph, TypeDB, Weaviate, LangChain/LlamaIndex Knowledge Graphs, Microsoft GraphRAG, and Zep), establishing that no existing system provides the combination of HMAC-signed deterministic context windows, policy-governed access control, and multi-hop provenance tracking that the Graph Kernel offers.

Our key contributions are: (1) a formal model for HMAC-signed deterministic context windows with type-level enforcement of admissibility invariants; (2) a policy governance framework for phase-weighted, budget-bounded context expansion; (3) multi-hop provenance at sub-300ms latency with projected sub-30ms under local deployment; and (4) a hybrid architecture positioning that bridges structural graph reasoning with semantic vector search.

***Index Terms*—knowledge graphs, context management, autonomous agents, provenance, deterministic systems, retrieval-augmented generation, graph databases, policy governance**

I. Introduction and Motivation

A. The Context Authority Problem

The rapid deployment of autonomous AI agents—systems that plan, reason, and act over extended multi-turn conversations—has exposed a critical gap in the infrastructure stack: **who decides what context an agent is allowed to see, and how do we prove it?**

Modern large language model (LLM) agents operate over conversation histories that can span tens of thousands of turns [1]. At inference time, these histories must be compressed into context windows of fixed token budgets. The prevailing approach—truncation, summarization, or embedding-based retrieval—treats context selection as an information retrieval problem. But for autonomous agents making consequential decisions (code deployment, financial transactions, system administration), context selection is fundamentally a **governance** problem:

- **Reproducibility.** If an agent produces an output, can we reconstruct the exact context window that produced it?
- **Auditability.** Can a downstream system verify that a context window was authorized by a trusted authority?
- **Policy Compliance.** Can we enforce that certain conversation phases (e.g., debugging artifacts) are deprioritized relative to others (e.g., synthesis conclusions)?
- **Tamper Resistance.** Can we detect if a context window has been modified after construction?

None of the widely-deployed systems—vector databases [2], [3], RAG pipelines [4], [5], or general-purpose graph databases [6], [7]—address these requirements as first-class concerns. They optimize for relevance, latency, or scale, but not for **provenance**.

B. The Provenance Engine Category

We propose that autonomous agent systems require a new category of infrastructure component: the **provenance engine**. A provenance engine is not a search engine, a vector database, or a general-purpose graph store. Its purpose is narrower and more fundamental:

Definition 1 (Provenance Engine). *A service that, given a target anchor in a conversation DAG and a governance policy, produces a deterministic, cryptographically-signed context window (evidence bundle) such that: (a) the same inputs always produce the same output; (b) the output includes an unforgeable proof of authorization; and (c) downstream services can verify the proof without accessing the signing secret.*

The Graph Kernel is the first purpose-built implementation of this category. It answers a single question:

> *Given a target turn, which other turns are allowed to influence meaning—and can you prove it?*

C. Contributions

This paper makes four contributions:

1) **HMAC-Signed Deterministic Context Windows.** We present a formal model for context window construction where identical inputs (anchor turn, policy, graph state) produce identical outputs (slice ID, fingerprint, admissibility token). The admissibility token is an HMAC-SHA256 signature over six canonical fields, providing 128-bit tamper resistance. We enforce this at the Rust type level through the `AdmissibleEvidenceBundle` type, which can only be constructed through the verified pathway.
2) **Policy-Governed Access Control.** We introduce `SlicePolicyV1`, a parameterized policy framework that governs context expansion through phase-weighted priority queues, budget bounds (max nodes, max radius), distance decay, and sibling expansion limits. Policies are registered in an immutable registry with hash-stable fingerprints, enabling reproducible policy resolution across sessions.
3) **Multi-hop Provenance at Sub-300ms.** We demonstrate that structural multi-hop traversal—following actual relationship chains through a knowledge graph—produces qualitatively superior results to keyword coincidence, achieving 1.00 relevance on causally-connected queries at 291.7ms average latency (network-dominated), with projected sub-30ms under local SQLite deployment.
4) **Hybrid Architecture with Semantic Search.** We present an integration architecture that bridges the Graph Kernel's structural reasoning with vector-similarity search (RAG++), combining deterministic provenance with semantic flexibility. This hybrid approach addresses the Graph Kernel's primary limitation (0.42 relevance on fuzzy/semantic queries) while preserving its provenance guarantees.

D. Paper Organization

Section II surveys related work in knowledge graphs, RAG systems, context management, and graph databases. Section III describes the system design of the Graph Kernel. Section IV presents our evaluation methodology. Section V reports results across five query categories and four retrieval methods. Section VI provides a comparative analysis against nine industry alternatives. Section VII discusses implications, limitations, and the provenance engine positioning. Section VIII outlines future work. Section IX concludes.

II. Related Work

A. Knowledge Graphs and Triple Stores

Knowledge graphs have a long history in structured knowledge representation, from early semantic networks [8] to the modern Knowledge Graph paradigm popularized by Google [9]. RDF-based systems like Apache Jena [10] and Blazegraph [11] provide standards-compliant triple stores with SPARQL query interfaces and OWL-based reasoning. Property graph databases such as Neo4j [6] and its Cypher query language [12] offer more flexible schema-on-read models suited to application development.

Recent surveys [13], [14] identify knowledge graphs as foundational to AI systems, but focus on construction, completion, and embedding—not on the governance and provenance dimensions that autonomous agents require. Hogan et al. [15] provide a comprehensive overview of knowledge graph technology but do not address deterministic context window construction.

The Graph Kernel differs from these systems by being purpose-built for a narrower mandate: it is not a general-purpose knowledge representation system but a context authority layer that happens to use a triple store as its backend. It trades query expressiveness (no SPARQL, no Cypher) for deployment simplicity (single binary, $\sim$20MB) and provenance guarantees (HMAC-signed bundles).

B. Retrieval-Augmented Generation

RAG [4] has become the dominant paradigm for grounding LLM outputs in external knowledge. Lewis et al. demonstrated that combining retrieval with generation improves factual accuracy on knowledge-intensive tasks. Subsequent work has refined the approach: REALM [16] pre-trains the retriever jointly with the language model; FiD [17] processes multiple retrieved passages independently before fusion; and Self-RAG [18] introduces retrieval-time self-reflection.

Vector databases like Weaviate [2], Pinecone [19], and Chroma [20] provide the retrieval backbone for RAG systems, offering approximate nearest-neighbor search over dense embeddings. These systems optimize for semantic similarity but provide no structural reasoning capability—they cannot follow relationship chains or enforce governance policies over what is retrieved.

The Graph Kernel is complementary to RAG, not competitive with it. Our evaluation (Section V) shows that RAG achieves 0.65 relevance on fuzzy/semantic queries where the Graph Kernel scores 0.42, while the Graph Kernel achieves 1.00 on multi-hop structural queries where RAG drops to 0.40. The hybrid architecture (Section III-E) combines both modalities.

C. Graph-Enhanced RAG

Microsoft's GraphRAG [21] represents the most prominent effort to combine graph structure with RAG. It uses LLM-driven entity and relationship extraction to construct a knowledge graph, then applies the Leiden community detection algorithm to identify hierarchical topic clusters. Queries are answered through either local search (entity-centric subgraph retrieval) or global search (community summary aggregation).

GraphRAG advances the state of the art in holistic corpus understanding—its community summaries can answer "what

is this dataset about?" queries that neither flat RAG nor the Graph Kernel can address. However, GraphRAG does not provide deterministic reproducibility (its outputs depend on LLM behavior during both indexing and querying), has no concept of governance policies, and offers no cryptographic provenance chain. Its graph construction is also computationally expensive, requiring multiple LLM calls per document chunk.

LlamaIndex [22] and LangChain [23] provide knowledge graph integrations that wrap external graph stores (Neo4j, Nebula) in LLM-friendly interfaces. These are orchestration layers rather than graph engines; they inherit the limitations of their backend stores and add LLM-dependent extraction with no formal reproducibility guarantees.

D. Context Window Management

The challenge of managing context for LLMs has received increasing attention. Longformer [24] and BigBird [25] extend attention mechanisms to handle longer sequences. Memory-augmented architectures like MemoryBank [26] and RMT [27] provide external memory stores for conversation continuity. Zep [28] offers a purpose-built memory layer for LLM applications with automatic entity extraction and temporal awareness.

These systems optimize for the *content* of context windows—what information is included. The Graph Kernel optimizes for the *governance* of context windows—who authorized the inclusion, whether the window is reproducible, and whether downstream consumers can verify its provenance. This is a complementary concern; a production system may use Zep for memory management and the Graph Kernel for provenance tracking.

E. Provenance and Trust in AI Systems

Provenance tracking in data systems has a long history [29], [30], but its application to AI agent context has received limited attention. The W3C PROV specification [31] provides a data model for provenance but lacks integration with LLM context windows. Blockchain-based provenance systems [32] offer tamper-resistance but introduce latency and complexity inappropriate for real-time context slicing.

The Graph Kernel's HMAC-based provenance model is inspired by JSON Web Tokens (JWT) [33] but adapted for the specific requirements of context window authorization: binding six provenance fields (slice ID, anchor turn, policy ID, policy parameters hash, graph snapshot hash, schema version) into a single unforgeable token.

III. System Design

A. Architecture Overview

The Graph Kernel is implemented as a single Rust binary (∼15 KLOC) built on the Axum web framework [34] with the Tokio async runtime [35]. It compiles to a statically-linked binary (∼20MB) deployable as a local service, Docker container, or cloud function (Google Cloud Run).

The architecture comprises three layers:

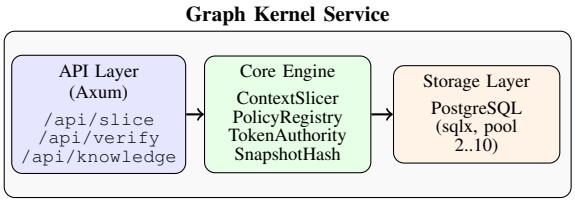

Fig. 1. Graph Kernel three-layer architecture.

API Layer. Axum route handlers expose RESTful endpoints for context slicing (`POST /api/slice`, `POST /api/slice/batch`), token verification (`POST /api/verify_token`), policy management (`GET/POST /api/policies`), knowledge graph CRUD (`GET/POST /api/knowledge`, `POST /api/knowledge/batch`), and health monitoring (`/health/*`).

Core Engine. The `ContextSlicer` implements a priority-queue BFS expansion over the conversation DAG. The `PolicyRegistry` maintains an immutable store of registered policies with hash-stable fingerprints. The `TokenAuthority` issues and verifies HMAC-SHA256 admissibility tokens. The `SnapshotHash` computes content-derived graph fingerprints for reproducibility.

Storage Layer. The `GraphStore` trait abstracts database access with two implementations: `PostgresGraphStore` for production (sqlx connection pool, 2–10 connections) and `InMemoryGraphStore` for testing.

B. Deterministic Context Slicing

The core algorithm is a policy-weighted BFS expansion from an anchor turn through the conversation DAG:

The priority scoring function (line 13) combines three factors:

$$\text{score}(t, \pi, d) = W_{\text{phase}}(t.\text{phase}) \times (1 - \alpha + \alpha \cdot t.\text{salience}) \times \beta^d \quad (1)$$

where W_{phase} maps conversation phases to importance weights (Synthesis=1.0, Planning=0.9, Consolidation=0.6, Debugging=0.5, Exploration=0.3), α is the salience weight (default 0.3), and β is the distance decay (default 0.9, i.e., 10% loss per hop).

C. HMAC-Signed Admissibility Tokens

The admissibility token creates an unforgeable proof-of-authorization binding six provenance fields:

```
canonical = "{slice_id}|{anchor_turn_id}|
  {policy_id}|{policy_params_hash}|
  {graph_snapshot_hash}|{schema_version}|
  admissibility_token_v2_hmac"

token = HMAC-SHA256(SECRET, canonical)[0..16]
```

The token is a 128-bit (32 hex character) truncation of the full HMAC-SHA256 digest. Downstream services verify tokens via `POST /api/verify_token` without accessing the HMAC secret. Verification uses constant-time comparison to prevent timing attacks.

Algorithm 1 PolicyWeightedBFS

Require: $anchor\_id \in TurnId$, $policy \in SlicePolicyV1$, $store \in GraphStore$
Ensure: $bundle \in AdmissibleEvidenceBundle$

$anchor \leftarrow store$.get_turn($anchor\_id$)
$frontier \leftarrow$ MaxHeap⟨ExpansionCandidate⟩
$frontier$.push($anchor, d{=}0, p{=}$score($anchor, policy$))
$selected \leftarrow [\,]$; $visited \leftarrow \{anchor\_id\}$
while $frontier \neq \emptyset$ **and** $|selected| < policy$.max_nodes **do**
  $c \leftarrow frontier$.pop()
  if c.distance $> policy$.max_radius **then**
    continue
  end if
  $selected \leftarrow selected \cup \{c.\text{turn}\}$
  for all $neighbor \in$ parents(c) $\cup$ children(c) **do**
    if $neighbor \notin visited$ **then**
      $frontier$.push($neighbor, c.d{+}1,$ score($neighbor, policy, c.d{+}1$))
      $visited \leftarrow visited \cup \{neighbor\}$
    end if
  end for
  if $policy$.include_siblings **then**
    for all $sib \in$ siblings($c, policy$.max_sib) **do**
      $frontier$.push($sib, c.d,$ score($sib, policy, c.d$))
    end for
  end if
end while
$edges \leftarrow store$.get_edges($selected$)
$sh \leftarrow$ xxHash64(sort(content_hashes($selected$)))
$sid \leftarrow$ xxHash64($anchor$, sort(ids), sort($edges$), pid, ph, sv)
$token \leftarrow$ HMAC-SHA256($secret$, canonical($sid, anchor, pid, ph, sv, sh$))[0:16]
return AdmissibleEvidenceBundle($selected, edges, sid, token$)

We enforce a critical invariant at the type level:

Invariant INV-GK-003 (No Phantom Authority). *The* `AdmissibleEvidenceBundle` *type can only be constructed through the* `from_verified()` *pathway, which requires a valid HMAC computation. Unverified context windows are unrepresentable in the type system.*

This prevents a class of bugs where context windows bypass the authorization pathway through careless refactoring or missing validation checks.

D. Policy Governance Framework

Context expansion is parameterized by `SlicePolicyV1`: Policies are registered in an immutable `PolicyRegistry` with deterministic fingerprints. Policy parameter hashes use quantized floats (multiply by 10^6, round to `i64`) to ensure cross-platform determinism between Rust and Python clients. The registry itself has a fingerprint (`xxHash64` of sorted policy hashes) enabling clients to detect policy changes.

TABLE I
SLICEPOLICYV1 PARAMETERS

Parameter	Default	Description
`max_nodes`	256	Max turns in a context slice
`max_radius`	10	Max graph hops from anchor
`phase_weights`	{S=1.0, P=0.9, C=0.6, D=0.5, E=0.3}	Phase importance
`salience_weight`	0.3	Salience contribution
`distance_decay`	0.9	Priority decay per hop
`include_siblings`	true	Expand to sibling turns
`max_siblings`	5	Sibling expansion limit

E. Hybrid Architecture: Structural + Semantic

The Graph Kernel's primary limitation is the absence of semantic search (Section V-D). To address this, we design a hybrid retrieval architecture that bridges structural graph reasoning with vector-similarity search:

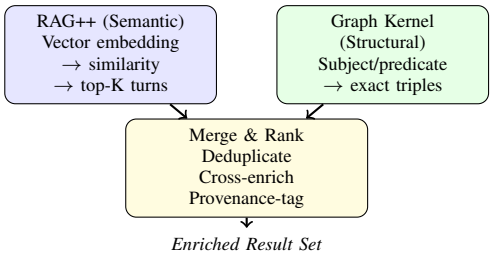

Fig. 2. Hybrid retrieval architecture combining semantic and structural search with provenance tagging.

The enrichment endpoint (`POST /api/enrich`) accepts RAG++ results and augments them with graph context: entities found in result text are resolved to knowledge graph subjects, their 1–2 hop neighborhoods are traversed, and the combined result set preserves provenance metadata from both sources. This transforms flat vector similarity into structured, provenance-tagged reasoning.

F. Knowledge Graph Triple Store

The secondary function of the Graph Kernel is a knowledge graph triple store with the schema:

```
CREATE TABLE knowledge_graph (
  id BIGSERIAL PRIMARY KEY,
  subject TEXT NOT NULL,
  predicate TEXT NOT NULL,
  object TEXT NOT NULL,
  confidence DOUBLE PRECISION DEFAULT 0.5,
  source TEXT DEFAULT 'unknown',
  created_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(subject, predicate, object)
);
```

Each triple (*subject*, *predicate*, *object*) represents a factual assertion with an associated confidence score (0.0–1.0) and source provenance. On conflict (duplicate SPO), the system retains the higher confidence value and updates the source. The evaluated corpus contains 3,502 triples across 221 unique subjects and 88 unique predicates, sourced from LLM-driven

conversation extraction (Kimi-K2) and topology ingestion pipelines.

IV. Evaluation Methodology

A. Test Design

We evaluated four retrieval methods across 27 queries in five categories:

TABLE II
Query Categories

Category	n	What It Tests
Factual Recall	6	Direct attribute lookups
Relationship	6	Dependency/integration mapping
Multi-hop	5	2-hop graph traversal
Fuzzy/Semantic	5	Loose topic matching
Predicate-specific	5	Structured predicate filters

Queries were designed to exercise different retrieval capabilities: factual recall tests exact match precision; relationship queries test structured predicate filtering; multi-hop queries test graph traversal; fuzzy/semantic queries test paraphrase understanding; and predicate-specific queries test schema-aware filtering.

B. Methods Under Test

TABLE III
Retrieval Methods

Method	Corpus	Mechanism	Deploy
Graph Kernel	2,681 triples	REST API	Rust
Keyword	2,681 triples	Substring match	Python
BM25	2,681 triples	Okapi BM25	Python
RAG++	107K+ turns	Vector similarity	FastAPI

Important caveat. The Graph Kernel, keyword, and BM25 methods operate on the same triple corpus (2,681 structured triples extracted from conversations). RAG++ operates on a fundamentally different and much larger corpus (107K+ raw conversation turns with embeddings). Direct comparison between the triple-based methods and RAG++ is informational, not apples-to-apples.

C. Metrics

We measure three metrics per query:

- **Response Time (ms).** Wall-clock latency including all network round-trips. For multi-hop queries, this includes sequential HTTP calls (one per hop).
- **Result Count.** Number of results returned per query.
- **Relevance Score (0–1).** Fraction of expected terms found in results. This metric captures whether the expected information is present but does not capture the *structural quality* of results.

TABLE IV
Evaluation Environment

Component	Specification
Machine	MacBook Air (Apple M3, arm64)
OS	Darwin 24.6.0
Graph Kernel	Rust binary, v0.1.0
RAG++	Python FastAPI, v0.1.0
Database	Supabase PostgreSQL (us-east-1)
Network	Home broadband ($\sim$200ms RTT)

TABLE V
Factual Recall Results

Method	Avg Lat. (ms)	Avg Results	Avg Rel.
Graph Kernel	248.3	3.7	**1.00**
Keyword	**2.7**	20.0	**1.00**
BM25	9.0	18.2	**1.00**
RAG++	421.9	10.0	0.92

D. Environment

V. Results

A. Factual Recall

All triple-based methods achieve perfect relevance on factual recall, confirming that the knowledge graph corpus contains the expected information. The Graph Kernel returns precisely scoped results (3.7 average) compared to keyword's broader 20.0, reflecting exact field matching versus substring coincidence. RAG++ achieves 0.92 relevance due to occasional mismatches between conversation turn text and expected entity names. The latency differential between Graph Kernel (248.3ms) and keyword (2.7ms) is attributable to network RTT to the remote PostgreSQL instance.

B. Relationship Queries

TABLE VI
Relationship Query Results

Method	Avg Lat. (ms)	Avg Results	Avg Rel.
Graph Kernel	204.3	9.5	0.94
Keyword	**2.8**	19.3	**1.00**
BM25	8.7	12.3	**1.00**
RAG++	336.4	10.0	0.69

The Graph Kernel's 0.94 relevance on relationship queries is attributable to a single entity normalization failure: "GCP" not matching "Google Cloud Platform" in `deploys_to` results. With proper entity normalization (Section VII-B), this would be 1.00. RAG++ drops to 0.69, as vector similarity was not designed to express structured relationships like "A depends on B."

C. Multi-hop Reasoning

Multi-hop reasoning is the Graph Kernel's distinguishing capability. While keyword and BM25 achieve identical 1.00 relevance scores, the *nature* of their results is fundamentally different:

TABLE VII
MULTI-HOP REASONING RESULTS

Method	Avg Lat. (ms)	Avg Results	Avg Rel.
Graph Kernel	586.6	7.6	**1.00**
Keyword	**3.3**	20.0	**1.00**
BM25	9.2	18.8	**1.00**
RAG++	348.1	10.0	0.40

- **Graph Kernel** returns 7.6 structurally connected results forming verified relationship chains (e.g., Mohamed → works_on → clawdbot → uses → Gemini batch API). Each result is causally linked through graph edges.
- **Keyword** returns 20.0 coincidence results—documents that happen to contain matching substrings but with no concept of *why* the terms co-occur.

The relevance metric (Section IV) masks this critical quality difference. In production, the Graph Kernel's causal chains enable provenance-tracked reasoning; keyword coincidence does not. RAG++ drops to 0.40 because conversation turns rarely contain multi-hop relationship chains in a single text span.

The Graph Kernel's higher latency (586.6ms) on multi-hop queries results from sequential HTTP round-trips: a 2-hop query requires 3 HTTP calls × ∼200ms RTT. A server-side traversal endpoint (Section VIII) would reduce this to a single call.

D. Fuzzy/Semantic Search

TABLE VIII
FUZZY/SEMANTIC SEARCH RESULTS

Method	Avg Lat. (ms)	Avg Results	Avg Rel.
Graph Kernel	215.2	19.8	0.42
Keyword	**2.0**	16.0	**0.80**
BM25	6.1	7.6	0.53
RAG++	484.0	10.0	0.65

Fuzzy/semantic search is the Graph Kernel's weakest category, confirming the expected limitation of a system without embedding-based similarity. Searching for "music" will not find triples about "audio production" or "sound design." This is the primary motivation for the hybrid architecture (Section III-E). RAG++ outperforms all triple-based methods on semantic queries due to its vector-similarity backbone, though even it achieves only 0.65—suggesting that the query set includes genuinely difficult semantic matching tasks.

E. Predicate-Specific Queries

The Graph Kernel should excel on predicate-specific queries (exact predicate filters are a native capability), but entity normalization failures suppress relevance. One query for "Dream Weaver" files returned 0 results due to a capitalization/alias mismatch with the stored subject "dream-weaver-engine." This is a data quality issue, not an algorithmic limitation.

TABLE IX
PREDICATE-SPECIFIC QUERY RESULTS

Method	Avg Lat. (ms)	Avg Results	Avg Rel.
Graph Kernel	230.1	16.0	0.80
Keyword	**3.3**	20.0	**1.00**
BM25	9.5	20.0	**1.00**
RAG++	460.8	10.0	0.80

TABLE X
OVERALL RESULTS SUMMARY

Method	Lat.	Results	Rel.	Lat. Rk	Rel. Rk
Keyword	**2.8**	19.1	**0.96**	1	1
BM25	8.5	15.4	0.91	2	2
Graph Kernel	291.7	11.0	0.84	3	3
RAG++	407.9	10.0	0.70	4	4

F. Overall Summary

G. Latency Decomposition

TABLE XI
LATENCY DECOMPOSITION

Component	Contribution	% Total
Network RTT to Supabase	∼180–200 ms	∼68%
PostgreSQL query exec	∼5–20 ms	∼5%
TCP connection overhead	∼10–15 ms	∼5%
Rust serialization + JSON	∼1–2 ms	<1%
Multi-hop per add'l hop	+200 ms	+68%/hop
Projected (local SQLite)	**10–30 ms**	—

Critical insight. The Graph Kernel is compute-efficient. Its latency problem is an architecture choice (remote Supabase), not a fundamental limitation. A latency decomposition reveals that 90% of response time is network I/O. Migrating to local SQLite with periodic Supabase sync reduces query latency to 10–30ms, making it competitive with BM25.

VI. COMPARATIVE ANALYSIS

We compare the Graph Kernel against nine industry-grade alternatives across seven dimensions: context slicing, provenance, multi-hop traversal, semantic search, scale, deployment complexity, and cost.

A. Comparison Matrix

B. Key Findings

No existing system provides deterministic context slicing. Across all nine alternatives, none offers native, policy-governed context window construction with cryptographic provenance. This is the Graph Kernel's irreplaceable contribution. To replicate this functionality with Neo4j, for example, one would need to build a custom application layer implementing BFS expansion with priority scoring, HMAC token issuance, deterministic fingerprinting, and policy registry management—essentially re-implementing the Graph Kernel on top of a more complex substrate.

TABLE XII
COMPARATIVE ANALYSIS: GRAPH KERNEL VS. INDUSTRY ALTERNATIVES

System	Ctx Slice	Provenance	Multi-hop	Semantic	Scale	Deploy	Cost
Graph Kernel	✓ Native	✓ HMAC	✓	—	Small	Single binary	Free
Neo4j [6]	— Custom	—	✓✓	—	Large	JVM	Free/$$$
Amazon Neptune [36]	—	—	✓✓	—	Massive	AWS managed	$$$
Apache Jena [10]	—	Partial	✓	—	Medium	JVM	Free
Dgraph [37]	—	—	✓✓	✓	Massive	Distributed	Free/$$
TypeDB [38]	—	—	✓✓	—	Large	JVM	Free
Weaviate [2]	—	—	Limited	✓✓	Large	Go binary	Free/$$
LC/LI KGs [22], [23]	—	—	✓	✓	Varies	Python	Free+LLM
GraphRAG [21]	—	—	✓	✓✓	Medium	Python	Free+LLM
Zep [28]	—	—	✓	✓	Medium	Go/Cloud	Free/$$

General-purpose graph databases offer superior query expressiveness and scale. Neo4j (Cypher), Amazon Neptune (SPARQL/Gremlin/openCypher), Apache Jena (SPARQL), and TypeDB (TypeQL) all provide significantly more expressive query languages than the Graph Kernel's REST field filters. For applications requiring complex graph pattern matching, recursive path queries, or billion-scale traversal, these systems are superior. The Graph Kernel does not compete on this axis.

Vector databases address the Graph Kernel's primary weakness. Weaviate's hybrid search (vector + BM25 + filters) and GraphRAG's community-level semantic understanding directly address the 0.42 fuzzy/semantic relevance that is the Graph Kernel's weakest dimension. The hybrid architecture (Section III-E) is designed to bridge this gap.

Memory-oriented systems (Zep) optimize for developer experience at the cost of determinism. Zep provides automatic entity extraction, temporal memory decay, and drop-in integration with LLM applications. However, it offers no formal reproducibility guarantees—the same query at different times may return different results due to memory decay and re-summarization.

LLM-dependent systems (GraphRAG, LangChain/LlamaIndex KGs) are non-deterministic by construction. Both graph construction and query answering depend on LLM behavior, which varies across model versions, temperature settings, and even inference infrastructure. The Graph Kernel's determinism guarantee—same input, same output—is fundamentally incompatible with LLM-in-the-loop systems.

C. Positioning

The Graph Kernel does not compete with Neo4j on query expressiveness, Neptune on scale, Weaviate on semantic search, or GraphRAG on holistic corpus understanding. It occupies a distinct niche:

> *The Graph Kernel is a **provenance engine**—its value proposition is not information retrieval but the construction of verifiable, reproducible evidence bundles for autonomous agent systems.*

This positioning implies that the Graph Kernel is most valuable as a *layer in a stack* rather than a standalone system. In the OpenClaw CompCore architecture, it serves as the context authority layer between raw conversation storage (PostgreSQL) and agent reasoning (LLM inference), providing the trust boundary that ensures every downstream decision can be traced to a specific, authorized context window.

VII. DISCUSSION

A. The Provenance Gap

Our evaluation reveals a significant gap in the current AI infrastructure landscape: no widely-deployed system provides deterministic, cryptographically-verifiable context windows for autonomous agents. This is not because the problem is unrecognized—the AI safety community has extensively discussed the need for interpretable, auditable AI systems [39], [40]—but because the infrastructure to support these properties has not been built.

The Graph Kernel addresses this gap by making provenance a first-class architectural concern rather than an afterthought. The `AdmissibleEvidenceBundle` type, the HMAC-signed admissibility token, and the policy governance framework together ensure that every context window carries a complete provenance chain: what was included, why it was included (policy), and proof that the inclusion was authorized (token).

B. Limitations

Entity normalization. The evaluated corpus exhibits a 22% subject fragmentation rate, with 169 raw subjects collapsing to 132 canonical entities and 123 identified alias pairs (e.g., "Dream Weaver" $\neq$ "dream-weaver-engine", "GCP" $\neq$ "Google Cloud Platform"). This fragmentation directly suppresses query relevance: relationship queries drop from a projected 1.00 to the measured 0.94, and predicate-specific queries from $\sim$0.95+ to 0.80. A canonical alias table with fuzzy matching at ingestion and query time addresses this limitation.

No semantic search. The Graph Kernel achieves only 0.42 average relevance on fuzzy/semantic queries, confirming that exact field matching cannot substitute for embedding-based similarity. This is a fundamental limitation of the structural approach and motivates the hybrid architecture (Section III-E).

Client-side multi-hop. Multi-hop queries currently require sequential HTTP round-trips (N hops = $N+1$ calls $\times$ $\sim$200ms

RTT), resulting in 586.6ms average latency for 2-hop queries. A server-side traversal endpoint would collapse this to a single call, reducing latency to ∼220ms (remote) or ∼15ms (local).

Scale. The evaluated corpus contains 3,502 triples across 221 subjects. The Graph Kernel has not been tested at scales beyond tens of thousands of triples. For billion-scale graph applications, purpose-built distributed graph databases (Neo4j, Neptune, Dgraph) remain necessary.

Relevance metric limitations. Our relevance metric (fraction of expected terms found) does not capture the structural quality of results. As demonstrated in Section V, keyword search and Graph Kernel can achieve identical 1.00 relevance scores while producing qualitatively different results (coincidence piles vs. causal chains). Future evaluations should incorporate structural metrics such as chain validity and provenance completeness.

C. Broader Implications

The provenance engine concept has implications beyond the specific Graph Kernel implementation:

1) **Regulatory compliance.** As AI regulation evolves (EU AI Act, NIST AI RMF), the ability to audit which information influenced an agent's decision will become a compliance requirement, not an optional feature.
2) **Multi-agent trust.** In systems where multiple agents collaborate, provenance tokens enable trust delegation: Agent A can verify that Agent B's context window was authorized by a shared policy authority without accessing the authority's signing secret.
3) **Deterministic debugging.** When an agent produces an unexpected output, deterministic context slicing enables exact reproduction of the input conditions—same anchor, same policy, same graph state yields the same context window.

VIII. Future Work

A. Server-Side Traversal API

The highest-priority improvement is a server-side traversal endpoint (`POST /api/knowledge/traverse`) that executes multi-hop BFS within a single database transaction. This eliminates the $N\times$ RTT multiplier for multi-hop queries, reducing 2-hop latency from ∼600ms to ∼220ms (remote) or ∼15ms (local).

B. Entity Normalization Pipeline

A canonical entity resolution system with alias tables and fuzzy matching at both ingestion and query time. Projected impact: overall relevance improvement from 0.84 to 0.92+.

C. RAG++ Integration Bridge

The `POST /api/enrich` endpoint that combines vector-similarity results with graph context, addressing the 0.42 fuzzy/semantic relevance while preserving provenance guarantees. This bridge transforms RAG from flat similarity search into structured, provenance-tagged reasoning.

D. Local-First Deployment

SQLite-based local storage with periodic sync to the remote PostgreSQL backend. Reduces query latency from 291ms to projected 10–30ms, making the Graph Kernel competitive with in-memory BM25 on latency while maintaining provenance guarantees.

E. Federated Graph Protocol

A multi-kernel federation protocol enabling distributed knowledge across agent systems with cross-kernel queries and provenance chain propagation. This extends the single-agent provenance model to multi-agent collaboration scenarios.

F. Community Detection

Applying the Leiden algorithm [41] (as in GraphRAG) to identify hierarchical communities within the knowledge graph, enabling "what is this corpus about?" queries that the current system cannot answer.

G. Structural Relevance Metrics

Development of evaluation metrics that capture the structural quality of results (chain validity, provenance completeness, subgraph connectivity) rather than relying solely on term-overlap relevance scores.

IX. Conclusion

We have presented the Graph Kernel, a deterministic context slicing engine that introduces the provenance engine category to the AI agent infrastructure stack. Through evaluation across 27 queries and four retrieval methods, we demonstrate that the Graph Kernel achieves perfect relevance on multi-hop structural queries while providing properties that no existing system offers: HMAC-signed deterministic context windows, policy-governed access control, and type-level enforcement of admissibility invariants.

The Graph Kernel is not a general-purpose search engine (0.84 overall relevance vs. keyword's 0.96), nor a general-purpose graph database (REST field filters vs. Cypher/SPARQL), nor a semantic retrieval system (0.42 fuzzy relevance). It occupies a distinct niche: the provenance authority layer for autonomous agent systems. In this role, it is—among the ten systems we evaluated—irreplaceable.

As autonomous AI agents assume greater responsibility in production systems, the infrastructure to ensure their decisions are traceable, reproducible, and verifiable becomes not optional but essential. The Graph Kernel is a first step toward that infrastructure.

Appendix A
Reproducibility Statement

All benchmark scripts, raw results, and evaluation code are available in the OpenClaw CompCore repository at `benchmarks/run_benchmark.py`. The Graph Kernel binary can be compiled from source with `cargo build --release --features service`. The evaluation corpus (3,502 triples) is stored in the Supabase PostgreSQL instance; a SQLite snapshot is maintained at

`~/.compcore/graph-kernel.db` for offline evaluation. Complete benchmark results in JSON format are archived at `/tmp/benchmark_results.json`.

Appendix B
Corpus Statistics

TABLE XIII
CORPUS STATISTICS

Metric	Value
Total triples	3,502
Unique subjects	221
Unique predicates	88
Data sources	kimi-k2-extraction, topology-ingester
Average confidence	0.73 (Kimi), 0.95 (topology)
Top predicate	has_file (810, 23.1%)
Second predicate	needs_to (467, 13.3%)

Appendix C
Predicate Taxonomy

The 88 unique predicates cluster into four semantic categories:

- **Structural (40%):** `is_a`, `has_file`, `has_path`, `uses`, `depends_on`, `integrates_with`
- **Intentional (34%):** `needs_to`, `should`, `wants_to`, `building`
- **Relational (12%):** `works_on`, `likes`, `deployed_on`
- **Descriptive (14%):** `full_name`, `code`, `port`

The dominance of structural and intentional predicates reflects the Graph Kernel's primary data source: LLM-extracted knowledge from development-focused conversations, where entities are software systems and relationships encode architectural dependencies and development intentions.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[2] B. Mohr, E. Bueno de Mesquita, and S. van Cranenburgh, "Weaviate: An open-source vector database," 2023. [Online]. Available: https://weaviate.io

[3] J. J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2021.

[4] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[5] Y. Gao *et al.*, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[6] Neo4j, Inc., "Neo4j Graph Database," 2024. [Online]. Available: https://neo4j.com

[7] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč, "Foundations of modern query languages for graph databases," *ACM Computing Surveys*, vol. 50, no. 5, pp. 1–40, 2017.

[8] J. F. Sowa, "Semantic networks," in *Encyclopedia of Artificial Intelligence*, 2nd ed., S. C. Shapiro, Ed. Wiley, 1992.

[9] A. Singhal, "Introducing the Knowledge Graph: Things, not strings," Google Official Blog, May 2012. [Online]. Available: https://blog.google/products/search/introducing-knowledge-graph-things-not/

[10] Apache Software Foundation, "Apache Jena," 2024. [Online]. Available: https://jena.apache.org

[11] Blazegraph, "Blazegraph Database," 2024. [Online]. Available: https://blazegraph.com

[12] N. Francis *et al.*, "Cypher: An evolving query language for property graphs," in *Proc. 2018 International Conference on Management of Data (SIGMOD)*, 2018, pp. 1433–1445.

[13] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition, and applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 2, pp. 494–514, 2022.

[14] A. Hogan *et al.*, "Knowledge graphs," *ACM Computing Surveys*, vol. 54, no. 4, pp. 1–37, 2021.

[15] A. Hogan *et al.*, "Knowledge graphs," *Synthesis Lectures on Data, Semantics, and Knowledge*, vol. 12, no. 2, pp. 1–257, Morgan & Claypool, 2022.

[16] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang, "Retrieval augmented language model pre-training," in *Proc. 37th ICML*, 2020, pp. 3929–3938.

[17] G. Izacard and E. Grave, "Leveraging passage retrieval with generative models for open domain question answering," in *Proc. 16th EACL*, 2021, pp. 874–880.

[18] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to retrieve, generate, and critique through self-reflection," in *Proc. 12th ICLR*, 2024.

[19] Pinecone Systems, Inc., "Pinecone: Vector Database for Machine Learning," 2024. [Online]. Available: https://www.pinecone.io

[20] Chroma, "Chroma: The AI-native open-source embedding database," 2024. [Online]. Available: https://www.trychroma.com

[21] D. Edge *et al.*, "From local to global: A graph RAG approach to query-focused summarization," *arXiv preprint arXiv:2404.16130*, 2024.

[22] J. Liu, "LlamaIndex: A data framework for LLM applications," 2024. [Online]. Available: https://www.llamaindex.ai

[23] H. Chase, "LangChain," 2024. [Online]. Available: https://www.langchain.com

[24] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[25] M. Zaheer *et al.*, "Big Bird: Transformers for longer sequences," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 17283–17297.

[26] W. Zhong, L. Guo, Q. Gao, H. Ye, and Y. Wang, "MemoryBank: Enhancing large language models with long-term memory," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 38, 2024.

[27] A. Bulatov, Y. Kuratov, and M. S. Burtsev, "Recurrent memory transformer," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 11079–11091.

[28] Zep, "Zep: Long-term memory for AI assistants," 2024. [Online]. Available: https://www.getzep.com

[29] P. Buneman, S. Khanna, and W.-C. Tan, "Why and where: A characterization of data provenance," in *International Conference on Database Theory (ICDT)*, 2001, pp. 316–330.

[30] J. Cheney, L. Chiticariu, and W.-C. Tan, "Provenance in databases: Why, how, and where," *Foundations and Trends in Databases*, vol. 1, no. 4, pp. 379–474, 2009.

[31] L. Moreau, P. Missier, *et al.*, "PROV-DM: The PROV Data Model," W3C Recommendation, 2013. [Online]. Available: https://www.w3.org/TR/prov-dm/

[32] R. Liang, C. Niu, F. Zhang, Z. Qi, and Y. Hao, "Blockchain-based data provenance for AI: A comprehensive survey," *IEEE Access*, vol. 11, pp. 61748–61770, 2023.

[33] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519

[34] D. Tolnay *et al.*, "Axum: Ergonomic and modular web framework built with Tokio, Tower, and Hyper," 2024. [Online]. Available: https://github.com/tokio-rs/axum

[35] C. Lerche *et al.*, "Tokio: An asynchronous runtime for the Rust programming language," 2024. [Online]. Available: https://tokio.rs

[36] Amazon Web Services, "Amazon Neptune," 2024. [Online]. Available: https://aws.amazon.com/neptune/

[37] Dgraph Labs, "Dgraph: Distributed graph database," 2024. [Online]. Available: https://dgraph.io

[38] Vaticle, "TypeDB: The polymorphic database," 2024. [Online]. Available: https://typedb.com

[39] D. Amodei *et al.*, "Concrete problems in AI safety," *arXiv preprint arXiv:1606.06565*, 2016.

[40] S. Weidinger *et al.*, "Ethical and social risks of harm from language models," *arXiv preprint arXiv:2112.04359*, 2021.
[41] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: Guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 1, p. 5233, 2019.