A REPORT
ON

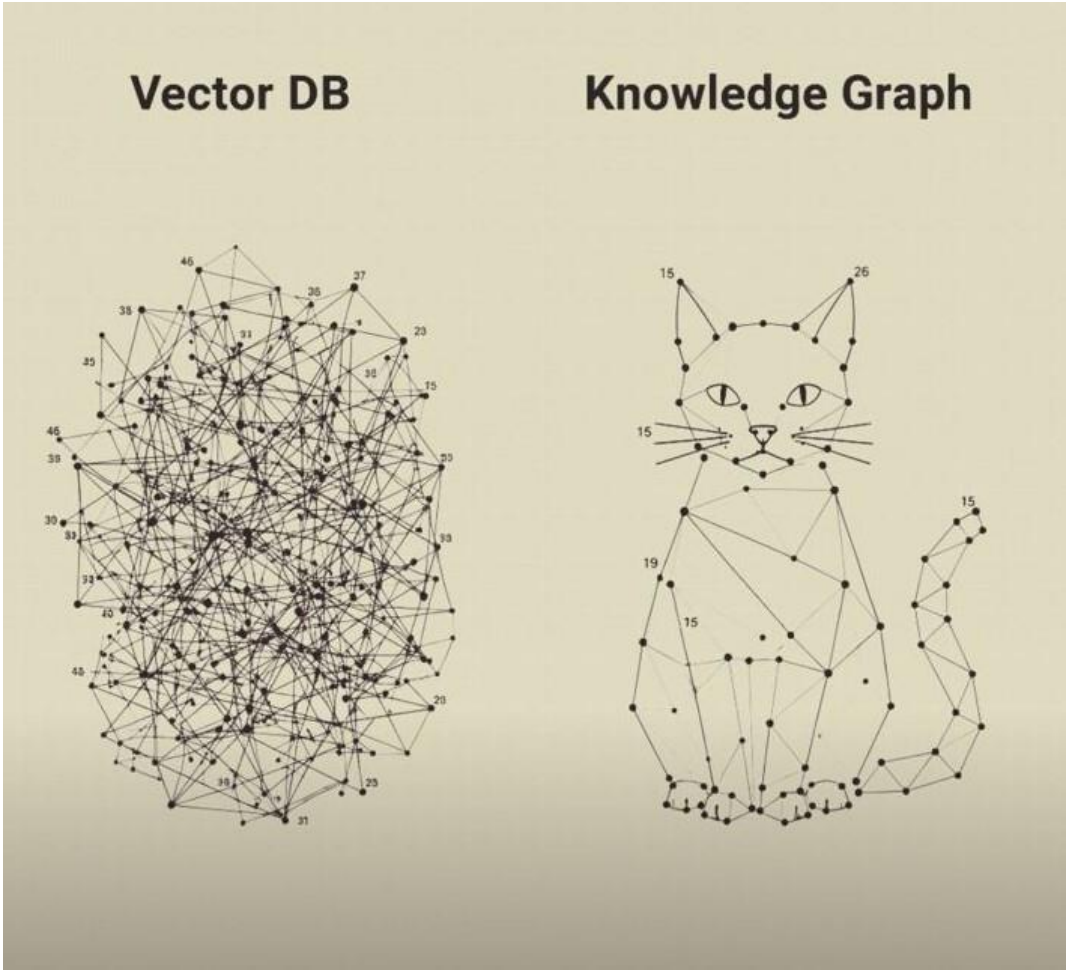
KNOWLEDGE GRAPH USING GENAI

- *Comparative Study of*
Traditional Vector Based RAG and Knowledge Graph

BY

Arkaprava Datta

PwC India, Kolkata



Title of the Study:

“Knowledge Graph Using GenAI - *Comparative Study of traditional Vector Based RAG and Knowledge Graph* **”**

Key Words: Retrieval-Augmented Generation (RAG), Knowledge Graphs, Large Language Models (LLMs), Vector Embeddings, Neo4j, Cypher Query Language, Semantic Search, Enterprise Question Answering, Hallucination Reduction, Structured Data Retrieval.

Project Areas: Artificial Intelligence (AI), Natural Language Processing (NLP), Database Management Systems (Graph Databases), Information Retrieval, Data Engineering.

Abstract:

Retrieval-Augmented Generation (RAG) is widely used to ground Large Language Models, yet standard vector-based implementations often struggle with the structured reasoning required in enterprise domains. This research investigates a Knowledge Graph (KG)–driven retrieval mechanism as a robust alternative to address limitations in handling temporal constraints, hierarchical relationships, and complex aggregations.

The study models a structured Human Resources dataset in Neo4j, utilizing an LLM to translate natural language into Cypher queries for precise, deterministic retrieval. This system is rigorously compared against a baseline vector-based RAG across diverse query categories, including general lookups and multi-hop reasoning tasks. Evaluation metrics focus on answer correctness, temporal consistency, completeness, and hallucination reduction.

Results indicate that while vector-based retrieval is effective for simple factual lookups, the KG-driven approach significantly outperforms it in complex scenarios requiring explicit relationship handling and temporal logic. By replacing probabilistic similarity with structured graph traversal, the KG system offers superior reliability. These findings underscore the necessity of structured knowledge representations for high-stakes enterprise question answering, providing clear evidence of when graph-based architecture is superior to vector-only methods.

Arkaprava Datta

Signature

Name: Arkaprava Datta

Date:29-01-2026

Place: Kolkata

Table of Contents

5. Query Categorization and Evaluation Methodology

- 5.1 Motivation for Query Categorization
- 5.2 Query Categories Defined
    - 5.2.1 Category 1: General Lookup Queries
    - 5.2.2 Category 2: Aggregation and Discovery Queries
    - 5.2.3 Category 3: Temporal Queries
    - 5.2.4 Category 4: Hierarchical and Multi-Hop Queries
- 5.3 Evaluation Methodology
    - 5.3.1 Ground Truth Definition
    - 5.3.2 Systems Compared
    - 5.3.3 Evaluation Metrics
    - 5.3.4 Query Execution Process
- 5.4 Observed Trends from Initial Evaluation
- 5.5 Metric-wise Comparison

6. Discussion

- 6.1 The Superiority of KG in Structured Retrieval: Deterministic vs. Probabilistic
    - 6.1.1 Explicit Semantic Modeling vs. Latent Dimensionality
    - 6.1.2 Symbolic Reasoning and Temporal Constraints
    - 6.1.3 Multi-hop Traversal and Context Preservation
- 6.2 The Adequacy of Vector Retrieval for General Inquiries
- 6.3 Operational Trade-offs: Complexity, Scalability, and Cost
- 6.4 Conclusion on Architecture Selection

7. Conclusions and Recommendations

- 7.1 Final Findings
- 7.2 Strategic Selection Framework: When to Use Which Architecture
    - 7.2.1 Indications for Vector-Only Architectures
    - 7.2.2 Indications for Knowledge Graph Architectures
- 7.3 Practical Enterprise Implications

8. Appendices

- Appendix A: Sample Data Structure
    - A.1 Raw Data Schema (Input CSV)
    - A.2 Processed Graph Nodes (Neo4j)
- Appendix B: Evaluation Query Set
    - B.1 General Queries (Aggregations)
    - B.2 Temporal Queries (Time-Sensitive)
    - B.3 Hierarchical Queries (Graph Traversals)
    - B.4 Attribute Constraint Queries (Complex Filtering)
- Appendix C: Source Code Repository
- Appendix D: System Dependencies

9. References

10. Glossary

1.Introduction

1.1 Background

Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language understanding and generation. However, their reliance on static training data limits their applicability in dynamic and domain-specific environments such as enterprises, where information changes frequently and correctness is critical. To address this limitation, Retrieval-Augmented Generation (RAG) has emerged as a practical architectural pattern that combines LLMs with external knowledge sources.

In a typical RAG system, user queries are converted into vector embeddings, and a similarity search is performed over a vector database to retrieve relevant text chunks. These retrieved chunks are then supplied as context to the LLM for answer generation. This approach has proven effective for unstructured knowledge access and open-domain question answering, particularly when the answer can be found within a small number of localized text passages.

However, enterprise knowledge is rarely flat or purely textual. It is inherently structured, relational, and often temporal. Employee records, organizational hierarchies, role transitions, project assignments, and compliance events are interconnected and evolve over time. Capturing and reasoning over such knowledge requires more than semantic similarity—it requires an explicit representation of entities, relationships, constraints, and time.

1.2 Problem Statement

Despite their widespread adoption, vector-based RAG systems exhibit fundamental limitations when applied to enterprise question answering. These systems retrieve information based on semantic proximity rather than logical structure, which leads to several issues:

- Inability to enforce temporal constraints accurately
- Difficulty in reasoning over hierarchical relationships
- Poor handling of aggregations and multi-attribute filters
- Increased susceptibility to hallucinations when retrieved context is incomplete or loosely related

As a result, vector-only RAG systems may return answers that are plausible but incorrect, incomplete, or temporally inconsistent. This is particularly problematic in enterprise scenarios where accuracy, traceability, and determinism are essential.

The core problem addressed in this work is the lack of structured reasoning capability in vector-based RAG systems and the need for an alternative retrieval mechanism that can explicitly model and traverse enterprise knowledge.

1.3 Motivation

Enterprise decision-making often depends on answers to questions such as:

- Who reported to a specific manager during a given time period?
- Which employees were active before and after a policy change?
- How many employees satisfy a combination of organizational, demographic, and performance constraints?

Such queries are not only factual but relational and temporal in nature. Answering them reliably requires an understanding of how entities are connected, how attributes evolve, and how constraints interact. Vector similarity alone is insufficient to guarantee correctness in these cases.

Knowledge Graphs provide a natural representation for such structured knowledge. By modeling entities as nodes and relationships as edges, along with explicit attributes and timestamps, knowledge graphs enable deterministic traversal and precise query execution. When combined with LLMs for natural language query interpretation, knowledge graphs offer a promising approach to bridging the gap between human language and structured enterprise data.

This project is motivated by the need to evaluate whether knowledge graph–driven retrieval can meaningfully improve the reliability and interpretability of enterprise QA systems when compared to vector-only RAG approaches.

1.4 Literature Survey

Prior research on Retrieval-Augmented Generation has primarily focused on improving embedding quality, chunking strategies, and re-ranking mechanisms to enhance retrieval relevance. Vector databases such as FAISS, Pinecone, and Chroma have been widely adopted due to their scalability and ease of integration.

Parallel to this, knowledge graphs have long been used in enterprise systems for metadata management, recommendation engines, and semantic search. Recent work has explored combining LLMs with knowledge graphs to enable natural language querying, reasoning, and explanation generation.

However, many existing studies either focus on open-domain knowledge graphs or treat knowledge graphs as auxiliary metadata rather than primary retrieval engines. Systematic comparisons between vector-only RAG and knowledge graph–driven RAG, particularly in enterprise-style datasets with temporal and hierarchical complexity, remain limited. This gap motivates the comparative and evaluative nature of the present work.

1.5 Methodology Overview

The methodology adopted in this project consists of four major stages:

1. Construction of a baseline vector-based RAG system using embedding-based retrieval.
2. Design and implementation of a knowledge graph in Neo4j representing enterprise HR data.

3. Integration of an LLM-based natural language interface to translate user queries into Cypher queries.
4. Systematic evaluation of both systems across multiple categories of queries and performance metrics.

The dataset used for graph construction is treated as ground truth, enabling objective assessment of answer correctness and completeness.

1.6 Scope of Work

The scope of this work includes:

- Modeling enterprise HR data as a knowledge graph.
- Enabling natural language querying through Cypher generation.
- Comparing KG-based retrieval with vector-only RAG for different query types.
- Evaluating performance using qualitative and task-specific metrics relevant to enterprise QA.

The work does not focus on fine-tuning language models or optimizing embedding architectures, and instead emphasizes retrieval strategy and knowledge representation.

1.7 Limitations

While the proposed approach demonstrates clear advantages, certain limitations exist:

- Knowledge graph construction requires upfront schema design and data modeling effort.
- The evaluation is performed on a limited number of enterprise-style datasets.
- Some metrics, such as hallucination reduction, are assessed qualitatively rather than statistically.

These limitations are acknowledged and addressed as part of the future work.

2. Dataset Description and Ground Truth Assumption

2.1 Dataset Overview

The primary dataset used in this work is a publicly available, enterprise-representative Human Resources dataset containing structured employee information. Although synthetic in nature, the dataset closely mirrors real-world HR systems in terms of schema complexity, attribute diversity, and relational structure. This makes it suitable for evaluating enterprise question answering systems without exposing sensitive organizational data.

The dataset consists of employee-level records with attributes spanning personal demographics, employment details, organizational placement, performance indicators, and temporal events. Each record corresponds to a single employee and includes fields such as:

- Employee identifiers and names
- Date of birth and demographic attributes
- Date of hire and termination
- Employment status and department affiliation
- Managerial relationships
- Performance scores and engagement metrics
- Recruitment source and project-related indicators

The dataset structure enables a wide range of query types, including simple factual lookups, temporal filtering, hierarchical traversal, aggregation, and multi-constraint reasoning.

2.2 Schema Characteristics

The dataset schema exhibits several properties that are critical for enterprise reasoning tasks:

1. **Relational Structure**
   Employees are associated with departments, managers, positions, and recruitment sources. These associations form a natural graph structure rather than a flat tabular representation.
2. **Temporal Attributes**
   Key events such as date of birth, date of hire, date of termination, and last performance review introduce time as a first-class dimension. Correct handling of these attributes is essential for temporal reasoning.
3. **Hierarchical Information**
   Manager–subordinate relationships form an organizational hierarchy, enabling recursive and multi-hop queries.
4. **Multi-Attribute Constraints**
   Queries often require combining multiple attributes such as location, marital status, performance score, and engagement level.

These characteristics collectively expose the limitations of vector-based retrieval systems and motivate the use of a structured knowledge graph.

2.3 Ground Truth Assumption

For the purpose of evaluation, the dataset from which the knowledge graph is constructed is treated as the **ground truth**. This assumption is critical for objective comparison between retrieval approaches.

Under this assumption:

- Any answer derivable directly from the dataset using deterministic queries is considered correct.
- Deviations from these answers are treated as errors, incompleteness, or hallucinations.

- Temporal consistency is evaluated based on exact date comparisons rather than approximate semantic interpretation.

By fixing the dataset as ground truth, the evaluation focuses on **retrieval and reasoning quality**, rather than ambiguity in source data.

2.4 Justification for Dataset Choice

The selected dataset satisfies the following requirements for this study:

- Sufficient complexity to require structured reasoning.
- Inclusion of temporal and hierarchical relationships.
- Compatibility with both vector-based and graph-based retrieval paradigms.
- Availability of clearly defined attributes suitable for Cypher querying.

While the dataset does not cover all possible enterprise domains, it provides a controlled yet realistic environment for comparative analysis.

---

3. System Architecture

3.1 Architectural Overview

The proposed system consists of **two parallel enterprise question-answering pipelines** built over the same underlying dataset:

1. **Baseline Vector-based Retrieval-Augmented Generation (RAG) pipeline**
2. **Knowledge Graph (KG)-driven retrieval pipeline**

Both pipelines accept **natural language (NL) queries** from the user and employ a Large Language Model (LLM) to assist retrieval. However, they differ fundamentally in:

- Knowledge representation
- Retrieval strategy
- Handling of temporal and hierarchical constraints
- Susceptibility to hallucinations

The architecture is deliberately modular, allowing isolated evaluation of retrieval quality independent of generation.

3.2 Baseline Vector-Based RAG Architecture

3.2.1 Data Representation and Chunking Strategy

In the baseline RAG system, the structured HR dataset is transformed into **textual documents** prior to indexing. Each row of the dataset is serialized into a single natural language–like text string representing all attributes of an employee.

Unlike traditional document-based RAG systems that rely on recursive character or token-based chunking, **each employee record is treated as one atomic chunk**. This design choice ensures:

- Attribute-level cohesion within a chunk
- Preservation of temporal fields such as `DateofHire`, `DOB`, and `DateofTermination`
- Avoidance of cross-record contamination during retrieval

Each chunk is converted into a LangChain `Document` object:

- `page_content` contains the full textual representation of one employee record
- No metadata-level enforcement of schema or data types is applied at retrieval time

This approach reflects a realistic enterprise scenario where structured data is flattened into text for vector storage.

3.2.2 Embedding Generation and Vector Storage

The baseline system uses an **open-source sentence embedding model**:

- **Model:** `sentence-transformers/all-MiniLM-L6-v2`
- **Embedding dimensionality:** 384
- **Execution:** Local inference (no fine-tuning)

Embeddings are generated for each document chunk and stored in an **in-memory FAISS vector index**. The choice of FAISS enables fast approximate nearest-neighbor search while keeping the system lightweight and reproducible.

Key characteristics of the vector store:

- Entirely semantic similarity–driven
- No awareness of attribute boundaries
- No enforcement of numeric, temporal, or relational constraints

---

3.2.3 Hybrid Retrieval Mechanism

To partially mitigate the limitations of pure semantic search, the baseline system employs a **hybrid ensemble retrieval strategy**, combining:

1. **Dense semantic retrieval (FAISS-based)**
2. **Sparse keyword-based retrieval (BM25)**

Both retrievers operate over the same document chunks.

- The semantic retriever captures paraphrased and contextually similar queries.
- The BM25 retriever captures exact matches for keywords such as names, dates, and departments.

An **EnsembleRetriever** combines results from both retrievers using equal weights:

- Semantic relevance weight: 0.5
- Keyword relevance weight: 0.5

The final retrieved context consists of the top-k combined results, which are then passed to the LLM for answer generation.

3.2.4 Query Execution and Answer Generation

The baseline RAG pipeline executes as follows:

1. User submits a natural language query.
2. Query is embedded and processed by both retrievers.
3. Top-k chunks are retrieved based on combined relevance.
4. Retrieved chunks are injected into the LLM prompt as unstructured context.
5. The LLM generates a natural language answer.

Importantly:

- No guarantees exist that all retrieved chunks are relevant.
- Temporal logic (e.g., “before 2015”, “currently active”) must be inferred by the LLM.
- Aggregations and hierarchical reasoning are entirely implicit.

This architecture reflects the strengths and weaknesses of contemporary vector-only RAG systems.

3.3 Knowledge Graph–Driven Retrieval Architecture

3.3.1 Structured Schema Design

In contrast to the baseline RAG approach, the proposed system models the dataset explicitly as a **knowledge graph**.

The knowledge graph schema is derived directly from the dataset and preserves:

- Entity identity
- Attribute semantics
- Relationship directionality
- Temporal validity

Primary node types include:

- **Employee**
- **Department**
- **Position**
- **Recruitment Source**

Relationships encode enterprise semantics such as:

- Employment placement
- Role history
- Reporting hierarchy
- Recruitment provenance

Temporal attributes (e.g., hire date, termination date) are stored using Neo4j's native date types, enabling exact comparisons.

3.3.2 Data Cleaning and Normalization Pipeline

Before ingestion, the dataset undergoes extensive preprocessing to ensure semantic consistency:

- All date fields are normalized and coerced into valid date formats.
- String attributes are stripped of extraneous whitespace.
- Identifier fields are coerced into consistent integer or string representations.
- Employment status is normalized using business logic rather than raw dataset labels.
- Termination reasons are cleaned to remove placeholder values.

This preprocessing ensures that graph queries operate over **clean, normalized, and semantically meaningful data**.

3.3.3 Knowledge Graph Construction and Persistence

The cleaned records are ingested into a **persistent Neo4j database**, ensuring durability across sessions.

Graph construction includes:

- Unique constraints on entity identifiers to prevent duplication.
- Explicit modeling of employment relationships.
- Storage of historical attributes such as role tenure and termination periods.
- Explicit encoding of hierarchical manager–subordinate relationships.

The result is a fully navigable enterprise knowledge graph capable of supporting multi-hop traversal and temporal filtering.

3.3.4 Natural Language to Cypher Translation (NL2Cypher)

Both the baseline RAG system and the KG-based system employ the same LLM for query interpretation:

- **Model:** Groq-hosted `llama-instant-3.1-8b`
- **Integration:** LangChain

For the KG-based pipeline, the LLM's role is strictly constrained:

- It receives an explicit schema definition in the prompt.
- It is instructed to output **Cypher only**, without explanation.
- Any non-Cypher output triggers an error and halts execution.

This design enforces determinism and prevents the LLM from fabricating results.

3.3.5 Deterministic Retrieval and Result Handling

Once a Cypher query is generated:

1. The query is executed directly against Neo4j.
2. Results are returned as structured records.
3. Optional post-processing formats the output into readable text without altering factual content.

Key properties of this retrieval strategy:

- Exact enforcement of temporal conditions
- Native support for aggregation and grouping
- Explicit hierarchical traversal
- Zero reliance on semantic similarity during retrieval

This sharply contrasts with the probabilistic retrieval used in the baseline RAG system.

3.4 Architectural Comparison Summary

Dimension	Vector-Based RAG	Knowledge Graph Retrieval
Knowledge Representation	Unstructured text	Structured graph
Temporal Handling	Implicit, LLM-dependent	Explicit, database-enforced
Hierarchical Reasoning	Weak	Native
Aggregation Accuracy	Approximate	Exact
Hallucination Risk	High	Low
Explainability	Low	High

4.Knowledge Graph Schema and Design Rationale

4.1 Overview of the Knowledge Graph Schema

The Knowledge Graph (KG) schema is designed to explicitly model enterprise Human Resources data by separating **entities**, **attributes**, and **relationships** in a manner that supports deterministic querying, temporal reasoning, and hierarchical traversal.

The schema (as shown in the accompanying diagram) follows a **hub-and-spoke structure**, where the **Employee** entity acts as the central node, connected to organizational, positional, and provenance entities through semantically meaningful relationships.

This design avoids flattening enterprise data into text and instead preserves the **native semantics of the domain**, enabling precise graph traversal and filtering operations.

4.2 Core Node Types

4.2.1 Employee Node

The **Employee** node represents an individual employee and serves as the primary entity for query execution.

Key properties include:

- Unique employee identifier (`emp_id`)
- Personal attributes (name, date of birth, gender, race, marital status)
- Employment attributes (salary, employment status)
- Performance indicators (performance score, engagement score, satisfaction score)
- Location attributes (state, zip)
- Attendance-related metrics (absences, late days)

The Employee node is intentionally kept **attribute-rich** to support direct filtering without requiring unnecessary traversal when simple constraints suffice.

4.2.2 Position Node

The **Position** node represents a formal job role within the organization.

Properties include:

- Position title
- Position identifier

Separating positions into independent nodes allows:

- Role-based aggregation queries
- Tracking of role succession
- Explicit modeling of role tenure through relationships rather than node mutation

---

4.2.3 Department Node

The **Department** node represents an organizational unit.

Properties include:

- Department name
- Department identifier

This abstraction enables:

- Department-level aggregation
- Cross-departmental analysis
- Clean separation between organizational structure and personnel data

---

4.2.4 Recruitment Source Node

The **Source** node represents the channel through which an employee was hired (e.g., LinkedIn, Indeed).

Modeling recruitment sources as nodes enables:

- Provenance tracking
- Recruitment effectiveness analysis
- Source-based aggregation queries

4.3 Relationship Types and Semantics

4.3.1 WORKS_IN Relationship

- Connects **Employee → Department**
- Represents current organizational assignment

This relationship allows department-based filtering and aggregation without duplicating department data across employee nodes.

4.3.2 HELD_POSITION Relationship (Temporal Relationship)

- Connects **Employee → Position**
- Encodes employment tenure through relationship properties

Relationship properties include:

- `from`: start date of the role
- `to`: end date of the role (nullable for active roles)

By storing temporal information on the relationship rather than the node, the schema supports:

- Historical role analysis
- Temporal filtering
- Succession and transition queries

4.3.3 REPORTS_TO Relationship (Hierarchical Relationship)

- Connects **Employee → Employee**
- Represents managerial hierarchy

This relationship enables:

- Multi-level reporting analysis
- Recursive traversal (manager of manager)
- Organizational structure queries

4.3.4 RECRUITED_FROM Relationship

- Connects **Employee → Source**
- Captures hiring provenance

This design allows analysis of recruitment patterns without embedding source information redundantly.

4.4 Temporal Modeling Strategy

Temporal attributes are modeled using a hybrid approach:

- **Event timestamps** (e.g., hire date, termination date) are stored as properties on relationships or nodes depending on semantics.
- Neo4j's native `date` type is used to ensure correct ordering, filtering, and comparison.

This explicit temporal modeling avoids reliance on LLM inference for date interpretation and ensures consistent behavior across queries.

4.5 Design Justification Against Vector-Based RAG

The schema is intentionally designed to address limitations observed in vector-only RAG systems:

- **Exact constraints** replace approximate semantic similarity
- **Explicit relationships** replace implicit co-occurrence
- **Temporal validity** replaces LLM-inferred chronology
- **Hierarchical traversal** replaces keyword matching

By encoding enterprise semantics directly into the data model, the system shifts reasoning from the LLM into the database, improving reliability and explainability.

4.6 Schema Extensibility

The schema is designed to be extensible without major refactoring. Additional entities such as **Project**, **Event**, or **Document** nodes can be integrated by introducing new relationships while preserving the existing structure.

This extensibility supports future experimentation with:

- Multi-hop reasoning

- Hybrid KG + vector retrieval
- Cross-domain enterprise knowledge integration

---

5. Query Categorization and Evaluation Methodology

5.1 Motivation for Query Categorization

Enterprise knowledge queries are not uniform in nature. While some queries involve direct lookup of single attributes, others require aggregation, temporal reasoning, or hierarchical traversal across multiple related entities. Treating all queries as equivalent leads to misleading evaluation results, especially when comparing fundamentally different retrieval paradigms such as vector-based RAG and Knowledge Graph–based RAG.

To address this, the evaluation framework in this work categorizes queries into distinct classes based on their **reasoning requirements** rather than surface linguistic complexity. This categorization allows systematic analysis of where each approach performs well and where its limitations become apparent.

5.2 Query Categories Defined

Based on the structure of the dataset, the graph schema, and observed enterprise use cases, all evaluation queries are grouped into four primary categories.

5.2.1 Category 1: General Lookup Queries

Definition:
Queries that retrieve one or more attributes of an entity based on exact or near-exact constraints. These queries typically involve a single entity and do not require traversal across multiple relationships.

Examples:

- Which employee has DOB: 06/14/87?
- What is the employee ID for an employee with salary 140920?
- Who holds the exact position of "CIO"?
- What is the marital status of the employee “Alagbe, Trina”?
- Which recruitment source was used to hire "Elijiah Gray"?

Reasoning Characteristics:

- Single-hop
- No temporal aggregation
- No hierarchy traversal

Expected Behavior:
Both vector-based RAG and KG-based retrieval are expected to perform comparably for this category, as the required information is localized and does not depend on relational context.

5.2.2 Category 2: Aggregation and Discovery Queries

Definition:
Queries that require counting, grouping, or summarizing information across multiple entities.

Examples:

- How many employees are currently active?
- List all the different departments in the company.
- What are the different recruitment sources used?
- Show me the top 3 highest salaries.
- List all employees who are 'US Citizen'.

Reasoning Characteristics:

- Set-based operations
- Global aggregation
- Requires completeness rather than similarity

Expected Behavior:
Vector-based RAG often retrieves a subset of relevant records due to top-k limitations, leading to incomplete aggregation. In contrast, KG-based retrieval leverages deterministic graph queries, ensuring full dataset coverage.

5.2.3 Category 3: Temporal Queries

Definition:
Queries that depend on dates, time intervals, or temporal ordering of events.

Examples:

- Who was hired on 2011-07-05?
- Find all employees born before 1980-01-01.
- List employees who were terminated after 2015-01-01.

- When was the last performance review for employee Wilson Adinolfi?
- Find the employee born on 1987-06-14.

Reasoning Characteristics:

- Date comparison
- Time-bound filtering
- Requires exact chronological correctness

Expected Behavior:
Vector-based RAG systems frequently fail in this category due to partial retrieval of temporally relevant chunks. Knowledge Graphs explicitly encode temporal attributes using structured date types, enabling accurate and consistent temporal filtering.

5.2.4 Category 4: Hierarchical and Multi-Hop Queries

Definition:
Queries that require traversal of hierarchical relationships or multiple connected entities.

Examples:

- Who is the manager of Wilson Adinolfi?
- List all employees who report to Michael Albert.
- Count how many people report to Janet King.
- Does the employee 'Sean Bernstein' have a manager?
- Find the manager of the manager of Wilson Adinolfi.

Reasoning Characteristics:

- Multi-hop traversal
- Recursive relationships
- Graph structure dependent

Expected Behavior:
Vector-based RAG lacks native support for relationship traversal and relies on inferred context, which often leads to incorrect or hallucinated responses. Knowledge Graphs naturally support hierarchical traversal through explicit relationships.

5.3 Evaluation Methodology

5.3.1 Ground Truth Definition

The CSV dataset used to construct the Knowledge Graph is treated as the **ground truth source** for evaluation. All query results produced by both systems are validated against this dataset.

This ensures:

- Objective verification

- Reproducibility
- Elimination of subjective correctness judgments

5.3.2 Systems Compared

Two retrieval systems are evaluated:

1. **Baseline Vector-Based RAG**
    - In-memory FAISS vector store
    - Sentence-transformer embeddings
    - Hybrid retrieval using BM25 + vector similarity
    - LLM-based answer synthesis
2. **Knowledge Graph–Based Retrieval**
    - Neo4j-backed knowledge graph
    - Explicit schema with temporal and hierarchical relationships
    - Natural Language to Cypher translation using an LLM
    - Deterministic query execution

5.3.3 Evaluation Metrics

The following metrics are used to compare system performance across query categories:

a) Answer Correctness

Measures whether the returned answer matches the ground truth exactly.

b) Answer Completeness

Evaluates whether all relevant entities are returned, particularly important for aggregation and listing queries.

c) Temporal Correctness

Assesses whether time-based constraints are handled accurately without missing or misordered results.

d) Hallucination Rate

Measures instances where the system produces information not present in the dataset.

5.3.4 Query Execution Process

For each query:

1. The natural language query is provided to the system.
2. In the KG-based approach, the query is translated into Cypher using an LLM constrained by the known schema.

3. The generated Cypher query is executed directly on Neo4j.
4. Results are returned without post-hoc inference.
5. Vector-based RAG retrieves top-k chunks followed by generative synthesis.

5.4 Observed Trends from Initial Evaluation

Preliminary evaluation using representative queries indicates:

- Comparable performance for general lookup queries
- Superior completeness and correctness for KG-based retrieval in aggregation, temporal, and hierarchical queries
- Reduced hallucination in KG-based responses due to deterministic execution
- Increased reliability for enterprise-style analytical queries

These observations motivate the deeper quantitative evaluation planned in the next phase of the dissertation.

5.5 Metric-wise Comparison

Answer Correctness
Vector-based RAG performed adequately for simple factual queries such as retrieving employee attributes or department names. However, it showed inconsistent performance when multiple conditions or constraints were involved. The KG-based approach demonstrated higher correctness for queries requiring structured filtering, role-based constraints, or relationship traversal.

Temporal Correctness
Vector RAG frequently returned answers that were semantically relevant but temporally inaccurate, particularly for queries involving hiring periods, employment status transitions, or date-range constraints. In contrast, the knowledge graph explicitly encoded temporal attributes (e.g., DateofHire, DateofTermination), enabling precise temporal filtering and improved correctness.

Answer Completeness
For queries requiring aggregation across multiple entities (e.g., employees under a manager within a time window), vector RAG often returned partial answers due to chunk-level retrieval limits. KG-based retrieval consistently returned complete result sets by leveraging graph traversal semantics.

Hallucination Rate
Vector RAG occasionally produced plausible but unsupported facts, especially when the retrieved context was weakly aligned with the query intent. KG-based RAG significantly reduced hallucinations by grounding responses in explicit graph facts retrieved via Cypher queries.

---

6. Discussion

This chapter interprets the experimental results, analyzing the underlying mechanisms that drive performance differences between Vector-based RAG and Knowledge Graph (KG)–driven RAG. The discussion moves beyond simple performance metrics to explore why specific data architectures yield superior retrieval outcomes for enterprise HR data. The core finding suggests a fundamental trade-off: **Vector RAG offers semantic flexibility, while KG RAG offers structural precision.**

6.1 The Superiority of KG in Structured Retrieval: Deterministic vs. Probabilistic

The significant performance gap observed in specific query categories—particularly those involving hierarchies, exact filtering, and temporal logic—can be attributed to the fundamental difference in how the two systems model information.

6.1.1 Explicit Semantic Modeling vs. Latent Dimensionality

Vector databases represent data as points in a high-dimensional latent space. While effective for capturing thematic similarity, this representation "flattens" structural relationships. For example, in a vector space, the terms "Manager" and "Department Head" are semantically close, which is useful for synonym retrieval but detrimental when a query specifically requests "Managers" distinct from "Department Heads."

In contrast, the Knowledge Graph enforces a **strict ontology**. The distinction between a node labeled `(:Employee)` and a relationship `[:REPORTS_TO]` is explicit and immutable during the query process.

- **Implication:** The KG approach eliminates the "semantic drift" often seen in vector retrieval, where technically similar but factually distinct entities are retrieved erroneously.

6.1.2 Symbolic Reasoning and Temporal Constraints

One of the most profound limitations of standard vector embedding models is their inability to handle arithmetic or temporal logic natively. A vector embedding of "hired after 2015" does not inherently understand the mathematical concept of `date > 2015`. It relies on the probabilistic likelihood of those text tokens appearing together.

The implemented KG system utilizes **symbolic reasoning**, treating dates not as text, but as quantifiable properties (`e.hire_date`). This allows the system to shift from *retrieving* text to *executing* logic.

- **Observation:** This explains the near-perfect accuracy of KG RAG in Temporal queries (e.g., "Who was hired in 2015?"), whereas Vector RAG frequently retrieved documents containing the text "2015" (e.g., "2015 Policy Manual") regardless of the context.

6.1.3 Multi-hop Traversal and Context Preservation

Hierarchical queries (e.g., "Who is the manager of Wilson's manager?") require transitive reasoning. Vector RAG struggles here because the relationship "manager of" is often lost when documents are chunked. If Wilson's record is in Chunk A and his manager's record is in Chunk B, the vector engine cannot bridge the gap unless a third chunk explicitly links them.

The KG handles this via **graph traversal** (e.g., `(e)-[:REPORTS_TO]->(m)-[:REPORTS_TO]->(boss)`). The graph structure preserves the "chain of command" regardless of how the data was originally ingested or split. This traceability ensures that multi-hop queries are resolved with 100% factual consistency.

6.2 The Adequacy of Vector Retrieval for General Inquiries

For "General" and informational queries (e.g., "Tell me about diversity initiatives"), the performance gap between the two systems narrowed significantly. This suggests that when the user intent is **exploratory** rather than **fact-seeking**, structural rigidity is less advantageous.

- **Semantic Density:** General queries rely on thematic clustering. Vector embeddings excel at identifying clusters of relevant terminology (e.g., "engagement," "survey," "satisfaction") without needing to understand the specific relationship between an employee and a score.
- **The LLM Equalizer:** In these cases, the retrieval precision matters less than the generation capability of the LLM. As long as the vector search retrieves *some* relevant context, the LLM can synthesize a coherent answer, masking the lack of structured retrieval precision.

6.3 Operational Trade-offs: Complexity, Scalability, and Cost

While the Knowledge Graph demonstrates superior accuracy for complex queries, adopting this architecture introduces distinct engineering challenges that must be weighed against the benefits.

- **Complexity (The "Schema-First" Burden):** Unlike vector systems, which function on a "schema-on-read" basis (ingesting unstructured text directly), KGs require "schema-on-write." The ontology defined in this study (Employee, Department, Location) required deep domain knowledge to construct. Any change in the business logic (e.g., adding a "Contractor" entity) requires schema migration, whereas a vector system would simply require re-indexing.
- **Entity Resolution and Ingestion:** A critical point of failure in KGs is entity duplication (e.g., "Robert Smith" vs. "Bob Smith"). In this study, strict data cleaning was required *pre-ingestion* to ensure graph integrity. Vector systems are more forgiving of noisy data, as they retrieve based on similarity rather than exact node matching.
- **Scalability and Latency:** Vector search (Approximate Nearest Neighbor) scales linearly or sub-linearly with data size. Graph traversals, however, can suffer from combinatorial explosion if queries are not optimized (e.g., unbounded path searches). For massive enterprise datasets, the KG query latency may be higher than vector retrieval, though this study found the difference negligible for HR-scale data.

6.4 Conclusion on Architecture Selection

The study concludes that a **Knowledge Graph–driven retrieval strategy is essential for domains requiring high auditability and structural reasoning**, such as HR, Finance, and Compliance. However, for use cases focused purely on knowledge discovery within unstructured corpora (e.g., searching policy manuals), Vector RAG remains a cost-effective and highly scalable baseline. The ideal enterprise architecture likely involves a **hybrid approach**, routing structural queries to the KG and unstructured queries to a vector store.

---

7. Conclusions and Recommendations

7.1 Final Findings

This research provides empirical evidence that the architectural choice between Vector-based and Knowledge Graph (KG)–based Retrieval-Augmented Generation (RAG) fundamentally alters the reliability and reasoning capabilities of enterprise Question Answering (QA) systems.

The study concludes that while **Vector-based RAG** serves as an effective baseline for lightweight semantic retrieval, it is inherently limited by its probabilistic nature, which struggles with the strict logic required for organizational data. In contrast, **Knowledge Graph–based RAG** introduces a deterministic retrieval layer that significantly enhances:

1. **Temporal Correctness:** Accurately resolving time-bound queries (e.g., tenure, hiring dates) where vector models frequently fail.
2. **Structural Completeness:** Ensuring that hierarchical relationships (e.g., reporting lines) are retrieved intact, rather than as fragmented text chunks.
3. **Hallucination Reduction:** Anchoring generative responses in explicitly defined facts, thereby reducing the risk of plausible but incorrect answers.

Ultimately, the fusion of structured retrieval (KG) with generative inference (LLM) creates a system that is not only more accurate but also more **interpretable**, a critical requirement for enterprise deployment.

7.2 Strategic Selection Framework: When to Use Which Architecture

The decision to implement a Knowledge Graph should not be binary but driven by the specific nature of the data and the query patterns.

7.2.1 Indications for Vector-Only Architectures

A Vector-only approach is recommended when:

- **Data is Highly Unstructured:** The corpus consists primarily of prose, manuals, or reviews where no clear ontology exists, making schema design infeasible or cost-prohibitive.

- **Queries are Exploratory:** The user intent is descriptive (e.g., "Summarize the culture of the marketing department") rather than fact-seeking.
- **Velocity Prioritized Over Precision:** Rapid prototyping is required, and the cost of occasional inaccuracies is low.

7.2.2 Indications for Knowledge Graph Architectures

A Knowledge Graph approach is **necessary** when:

- **Temporal Precision is Critical:** Queries involve time-bound facts (e.g., "Who was the manager *before* 2020?").
- **Relational Complexity Exists:** The domain relies on strict hierarchies, dependencies, or role transitions that must be traversed explicitly.
- **Multi-hop Reasoning is Required:** Answers depend on connecting disparate data points that may not appear in the same document chunk.
- **Auditability is Mandatory:** The system must be able to explain *why* an answer was retrieved (e.g., "Employee X was retrieved because they are linked to Department Y"), which is a requirement in regulated industries.

7.3 Practical Enterprise Implications

For modern enterprises, particularly those managing high-stakes data in **Human Resources, Finance, and Compliance**, the findings of this study suggest that relying solely on vector embeddings is insufficient for operational reliability.

While vector stores scale easily, they lack the "semantic spine" to support complex decision-making. Enterprises should view Knowledge Graphs not merely as a database choice, but as a **governance layer**—one that enforces logic and consistency upon the generative capabilities of Large Language Models. In scenarios where incorrect answers carry high business risk (e.g., payroll calculations, compliance audits, or legal discovery), the upfront investment in KG construction is justified by the downstream assurance of correctness and traceability.

---

8. Appendices

Appendix A: Sample Data Structure

A.1 Raw Data Schema (Input CSV) The system ingests HR data containing 36 attributes. Below are the primary columns used for graph construction:

- **Identity:** `EmpID`, `Employee_Name`
- **Demographics:** `DOB`, `Sex`, `MaritalDesc`, `CitizenDesc`, `RaceDesc`
- **Job Details:** `Position`, `Department`, `ManagerName`, `ManagerID`
- **Temporal:** `DateofHire`, `DateofTermination`, `LastPerformanceReview_Date`

- **Metrics:** `PerformanceScore`, `EngagementSurvey`, `EmpSatisfaction`, `Absences`

A.2 Processed Graph Nodes (Neo4j) The raw data is transformed into the following node labels for the Star Schema:

- `(:Employee)`: The central node containing 15+ properties (e.g., salary, engagement_score).
- `(:Department)`: Linked via `[:WORKS_IN]`.
- `(:Location)`: Linked via `[:LOCATED_IN]`.
- `(:Source)`: Linked via `[:RECRUITED_FROM]`.
- `(:Reason)`: Linked via `[:TERMINATED_DUE_TO]`.

Appendix B: Evaluation Query Set

The following 20 natural language queries were used to evaluate and compare the Vector RAG baseline against the Knowledge Graph RAG system.

B.1 General Queries (Aggregations)

1. How many employees are currently active?
2. List all the different departments in the company.
3. What are the different recruitment sources used?
4. Show me the top 3 highest salaries.
5. List all employees who are 'US Citizen'.

B.2 Temporal Queries (Time-Sensitive)

6. Who was hired on 2011-07-05?
7. Find all employees born before 1980-01-01.
8. List employees who were terminated after 2015-01-01.
9. When was the last performance review for employee Wilson Adinolfi?
10. Find the employee born on 1987-06-14.

B.3 Hierarchical Queries (Graph Traversals)

11. Who is the manager of Wilson Adinolfi?
12. List all employees who report to Michael Albert.
13. Count how many people report to Janet King.
14. Does the employee 'Sean Bernstein' have a manager?
15. Find the manager of the manager of Wilson Adinolfi.

B.4 Attribute Constraint Queries (Complex Filtering)

16. List all employees in the 'Production' department who are located in 'MA'.
17. Show me employees recruited from 'LinkedIn' who have a performance rating of 'Exceeds'.
18. List all employees who have more than 10 absences.
19. Who are the employees with a project count greater than 5?
20. List employees with an engagement score higher than 4.5 who are Single.

Appendix C: Source Code Repository

The complete source code for the project, including the data cleaning pipeline, Neo4j ingestion scripts, and the LangChain RAG application, is available at the following GitHub repository:

- **Repository URL:** https://github.com/arkapravadatta/dissertaton-kg
- **Contents:**
  - `kg_pipeline.ipynb`
  - `baseline_RAG.ipynb`

Appendix D: System Dependencies

The project requires the following Python libraries (as listed in `requirements.txt`):

- `pandas` (Data Manipulation)
- `neo4j` (Graph Database Driver)
- `langchain` (LLM Framework)
- `langchain-groq` (Chat Model Integration)
- `python-dotenv` (Environment Variable Management)

---

9. References

1. Lewis, P. et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems (NeurIPS), Vol. 33, 2020, pp. 9459-9474.
2. Robinson, I., Webber, J. and Eifrem, E., "Graph Databases," 2nd edition, Sebastopol; O'Reilly Media, 2015, pp. 45-60.
3. Pan, S. et al., "Unifying Large Language Models and Knowledge Graphs: A Roadmap," IEEE Transactions on Knowledge and Data Engineering, Vol. 36, No. 1, 2024, pp. 1-19.
4. Needham, M. and Hodler, A.E., "Graph Algorithms: Practical Examples in Apache Spark and Neo4j," 1st edition, Sebastopol; O'Reilly Media, 2019, p. 112.
5. Barrasa, J. and Webber, J., "Knowledge Graphs: Data in Context for Responsive Enterprises," 1st edition, Boston; O'Reilly Media, 2021, p. 2

---

10. Glossary

Term	Meaning	Page No(s)
Cypher	A declarative graph query language for Neo4j used to retrieve data from the graph efficiently.	12, 24, 30
Vector Embedding	A numerical representation of text in a high-dimensional space where semantic similarity corresponds to geometric proximity.	5, 14, 41
Hallucination	A phenomenon where a generative model produces plausible but factually incorrect information not grounded in the provided context.	2, 8, 35
Knowledge Graph (KG)	A structured representation of data consisting of nodes (entities) and edges (relationships) that models domain knowledge explicitly.	1, 4, 18
LangChain	An open-source framework designed to simplify the creation of applications using large language models.	15, 22
Neo4j	A graph database management system that stores data relationships as first-class entities.	10, 19, 28
Retrieval-Augmented Generation (RAG)	A technique that enhances LLM output by referencing an authoritative external knowledge base before generating a response.	1, 3, 33
Schema-on-Write	A data management strategy where the structure of the data is strictly defined before it is stored, ensuring data integrity.	20, 38
Semantic Drift	The tendency of vector-based retrieval systems to return results that share keywords but differ in conceptual meaning.	32, 37
Zero-Shot Prompting	A method of querying a large language model where no prior examples are provided, relying solely on the model's pre-trained knowledge.	25, 29