Enterprise Knowledge Chat System: A Hybrid AI Architecture for Multi-Repository Code Intelligence

Asveth Vijayakumar

September 1, 2025

Abstract

Enterprise organizations operate hundreds of repositories across products and teams, fragmenting technical knowledge and slowing decision-making. Traditional code search and conversational assistants lack enterprise context, cross-repository identity, and explainable results. We present a hybrid AI system that unifies vector semantics, a simplified enterprise knowledge graph, and fast symbol search with product/team scoping. The system extracts repository facts, enriches them with large language models (LLMs), and serves fast, evidence-grounded answers with source attribution. Our architecture delivers sub-second symbol lookups, multi-second semantic synthesis, repository-level embeddings for high-level context, and explainable payloads suitable for enterprise audit and governance.

1. Introduction

Large enterprises maintain diverse ecosystems of repositories, technologies, and teams. Knowledge about architecture, dependencies, and security is scattered across code, documentation, configurations, and issue trackers (Potvin & Levenberg 2016, Sadowski et al. 2015). Conventional approaches, from grep-style search to modern embedding search, address parts of the problem but fall short in cross-repository identity, enterprise scoping, and evidence-grounded answers (Manning et al. 2008, Johnson et al. 2019).

This paper introduces a hybrid AI architecture designed to transform repository analysis into enterprise-grade conversational intelligence. Our contributions are:

- **Hybrid Search Orchestration:** A query classifier that selects among symbol, vector, and graph strategies, balancing speed and depth for each question type.
- **Unified Enterprise Payload:** A single vector collection with payload-based multi-tenancy that stores both technology-level and repository-level embeddings enriched with product, team, and identity metadata.
- **Flat Knowledge Graph for Scale:** A simplified Neo4j schema (single node label with rich properties and indexes) that supports fast enterprise queries and cross-product rollups.
- **LLM-Enhanced Explainability:** Evidence payloads with validation flags, confidence breakdowns, and semantic usage, enabling auditable, risk-aware answers.
- **Operational Performance:** Sub-100ms symbol queries, multi-second deep synthesis, health/analytics endpoints, and resilient fallbacks for external services.

2. Related Work and Limitations

2.1 Sourcegraph and Cody

Sourcegraph provides high-quality code search and semantic indexing with embeddings and precise symbol navigation (*Sourcegraph: Code Search and Cody* n.d.). For enterprise-wide questions, three gaps typically appear: (1) product/team multi-tenancy is modeled at search-time but not embedded as first-class payload schema; (2) repository-level business context (purpose, features, architecture) is not embedded alongside code-level points; (3) explainable cross-repository identity (what repeats across products) is not emphasized as a structured, queryable entity.

2.2 GitHub Code Search and Copilot Chat

GitHub Code Search (*GitHub: Code Search* n.d.) plus Copilot Chat (*GitHub Copilot* n.d.) enables powerful workspace assistance. However, it is oriented to a developer's local task context. It does not focus on enterprise scoping across products and owners, nor on KG-backed identity to unify technologies across repositories; evidence payloads for audit and governance are out of scope.

2.3 OpenGrok and Grep-Based Systems

Index-based text search (e.g., OpenGrok (*OpenGrok* n.d.)) scales well for large monoliths but misses semantics, cross-repo identity, and LLM-backed enrichment. It lacks repository-level summarization, confidence breakdowns, and provenance as part of the retrieval payload.

2.4 CodeQL and Static Analysis Frameworks

Static analysis (e.g., CodeQL (*CodeQL* n.d.), Semgrep) offers precise program reasoning and security checks but is rule-centric, not built for open-ended enterprise knowledge queries or multi-repository synthesis. It also does not natively return evidence payloads designed for conversational audit.

2.5 IDE Assistants (e.g., JetBrains)

IDE assistants improve productivity within a single workspace but lack an enterprise knowledge layer with unified identities, repository-level embeddings, and explainable evidence payloads shared across products.

2.6 Common Limitations Observed

Across these systems, we find recurring limitations: (1) limited, ad hoc enterprise scoping and tenancy; (2) weak repository-level summarization of purpose and architecture; (3) absent or fragile cross-repository identity; (4) no principled hybrid orchestration to trade speed vs. depth; (5) limited provenance and confidence instrumentation for audit.

3. Key Innovations

3.1 Agentic Reasoning Mode

An agent orchestrates tool calls for multi-step questions (Yao et al. 2023, Schick et al. 2023). Tools include: hybrid_search for enterprise-scoped retrieval, analyze_repositories for repository summaries, and compare_technologies for cross-product comparisons. Typical latency is 15 to 20 seconds with sourced answers.

The agent plans tool usage, invokes tools with constraints, aggregates evidence, and synthesizes a concise answer with confidence and citations. The agent shares the same evidence payloads as the hybrid engine, which keeps outputs explainable and auditable.

Entities and Contracts. The agent reasons over three primary entity types: (1) *query intents* classified into lookup, analysis, or comparison; (2) *scopes* that carry enterprise filters (products, repositories, teams, categories) and hard limits (result caps, time budgets); and (3) *evidence records* returned by tools, each with identity, enterprise attributes, location, context, and confidence components. These entities form the agent's working set and are immutable within a plan iteration for auditability.

Planning and Execution. Planning translates a user request into a sequence of bounded tool interactions. The agent prioritizes narrow scopes first (e.g., product-scoped lookups) to meet latency budgets, then widens only if confidence or coverage is insufficient. Evidence is aggregated into a transient evidence graph with deduplication by unified identity and scored with preference heuristics (validation method, scope match, freshness). Synthesis produces an answer that references the strongest evidence segments and carries a summarized confidence profile.

Resilience and Resource Governance. The agent enforces per-step time budgets and maximum result counts to avoid unbounded fan-out. On partial failures, the plan degrades by omitting unavailable strategies and marking coverage gaps in the synthesized response. Session-level caches of recent evidence allow follow-up queries to reuse prior retrievals, reducing redundant work and tail latency.

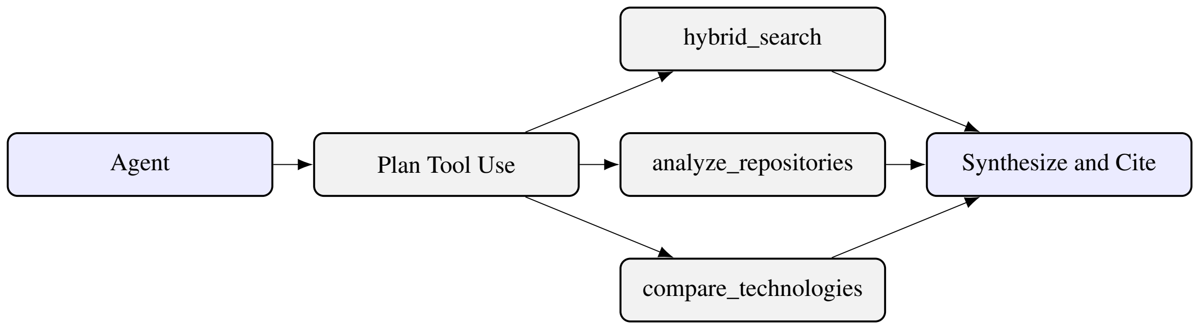

Figure 1: Agentic plan, call, and synthesize loop with citations.

3.2 Unified Enterprise Payload with Repository-Level Embeddings

A single vector collection holds technology points and repository-level points (README-derived) with product, team, owner, and identity metadata (Lewis et al. 2020, *Qdrant Vector Database* n.d., OpenAI n.d.).

Enterprise Entities. Two payload types coexist: *technology points* (name, category, location, truncated context) and *repository summaries* (purpose, features, architecture), both enriched with enterprise attributes (product, team, owner) and identity (unified and instance identifiers). Unified identity supports cross-repository consolidation; instance identity supports per-repository provenance and rollups.

Tenancy and Filtering. Payload-based multi-tenancy stores all entities in a single logical collection, enabling consistent indexing and global deduplication. Query-time filtering is applied via enterprise attributes and categories, ensuring that consumers can scope retrievals without maintaining separate physical indices.

Versioning and Governance. Each evidence record carries timestamps, a loader version, and confidence metadata. This enables time-bounded queries (latest vs snapshot), supports reproducible evaluations, and provides a governance trail for model and extractor evolution.

We construct repository-level embeddings from curated fields (summary, purpose, features, architecture) and attach enterprise metadata as payload attributes. Technology points include name, category, location, and truncated context, plus enterprise attributes (product, team) and identity (unified identity and repository instance identity). This design allows search-time filtering, cross-repo rollups, and evidence-rich result composition.

3.3 Hybrid Search with Intelligent Strategy Selection

A classifier routes each query to symbol-only (fast), vector-only (semantic), hybrid-fast (symbol+vector), hybrid-deep (symbol+vector+graph), or enterprise-scoped strategies (Manning et al. 2008).

Planner and Budgets. The strategy planner maps intents to execution graphs under explicit latency budgets. *Hybrid-fast* prioritizes exactness and returns early if confidence is sufficient; *hybrid-deep* augments coverage with semantic and identity-aware signals when the query demands breadth or synthesis.

Fusion and Ranking. Results from symbol, vector, and graph sources are normalized into a common evidence schema, deduplicated by unified identity, and ranked by a weighted score that considers observed confidence, validation method, query-scope alignment, and freshness. Repository summaries are interleaved with technology points when repository-level context improves answerability.

Caching and Adaptation. Hot symbol results and frequent repository summaries are cached with short TTLs to meet p95 latency targets while preserving freshness. The planner adapts dynamically, skipping expensive strategies if coverage is already sufficient or budgets are exhausted.

Lexical patterns and intent cues determine the initial plan. Symbol paths serve exact or pattern lookups in-memory to meet sub-100ms targets. If recall is insufficient or semantics are required, vector search augments results. For cross-product context, graph queries surface technologies shared across products. A deduplication and ranking step merges sources with preference heuristics and confidence signals.

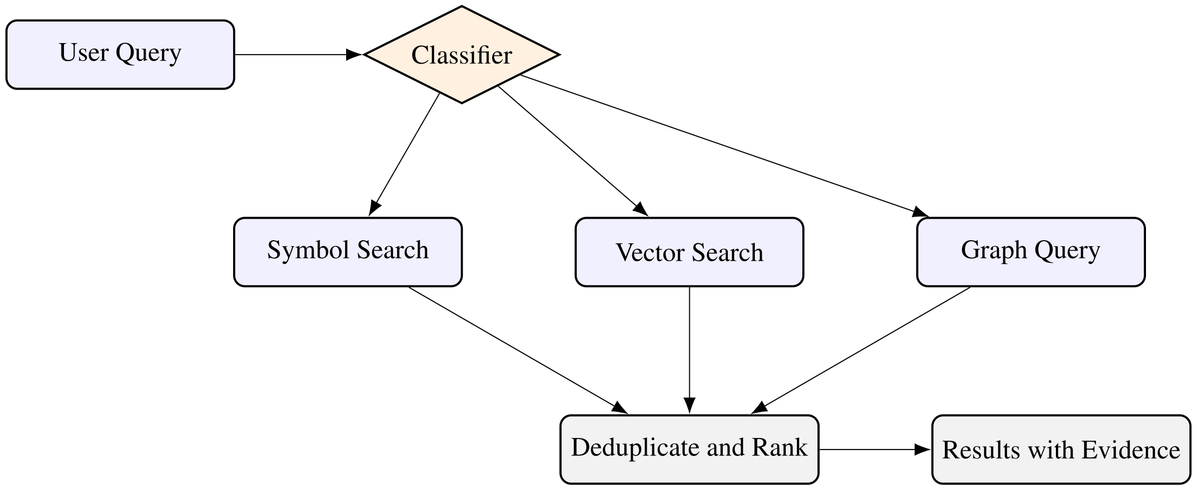

Figure 2: Strategy selection and hybrid composition.

3.4 Cross-Repository Identity with a Flat Knowledge Graph

A property-rich, index-optimized graph stores entities with *unified* and *instance* identities for technologies (*Neo4j Graph Database* n.d., Robinson et al. 2015, Hogan et al. 2021).

Identity Semantics. Unified identity binds equivalent technologies across repositories, while instance identity anchors observations to specific repositories with provenance. This dual identity enables *shared-across-products* analyses, trend detection, and technology standardization without deep traversals.

Query Patterns. Common queries include: shared technologies across two or more products; technology lineage and version rollups; and repository cohorts by category. The flat schema emphasizes indexed property filters over relationship depth to provide predictable latencies at scale.

A single node label with targeted indexes supports selective enterprise queries. Each technology stores a cross-repository identity and a repository-instance identity to compute shared-across-products metrics, trend analyses, and rollups with predictable performance.

3.5 LLM-Enhanced Evidence and Explainability

Evidence payloads include LLM-derived validation flags, confidence breakdowns, and semantic usage descriptors (Mitchell et al. 2019).

Evidence Lifecycle and Calibration. During extraction, pattern detectors emit candidates that are optionally validated by LLM signals; the resulting confidence breakdown (pattern, LLM, cross-validation) is stored alongside identity and provenance. At retrieval, confidence is exposed to consumers to inform risk-aware decisions, and calibration is periodically assessed against human judgements to maintain alignment.

During extraction, LLM signals complement pattern detection to annotate each point with validation and confidence components. At retrieval, evidence is assembled into the response, preserving file paths, categories, and provenance to enable audit and risk-aware decisions.

3.6 Enterprise Scoping and Operations

Product, repository, team, and category filters; optional inclusion of repository-level results; streaming, health, and analytics.

Filters are translated to payload predicates on vector searches and to property predicates on graph queries. Symbol indices are kept in-memory with TTL-based caches. Streaming uses server-sent events to progressively deliver metadata and results.

4. System Architecture

Figure 3 presents a simplified layout that includes the agent orchestrator. This section details the responsibilities of each component and the data flow between them to remove ambiguity and ensure the design is reproducible. Vector storage uses Qdrant (*Qdrant Vector Database* n.d.) and knowledge storage uses Neo4j (*Neo4j Graph Database* n.d.). Embeddings use OpenAI models (OpenAI n.d.), and the web/API layer is built on modern frameworks (Ramírez n.d.).

4.1 Data Flow and Interfaces

Ingestion and Analysis. Source content (code, documentation, configuration) is scanned to identify technologies, extract evidence locations, and summarize repositories. Outputs are normalized into structured analysis artifacts with identity and enterprise attributes.

Loading. Enterprise loaders persist technology points to the vector store (single collection, payload-based tenancy) and entities to the knowledge store (flat, property-rich schema). Identity and governance metadata are attached at load time to support reproducibility and audit.

Retrieval. The hybrid engine receives a user query and a *scope* (products, repositories, teams, categories) and executes symbol, vector, and graph strategies according to the planner's latency budget. Results are fused into a unified evidence set.

Delivery. The API exposes synchronous and streaming interfaces that return ranked evidence and summarized answers, with confidence and provenance included for downstream consumers.

4.2 Component Contracts

Analyzer guarantees deterministic normalization with provenance (locations and hashes). *Loaders* guarantee idempotent writes using stable identities. *Hybrid engine* guarantees at-least-one strategy execution with structured evidence and performance metadata. *API* guarantees bounded response sizes and optional streaming for progressive rendering.

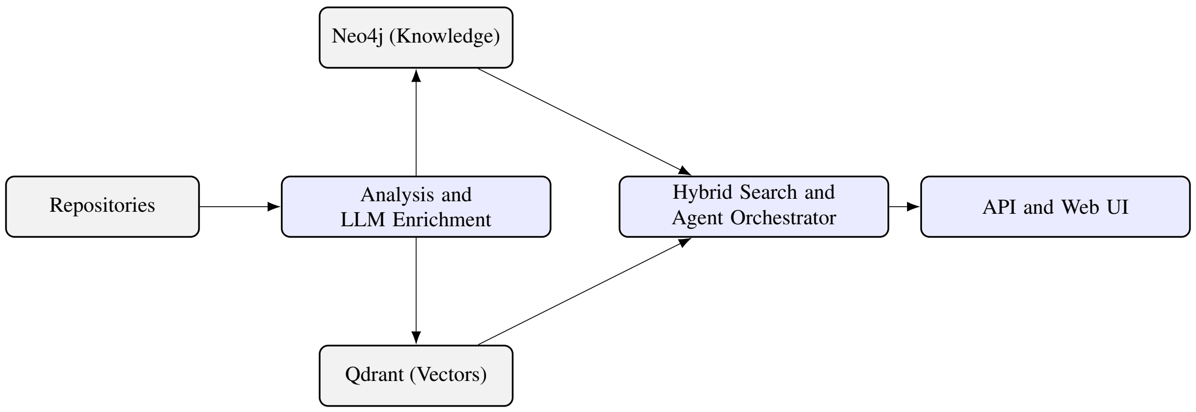
Neo4j (Knowledge)
Repositories
Analysis and LLM Enrichment
Hybrid Search and Agent Orchestrator
API and Web UI
Qdrant (Vectors)

Figure 3: Architecture with a combined hybrid engine and agent orchestrator, which plans tool calls and synthesizes sourced answers.

4.3 Component Responsibilities

Repositories are the ground truth. The *Analysis and LLM Enrichment* service extracts technologies, README context, and security findings, optionally validating evidence with LLM signals.

Qdrant (Vectors) stores a single enterprise collection that includes both technology-level and repository-level embeddings with rich payloads (product, team, unified identity, instance identity). **Neo4j (Knowledge)** stores entities in a flat schema with properties optimized for enterprise filtering and identity.

The **Hybrid Search and Agent Orchestrator** layer routes queries across symbol, vector, and graph strategies, merges evidence, and exposes search to the **API and Web UI**. The agentic mode shares the same evidence payloads while planning multi-step tool calls.

4.4 Agentic Reasoning Mode in the Stack

The agent orchestrator uses the same data and tools as the hybrid engine. It selects and invokes tools (hybrid search, repository analysis, and technology comparison), aggregates evidence, and synthesizes a cited answer. Tool outputs remain explainable because they rely on the unified evidence payload.

5. End-to-End Extraction and Enrichment Flow

The pipeline proceeds in five stages, with an optional sixth stage for agentic reasoning.

1. **Ingestion and Analysis:** Repositories are scanned to extract technologies, security findings, files analyzed, and README content.

2. **Embedding Creation:** Technology-level and repository-level embeddings are generated with enterprise payloads.

3. **Graph Loading:** Entities are written to the knowledge graph with unified and per-repository identities and targeted indexes.

4. **Hybrid Orchestration:** A query classifier routes to symbol, vector, and graph strategies with optional enterprise scoping.

5. **Agentic Reasoning (Optional):** The agent plans tool calls, aggregates evidence, and synthesizes a cited answer.

6. **Delivery and Telemetry:** The API and Web UI stream responses with metadata and performance metrics.

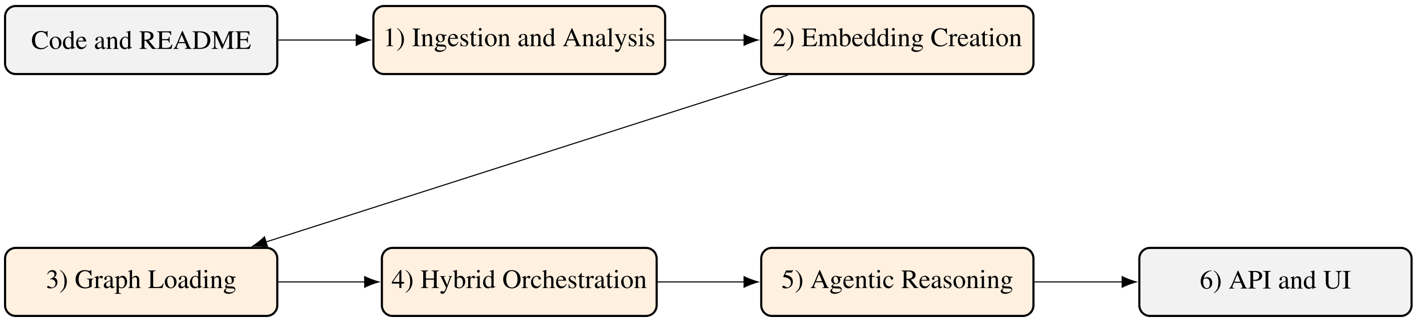

Figure 4: End-to-end flow with an optional agentic reasoning stage prior to delivery.

6. Search Experience and Evidence-Grounded Answers

Hybrid versus Agentic. Hybrid search serves precise lookups with sub-100ms symbol paths and multi-second semantic retrieval. Agentic reasoning addresses multi-step questions. It plans tool use, calls hybrid search and analysis tools, then synthesizes a cited answer.

Enterprise Scoping Mechanics. Scoping parameters are translated into payload predicates for vector search and property predicates for graph queries, which constrains candidate sets and improves precision and latency.

Evidence Composition. Each result includes identity, enterprise attributes, location, semantic context, and confidence. These fields support audit and explain why a result was returned.

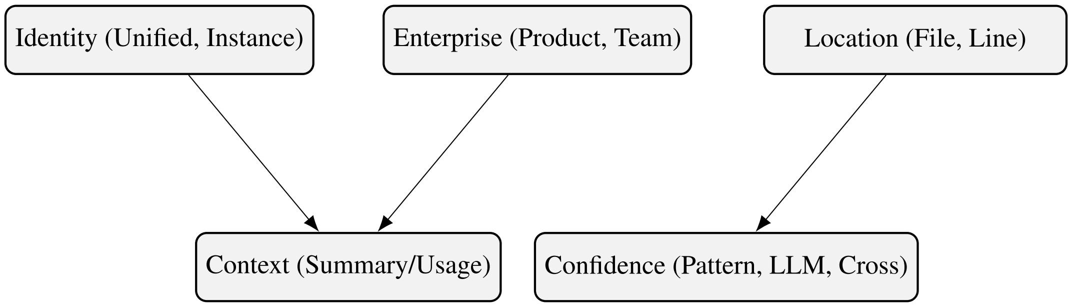
Identity (Unified, Instance)
Enterprise (Product, Team)
Location (File, Line)
Context (Summary/Usage)
Confidence (Pattern, LLM, Cross)

Figure 5: Evidence composition combines identity, enterprise attributes, location, context, and confidence.

Quality and Performance at a Glance. We target the 95th percentile (p95) latency of under 100ms for exact symbol lookups, under 2.5 seconds for hybrid-fast queries, under 3.5 seconds for hybrid-deep, and under 20 seconds for multi-step agentic answers (Dean & Barroso 2013). Quality is tracked via Precision at k, Recall at k (Manning et al. 2008), attribution accuracy (correct file path and category), and confidence calibration against human judgements (Guo et al. 2017).

Streaming and Health. The API supports responsive UIs via Server-Sent Events (WHATWG n.d.): `GET /search/stream` emits `metadata`, incremental `result` items, and a `complete` token. A simple health probe `GET /health` reports component readiness and basic counters.

Evidence Schema Snapshot. Results include an evidence-rich metadata payload with identity and confidence fields suitable for audit:

```
{
  "tech_name": "PostgreSQL" | "Repository",
  "product_name": "...",
  "repository_name": "...",
  "source": "symbol|vector|graph",
  "confidence": 0.86,
  "metadata": {
    "file_path": "src/db/config.cs",
    "line_number": 120,
    "validation_method": "llm_enhanced|pattern_only",
    "llm_validated": true,
    "confidence_breakdown": {
      "pattern_confidence": 0.62,
      "llm_confidence": 0.73,
      "cross_validation": 0.55
    },
    "unified_tech_id": "unified_postgresql",
    "repository_instance_id": "Product_Repo_PostgreSQL"
  }
}
```

7. Conclusion

This work presents an architecture for enterprise code intelligence that unifies vector search, a simplified knowledge graph, and fast symbol search. A classifier selects strategies for speed and depth. Repository-level embeddings add high-level context, while technology points capture concrete evidence with validation and confidence. The agentic mode extends the system to multi-step questions by planning tool calls and synthesizing sourced answers. Together these components provide fast lookups, explainable responses, and enterprise scoping suitable for audit and governance.

References

CodeQL (n.d.), `https://codeql.github.com/`. Accessed: 2025-09-01.

Dean, J. & Barroso, L. A. (2013), 'The tail at scale', *Communications of the ACM* **56**(2), 74–80.

GitHub: Code Search (n.d.), `https://docs.github.com/en/search-github/github-code-search`. Accessed: 2025-09-01.

GitHub Copilot (n.d.), `https://github.com/features/copilot`. Accessed: 2025-09-01.

Guo, C., Pleiss, G., Sun, Y. & Weinberger, K. Q. (2017), On calibration of modern neural networks, *in* 'Proceedings of ICML'.

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutiérrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S. & Zimmermann, A. (2021), 'Knowledge graphs', *ACM Computing Surveys* **54**(4).

Johnson, J., Douze, M. & Jégou, H. (2019), Billion-scale similarity search with gpus, *in* 'IEEE Transactions on Big Data'.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., K"uttler, H., Lewis, M., tau Yih, W., Rockt"aschel, T., Riedel, S. & Kiela, D. (2020), Retrieval-augmented generation for knowledge-intensive nlp, *in* 'Proceedings of NeurIPS'.
URL: *https://arxiv.org/abs/2005.11401*

Manning, C. D., Raghavan, P. & Sch"utze, H. (2008), Introduction to information retrieval, Cambridge University Press.

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D. & Gebru, T. (2019), 'Model cards for model reporting', *Proceedings of the Conference on Fairness, Accountability, and Transparency* .

Neo4j Graph Database (n.d.), `https://neo4j.com/`. Accessed: 2025-09-01.

OpenAI (n.d.), 'text-embedding-3-small', `https://platform.openai.com/docs/guides/embeddings`. Accessed: 2025-09-01.

OpenGrok (n.d.), `https://oracle.github.io/opengrok/`. Accessed: 2025-09-01.

Potvin, R. & Levenberg, J. (2016), Why google stores billions of lines of code in a single repository, *in* 'Communications of the ACM', Vol. 59, pp. 78–87.

Qdrant Vector Database (n.d.), `https://qdrant.tech/`. Accessed: 2025-09-01.

Ramírez, S. (n.d.), 'Fastapi', `https://fastapi.tiangolo.com/`. Accessed: 2025-09-01.

Robinson, I., Webber, J. & Eifrém, E. (2015), *Graph Databases*, O'Reilly Media.

Sadowski, C., Soederberg, E., Church, L., Maddila, C., Miller, A., Murphy, C. G., Samant, R. & Winter, C. (2015), Questions developers ask while diagnosing potential security vulnerabilities with static analysis, *in* 'Proceedings of CHI'.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N. & Scialom, T. (2023), Toolformer: Language models can teach themselves to use tools, *in* 'Proceedings of NeurIPS'.
URL: *https://arxiv.org/abs/2302.04761*

Sourcegraph: Code Search and Cody (n.d.), `https://sourcegraph.com/`. Accessed: 2025-09-01.

WHATWG (n.d.), 'Server-sent events', `https://html.spec.whatwg.org/multipage/server-sent-events.html`. Accessed: 2025-09-01.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. & Cao, Y. (2023), React: Synergizing reasoning and acting in language models, *in* 'Proceedings of ICLR'.
URL: *https://arxiv.org/abs/2210.03629*