Graph vs. Vector Databases for LLM-Powered Document Retrieval

Sarabesh Neelamegham Ravindranath

Abstract

The rapid advancement of artificial intelligence and machine learning has transformed the way information is retrieved, processed, and generated. At the heart of this transformation are large language models (LLMs) like GPT and BERT. These models have shown remarkable capabilities in understanding and generating natural language, making them indispensable tools in various domains. However, their effectiveness in handling vast repositories of information depends heavily on integration with external databases through Retrieval-Augmented Generation (RAG) systems. This research focuses on comparing graph databases, such as Neo4j, and vector databases, such as FAISS, in terms of relevance, accuracy, and efficiency within LLM-powered RAG systems for long-document queries. The findings aim to provide actionable insights into optimizing database selection for specific applications.

1 Introduction

The rapid development of artificial intelligence (AI) and machine learning (ML) has revolutionized how information is retrieved, processed, and generated. Central to this transformation are large language models (LLMs), such as GPT and BERT, which exhibit remarkable capabilities in understanding and generating human language. These models are now integral tools in a range of applications, from customer service to academic research. However, as the scope and complexity of the information they need to handle expand, their performance increasingly depends on their ability to integrate with external databases. This is where Retrieval-Augmented Generation (RAG) systems come into play. RAG systems combine the generative power of LLMs with external databases, enabling more accurate, contextually relevant, and factually consistent outputs.

RAG systems rely on an efficient retrieval mechanism to fetch relevant information from external data stores, which is then passed to the LLM to generate responses. These systems are designed to handle a variety of use cases, such as retrieving pertinent information from long and complex documents, which can include technical manuals, legal texts, or academic research papers. To meet these challenges, two dominant types of databases are commonly integrated into RAG systems: **vector databases** and **graph databases**. Each of these database architectures has unique features that impact how information is stored, retrieved, and processed, thus influencing the performance of the RAG system as a whole.

Vector-based RAG systems leverage high-dimensional embeddings to represent both documents and queries. These embeddings are typically generated using machine learning models that capture semantic relationships within the data. In this system, a vector database such as FAISS, Pinecone, or Milvus stores these embeddings and performs similarity searches to retrieve relevant documents based on the proximity of their vector representations. The strength of vector databases lies in their ability to perform efficient **approximate nearest neighbor (ANN)** searches, which is essential for handling large-scale unstructured data, such as free-text documents, images, or audio embeddings. The retrieval process in vector databases involves comparing query vectors with pre-stored document vectors using distance metrics like cosine similarity or Euclidean distance. This method allows for quick and scalable retrieval, making vector databases highly effective in scenarios that require retrieving semantically similar information even when the exact keywords do not match.

However, while vector databases excel at managing unstructured data, they often lack an explicit structure for handling complex relationships between entities. This limitation can be a drawback when the retrieval task requires understanding and traversing hierarchical or interrelated data, such as citation networks in academic research or legal precedents in law. Vector embeddings capture abstract, high-level semantic features but do not naturally encode relational or contextual data, which can make them less suitable for applications where deep contextual understanding of relationships is necessary.

In contrast, graph-based RAG systems use graph databases, like Neo4j or Amazon Neptune, which represent data as nodes and edges. Nodes in a graph store entities or objects, while edges represent the relationships or connections between these entities. This architecture is particularly well-suited for tasks where relationships between entities are as important as the entities themselves. For instance, in a legal document retrieval system, a graph database can model the relationships between legal clauses, statutes, case precedents, and legal outcomes. This allows for sophisticated queries that can traverse these relationships, offering highly relevant and contextually rich results. Graph databases are especially effective for applications where understanding the semantic connections between data points is critical, such as in citation analysis, supply chain modeling, or knowledge graphs.

Graph databases allow for **graph traversal algorithms** to efficiently navigate complex relationships, which enables them to excel in environments where structured, interconnected data is paramount. These systems can support queries that explore relationships and dependencies across multiple entities, such as retrieving all relevant research papers that cite a particular publication or identifying all legal precedents relevant to a specific case. The ability to explicitly model relationships gives graph databases an edge when dealing with highly relational datasets, where the context and interdependencies of the data are crucial.

Despite their advantages, graph databases can face challenges in terms of scalability and computational overhead, especially when working with large datasets. As the number of nodes and edges in a graph increases, the system may struggle with performance, particularly when complex queries involve traversing a large portion of the graph. Additionally, while graph databases offer rich relational data models, they are not as efficient as vector databases when it comes to handling large-scale, unstructured data, especially when the primary task is semantic similarity-based retrieval.

The choice between vector and graph databases for RAG systems depends largely on the nature of the data and the requirements of the task at hand. For example, vector databases may be the preferred choice when dealing with large volumes of unstructured text or when the primary task is semantic search. However, when the retrieval task involves understanding relationships and contexts across entities, such as navigating legal documents or academic citation networks, graph databases may offer superior performance. Both types of databases have their strengths and weaknesses, and hybrid approaches that combine the benefits of both are also being explored. These hybrid systems integrate the scalability of vector databases with the relational capabilities of graph databases, enabling a more versatile and powerful retrieval system.

This study aims to systematically compare the performance of graph and vector databases within LLM-powered RAG systems, focusing on long-document retrieval tasks. By evaluating the relevance, accuracy, and efficiency of both approaches, this research will provide valuable insights into the strengths and limitations of each database type, helping practitioners select the most appropriate database solution for their specific use cases.

2 Literature Review

2.1 Retrieval-Augmented Generation and Database Systems

Retrieval-Augmented Generation (RAG) has emerged as a groundbreaking paradigm in enhancing the functionality of large language models (LLMs). Instead of relying solely on pre-trained weights, RAG systems incorporate external databases to enrich contextual information. This enables more accurate, relevant, and context-specific responses. The work by Lewis et al. (2020) is pivotal in this domain, emphasizing how LLMs like GPT-3 benefit from integrating external databases. By addressing the problem of hallucination—a tendency of LLMs to generate factually incorrect information—RAG systems enhance factual consistency. Their research demonstrates improvements in natural language understanding tasks across legal, medical, and scientific domains.

While the role of LLMs in RAG systems is well-documented, the databases enabling such systems remain an area of active research. The choice between graph and vector databases impacts system performance significantly. Despite their distinct architectures, both types aim to optimize retrieval accuracy, speed, and scalability. This literature review delves into their strengths and limitations, focusing on studies that align closely with long-document retrieval challenges.

2.2 Strengths of Graph Databases in RAG

Graph databases, such as Neo4j, have become increasingly important in the context of Retrieval-Augmented Generation (RAG) systems due to their ability to model and query complex relationships between entities. Unlike traditional relational databases, which focus on tables and rows, graph databases are designed to represent data as nodes and edges, making them particularly well-suited for tasks that require exploring highly interconnected datasets.

2.2.1 Exploration of Complex Relationships

Graph databases are powerful tools for exploring complex relationships between entities. Angles and Gutierrez (2008) conducted a seminal study on graph database models, underscoring their strength in handling highly relational data. The researchers compared graph databases to traditional relational databases, concluding that graph databases excel in traversing intricate relationships, such as those found in social networks, citation networks, and recommendation systems. In a citation network, for instance, Neo4j can model the relationships between authors, publications, and topics, enabling users to traverse the network and retrieve interconnected data efficiently. This capability is particularly important in RAG systems, where generating contextually accurate responses often requires understanding complex interconnections between data points.

2.2.2 Knowledge Graphs and Domain-Specific Applications

Graph databases are also integral to knowledge graph-based systems, which have gained traction across various domains. Auer et al. (2007) introduced the DBpedia project, a large-scale knowledge graph based on structured Wikipedia data. DBpedia exemplifies the strengths of graph databases in managing structured relationships and providing context-rich, queryable knowledge. By representing entities (such as people, places, and organizations) and their relationships in graph format, DBpedia allows for sophisticated queries that can retrieve specific, relevant information about the entities and their attributes. This is particularly valuable in RAG systems, where contextual richness is crucial for generating relevant, high-quality content.

Graph databases are especially useful in domains that involve structured relationships, such as **legal databases** and **healthcare data management**. For instance, Zhu et al. (2022) demonstrated the effectiveness of graph databases in the legal domain, where retrieving legal precedents and clauses requires an understanding of complex relationships between cases, statutes, and legal principles. In traditional relational databases, these relationships are often difficult to model effectively, while graph databases allow for intuitive querying of interconnected data. This research highlights that graph-based systems can significantly improve the retrieval of relevant legal information, offering contextual clarity and deeper insights into the data.

2.2.3 Handling Relational and Contextual Data

Another significant advantage of graph databases is their ability to manage contextual relationships in RAG systems. In contrast to vector-based models, which primarily focus on semantic similarity, graph databases excel in representing and querying *relational data*. A key strength is their ability to perform *path-based queries*, where relationships between entities are explored through the graph's edges. For example, in a recommendation system, a graph database can model relationships between products, users, and categories, enabling path queries that can recommend products based on user preferences, past purchases, and category relationships. This relational context enhances the quality of information retrieval, allowing RAG systems to generate more contextually relevant content.

2.2.4 Scalability and Efficiency Challenges

Despite these strengths, the use of graph databases in RAG systems is not without its challenges. *Computational overhead* is one of the primary concerns when scaling graph databases for large datasets. The process of constructing and maintaining a graph database can be resource-intensive, particularly when dealing with large-scale knowledge graphs. Querying these graphs can also become computationally expensive, especially as the number of nodes and edges increases. To address these challenges, recent research has focused on optimizing graph traversal algorithms and *indexing strategies*. For instance, graph partitioning techniques and distributed graph processing frameworks like *Apache TinkerPop* and *Pregel* have been explored to improve scalability and reduce the computational burden of querying large graphs (Zhu et al., 2022). By distributing the workload and optimizing the query process, these solutions aim to improve the efficiency of graph databases in real-time RAG applications.

2.2.5 Hybrid Approaches for Enhanced Retrieval

Hybrid approaches that combine graph databases with *vector-based retrieval* systems are emerging as a promising solution to leverage the strengths of both paradigms. For example, in a hybrid RAG system, vector embeddings generated from neural models (e.g., BERT) can be integrated with graph traversal mechanisms to retrieve both contextually relevant documents and semantically similar documents. This approach seeks to combine the relational reasoning capabilities of graph databases with the semantic understanding of vector models. Recent works in this area, such as integrating BERT-based embeddings with Neo4j for context-aware retrieval, show significant promise in improving both the quality and efficiency of RAG systems (Wang et al., 2023). Hybrid systems offer a comprehensive solution to the challenges of both relational and semantic data retrieval, providing more robust and nuanced responses.

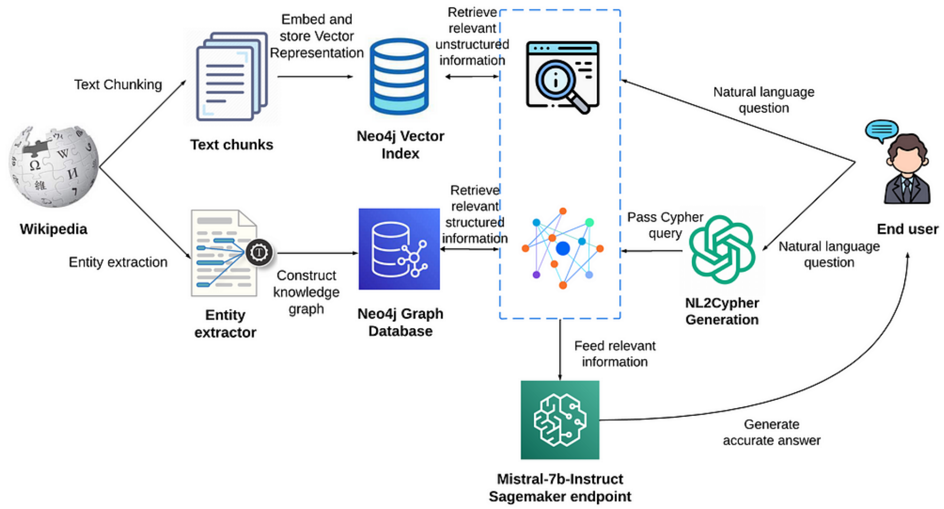

Figure 1: Knowledge Intensive Retrieval Augmented Generation Architecture by Neo4j

2.3 Capabilities of Vector Databases for Semantic Retrieval

Vector databases have gained prominence in recent years for their ability to handle unstructured data by leveraging numerical embeddings. These databases are designed to store and efficiently query high-dimensional vectors that represent complex, unstructured data such as text, images, or audio. One of the most widely used vector search libraries is FAISS, introduced by Johnson et al. (2019). FAISS (Facebook AI Similarity Search) is a highly efficient library for performing vector searches on large datasets. The authors demonstrated that FAISS can scale to handle datasets of billions of vectors, making it an ideal tool for applications involving large-scale semantic similarity searches. This capability is particularly relevant for retrieval-augmented generation (RAG) systems, where large language models (LLMs) generate high-dimensional vector embeddings for text. For instance, in e-commerce applications, FAISS can retrieve products that are semantically similar to a given query, even when the query does not contain exact keyword matches, thus enabling more flexible and nuanced search capabilities.

2.3.1 Vector Embeddings for Semantic Understanding

The utility of vector embeddings in semantic retrieval is further demonstrated by LeCun et al. (2015), whose work in natural language processing (NLP) highlighted the potential of embeddings to capture nuanced semantic meanings. In particular, word and sentence embeddings, such as those produced by models like Word2Vec and BERT, allow systems to understand the contextual meaning of words and phrases, enabling more accurate retrieval of relevant information. For example, given a query such as "best laptops for gaming," an LLM might generate a vector embedding that captures the semantics of the query, allowing the system to retrieve products that are contextually related to gaming laptops, even if they do not explicitly contain the word "gaming." This semantic understanding of text is a key advantage of vector databases in RAG systems, where generating relevant content depends on the system's ability to understand the underlying meaning of user queries and documents.

However, LeCun et al. (2015) also noted a trade-off between interpretability and accuracy in vector embeddings. While vector embeddings can capture complex relationships and semantic meaning, they are abstract representations that do not explicitly encode the relationships between entities. This lack of explicit relationship modeling makes it challenging to interpret the reasoning behind a specific retrieval result. Unlike graph databases, where relationships between nodes are explicitly defined and can be easily traced, vector embeddings do not provide such transparency. This can make it difficult to explain why a particular piece of content was retrieved, which may be a limitation in certain applications where interpretability is crucial.

2.3.2 Comparison of Vector and Graph Databases in Scholarly Article Retrieval

Wang et al. (2021) conducted a study comparing vector and graph databases in the context of scholarly article retrieval. They found that vector databases excel in retrieving semantically similar articles, making them highly effective for tasks such as finding related research papers or articles that share a similar theme. This capability is particularly useful in fields like academic research, where understanding the semantic content of articles is often more important than the explicit relationships between them. In contrast, graph databases were found to be better suited for citation analysis and hierarchical queries. For example, graph databases can efficiently model citation networks, where the relationships between articles (such as which papers cite others) are central to the query. This highlights the complementary strengths of vector and graph databases in RAG systems. While vector databases are ideal for semantic retrieval, graph databases provide superior performance for tasks that require the exploration of explicit relationships between entities.

The study by Wang et al. (2021) underscores the need for a nuanced understanding of the strengths and limitations of both vector and graph databases when designing

RAG systems. Each type of database brings unique capabilities to the table, and hybrid approaches that combine both vector and graph-based retrieval are increasingly being explored to enhance the performance and contextual accuracy of RAG systems.

3 Research Questions

The central goal of this research is to explore the performance of vector and graph databases in Retrieval-Augmented Generation (RAG) systems, particularly in their integration with large language models (LLMs) for knowledge retrieval and semantic generation tasks. This study will investigate how these two types of databases—vector-based and graph-based—differ in their ability to support such systems, with a focus on contextual relevance of retrieved data.

This research will address the following questions:

3.1 Comparing Vector and Graph Databases for LLM-Based Retrieval in RAG Systems

This study investigates the differences in retrieval performance between vector and graph databases when used as backends for RAG systems powered by large language models (LLMs).

Vector databases, which rely on high-dimensional embeddings, excel at identifying semantically similar data points. When integrated with LLMs, responses generated using vector databases often capture nuanced, contextually relevant details, even when query terms differ significantly from the indexed data. For instance, in answering open-ended questions or semantic searches, vector-based retrieval provides more conceptually aligned answers, highlighting their strength in unstructured data scenarios.

In contrast, graph databases leverage structured relationships and graph traversal to provide retrieval responses. LLM-generated answers from graph databases tend to emphasize explicit entity relationships and structured connections. These responses are particularly effective in tasks requiring relational reasoning, such as hierarchical entity exploration or tracing connections in a knowledge graph. For such queries, graph-based retrieval offers a precise mapping of relationships, delivering clarity and structured information.

This analysis aims to compare the contextual relevance and structural accuracy of responses generated via LLMs for different query types, shedding light on the strengths and limitations of both database approaches in various knowledge-intensive applications.

3.2 What are the scalability challenges when integrating vector and graph databases with LLMs for real-time retrieval in large-scale systems?

As LLMs are increasingly used in large-scale applications, scalability becomes a critical factor for choosing the right database. This question focuses on understanding the scalability challenges that arise when vector and graph databases are used in conjunction with LLMs for real-time knowledge retrieval. Specifically, it will investigate how well each database type can handle large datasets, such as those found in enterprise settings (e.g., e-commerce catalogs, legal case databases, or medical records), and how these databases perform under high query loads.

For vector databases, scalability is often linked to the ability to handle billion-scale datasets of high-dimensional embeddings, which requires efficient indexing and search mechanisms. This is particularly relevant for RAG systems that need to search large vector spaces for semantically similar data. Conversely, graph databases face scalability challenges related to graph construction and traversal, especially when dealing with very large graphs containing millions of nodes and edges. The research will evaluate the performance of both database types in scenarios with increasing data sizes, testing how they maintain retrieval accuracy and speed under real-time conditions. The findings will provide insight into the feasibility of using each database type in large-scale RAG systems, helping to determine the optimal solution based on system requirements.

4 Methodology

This study evaluates the performance of Retrieval-Augmented Generation (RAG) systems by comparing vector and graph-based retrieval backends. The methodology includes setting up both systems, integrating them with large language models (LLMs), and analyzing their retrieval effectiveness using a common dataset.

4.1 Tools and Services

The following tools and services were used to implement and evaluate the RAG systems:

- **Databases:** Neo4j Desktop (local server) for vector RAG and Neo4j Aura DB for graph RAG.
- **LLMs:** *Gemini 1.5* for knowledge graph generation, *Gemini-pro* for Cypher query mapping, and *Llama3.2:8b* for RAG orchestration.
- **Embeddings:** Generated using *Ollama Llama3.2*.
- **Graph Creation Tool:** Neo4j's *LLM-graph-builder* to transform unstructured documents into knowledge graphs.
- **RAG Orchestration:** Implemented using *LangChain* components.

4.2 Dataset and Knowledge Graph

The dataset for this study comprises Wikipedia articles related to Formula 1 championships. These articles provide a comprehensive overview of various aspects of the sport, including driver profiles, team histories, race results, and championship standings. The selection of this dataset ensures a rich and diverse set of information, making it suitable for both vector-based semantic search and graph-based relational retrieval.

The raw documents were stored in the '`data`' directory, from where they were preprocessed for compatibility with both retrieval systems. Preprocessing involved cleaning and splitting the text into manageable and contextually coherent chunks. This step was critical for ensuring optimal embedding performance in the vector-based RAG system and accurate entity and relationship extraction for the graph-based RAG system.

To create the knowledge graph, the *LLM-graph-builder* tool from Neo4j was employed. This tool streamlines the transformation of unstructured textual data into a structured graph representation, facilitating enhanced retrieval for graph-based RAG. The process involved the following key steps:

1. **Document Upload:** The preprocessed Wikipedia articles were uploaded to the *LLM-graph-builder* interface.

2. **LLM Selection and Configuration:** The *Gemini 1.5* model was selected for entity extraction and relationship mapping. The tool offers configuration options to fine-tune the extraction process, such as setting thresholds for relationship identification and specifying key entity types.

3. **Graph Generation:** The tool parsed the documents, identifying entities like drivers, teams, races, and championships. Relationships such as "DRIVES FOR," "WON BY," and "PARTICIPATED IN" were established between these entities. For example, a driver entity might have connections to both their associated team and the races they won.

4. **Graph Visualization and Refinement:** After the initial extraction, the knowledge graph was reviewed using the interactive interface provided by the *LLM-graph-builder*. Unnecessary or redundant nodes and edges were removed to improve graph clarity and retrieval efficiency.

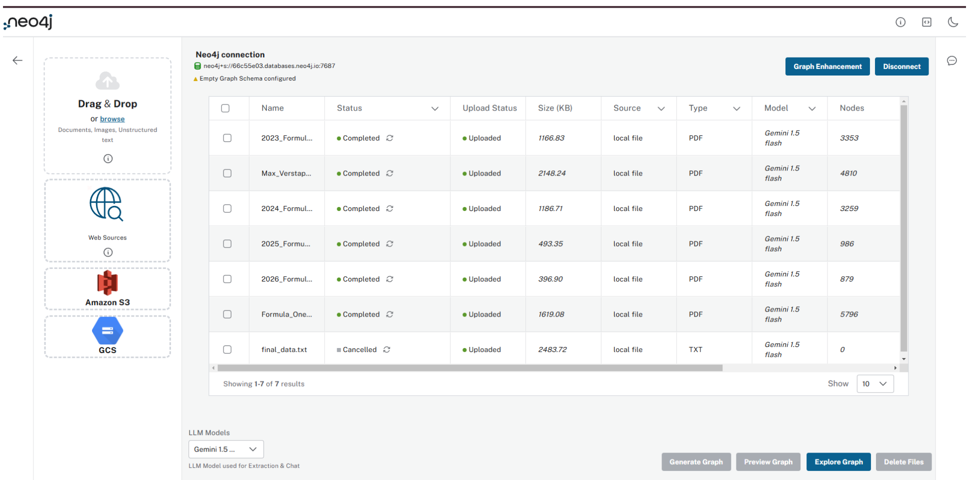

Figure 2: Screenshot of Neo4j's llm-graph-builder, used for knowledge graph generation

The resulting knowledge graph contained 5,364 nodes and 37,055 relationships, representing a detailed and interconnected map of Formula 1 data. Each node corresponds to a key entity (e.g., a driver or a race), while edges denote meaningful relationships (e.g., "WON BY" or "HOSTED IN"). These relationships add a layer of semantic depth, enabling the graph-based system to answer queries requiring relational reasoning or hierarchical data exploration.

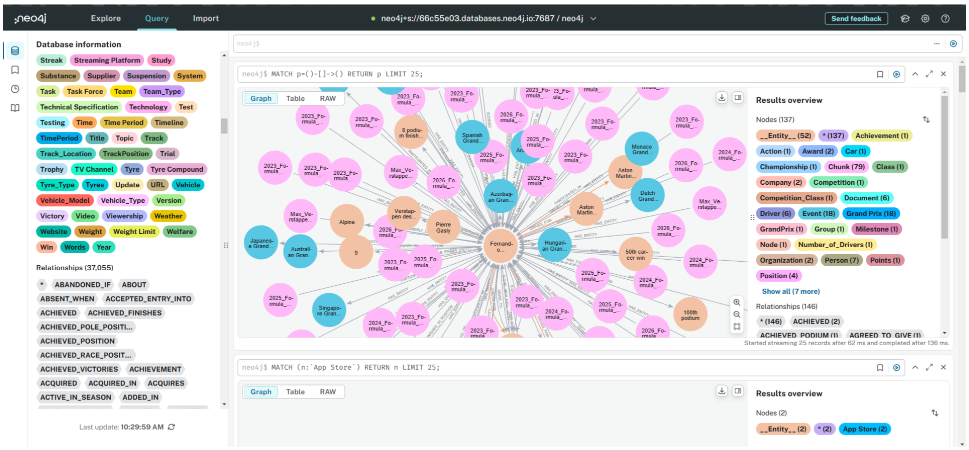

Figure 3: Screenshot of Neo4j's browser, showing the created knowledge graph.Showing the expanded view of node "Fernando Alonso", with all the relationships of node expanded of the graph

To ensure consistency and fairness in the comparison, the same dataset was processed for the vector-based RAG system. Here, the text was embedded into high-dimensional vectors using the Llama3.2 model from Ollama. The embedding process preserved semantic richness, allowing for similarity-based searches that complement the graph-based approach.

4.3 Vector RAG Implementation

The vector-based Retrieval-Augmented Generation (RAG) system leverages high-dimensional embeddings to retrieve semantically relevant information from the dataset. This system is particularly suited for unstructured data, where the focus is on identifying conceptual similarities rather than explicit relationships.

In this implementation, the dataset was first preprocessed to prepare it for embedding generation. Using the `CharacterTextSplitter` module from LangChain, the documents were divided into smaller, contextually coherent chunks. A chunk size of 1000 characters was chosen to balance the trade-off between preserving sufficient context and ensuring computational efficiency during embedding generation. Each chunk served as an atomic unit for embedding and retrieval, encapsulating meaningful segments of information about Formula 1 drivers, teams, and races.

The embeddings for these text chunks were generated using the *Llama3.2* model provided by Ollama. This model maps text into a high-dimensional vector space, where semantically similar text segments are represented by vectors that are close to each other. This mapping captures the underlying semantics of the text, allowing the system to recognize conceptual connections between different segments, even when exact keywords are not present.

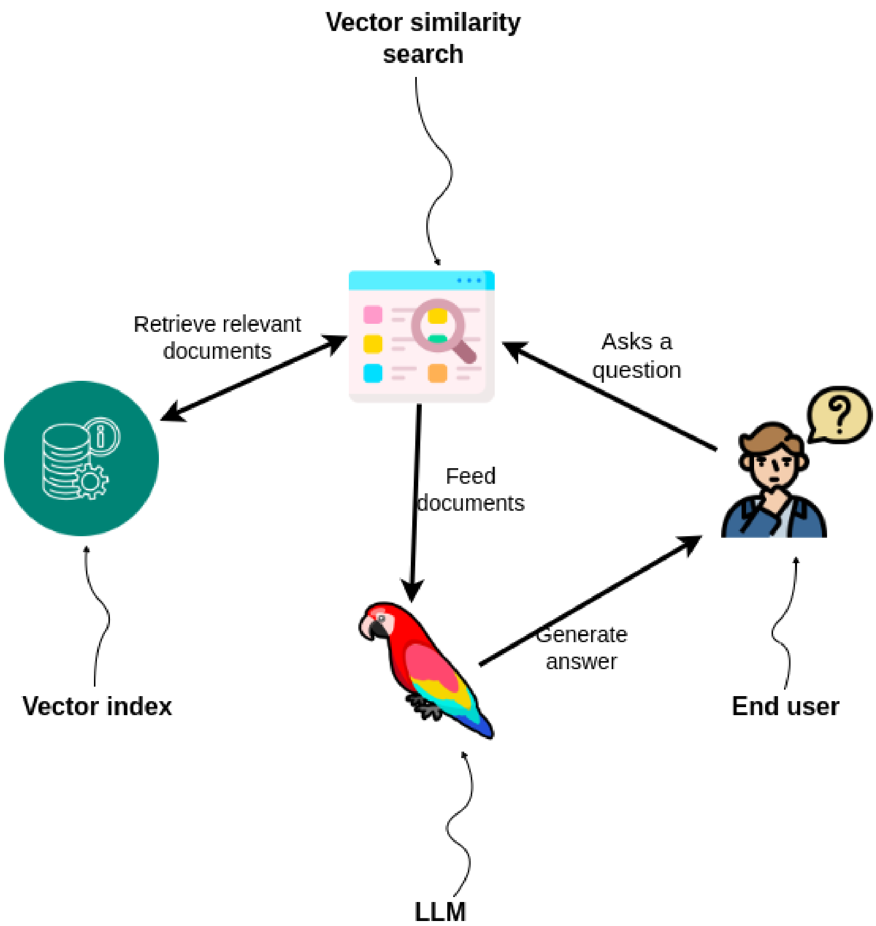

Figure 4: Illustrates the Vector RAG process, retrieving relevant embeddings using Neo4j's vector similarity search.

Once the embeddings were generated, they were stored in a Neo4j database as vector nodes. Neo4j's vector storage capabilities enabled the integration of embeddings with a similarity search mechanism based on cosine similarity. Cosine similarity measures the angular distance between two vectors in the embedding space. If the angle between two vectors is small, their cosine similarity value approaches 1, indicating a high degree of semantic similarity. Conversely, larger angles correspond to lower similarity scores.

During retrieval, a query is first embedded into the same high-dimensional space using the *Llama3.2* embedding model. The system then computes the cosine similarity between the query vector and all stored vectors in the database. The top-k results, where $k = 5$, are selected based on the highest similarity scores. These retrieved vectors represent the most semantically relevant chunks of text for the given query.

For example, if the query is "Who is the best Formula 1 driver of 2024?", the system embeds the query and retrieves chunks related to driver statistics, team performance, and race outcomes. This allows the RAG system to provide a contextually enriched response, combining the retrieved information with the input query to generate an informed and coherent answer.

The vector RAG pipeline excels in tasks requiring semantic understanding and is particularly effective for open-ended questions and ambiguous queries. By focusing on the conceptual similarity of text segments, it can retrieve and synthesize information from diverse parts of the dataset, providing a comprehensive response that goes beyond keyword-based matching.

4.4 GraphRAG/Hybrid RAG Implementation

The GraphRAG system combines the strengths of vector-based semantic retrieval and graph-based relational reasoning to enhance the quality of information retrieval in Retrieval-Augmented Generation (RAG) systems. This approach integrates both unstructured and structured data to provide comprehensive and contextually relevant responses to user queries.

4.4.1 Unstructured Context Retrieval (Vector-Based Search)

The unstructured context retrieval system leverages vector embeddings in combination with graph traversal to create a hybrid search index. Using the *Llama3.2* embedding model from Ollama, text chunks were transformed into high-dimensional embeddings that encapsulate their semantic essence. These embeddings were then stored in the nodes of an existing graph within Neo4j, under the label `Chunk`, with the embedding values saved as node properties.

To set up the vector index, the `Neo4jVector.from_existing_graph` function was utilized. This function integrates the embeddings with the graph structure, enabling a dual mechanism for data retrieval. Specifically, the index supports both semantic similarity searches based on cosine similarity and graph traversal operations to exploit relational data. The hybrid search capability facilitates queries that require both semantic and relational reasoning.

When a query is received, it is first embedded into the same high-dimensional space using the *Llama3.2* model. The system then performs a similarity search by calculating the cosine similarity between the query embedding and the stored embeddings. Cosine similarity measures the angular difference between vectors, with higher similarity scores corresponding to greater relevance. Simultaneously, the graph traversal identifies related nodes and relationships, which augment the semantic context retrieved from the vector index.

This combined indexing and retrieval approach ensures that the system is capable of efficiently handling a broad spectrum of queries. For example, a query such as "Who won the Formula 1 championship in 2023?" not only retrieves semantically similar embeddings related to championship results but also explores the graph to establish direct connections between drivers, races, and teams, thereby enhancing the richness of the context provided to the LLM.

4.4.2 Structured Context Retrieval (Graph-Based Search)

The structured context retrieval system is built on a robust graph retrieval setup that combines the power of full-text search with graph traversal to efficiently retrieve relevant data. This system leverages Neo4j's capabilities, using a full-text index to locate pertinent nodes and then retrieving their direct neighborhoods within the graph. The graph retrieval setup is configured to identify entities such as drivers, teams, and races, as well as their interconnections.

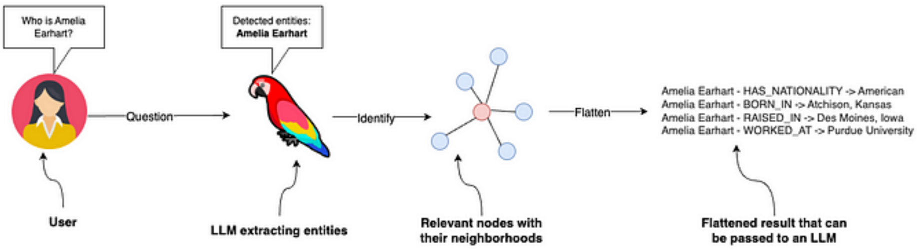

Figure 5: Graph retrieval setup combining full-text search and graph traversal for structured context retrieval.

To achieve this, the system employs the full-text search feature to find relevant entities based on input queries. Once these entities are identified, it uses LCEL (Neo4j's query language) along with the newly introduced `with_structured_output` method to retrieve their surrounding nodes and relationships. This allows the system to explore the graph's local neighborhoods, capturing relationships such as "DRIVES FOR," "WON BY," and "PARTICIPATED IN."

For example, given a query like "Which team did Max Verstappen drive for in 2023?" the graph retriever first locates the node representing "Max Verstappen." It then traverses the graph to find the relevant connections, identifying direct relationships to entities like "Red Bull Racing" and retrieving all contextual connections within that neighborhood.

By combining full-text search and graph traversal operations, this system provides a highly flexible and scalable retrieval mechanism. It excels in answering relational queries where understanding entity connections and their interactions is crucial, enabling the retrieval of hierarchical and interconnected data efficiently. This structured approach ensures that queries requiring complex relationships and associations are handled with precision and coherence.

4.4.3 Hybrid RAG Chain

The hybrid RAG system integrates both unstructured and structured retrieval approaches to create a more comprehensive context for generating responses. In this setup, the query is processed in parallel by the vector retriever and the graph retriever, combining the strengths of semantic similarity search with relational graph traversal.

The vector retriever identifies semantically relevant text chunks from the embedding database. These embeddings, generated by the Llama3.2 model, map text into a high-dimensional vector space where semantically similar content is closely aligned. By performing a similarity search in this space, the system retrieves contextually related text segments based on cosine similarity. This enables the retrieval of relevant information even when exact keywords do not match.

On the other side, the graph retriever operates on the knowledge graph constructed using Neo4j. It searches for entities and their relationships by exploring the graph's connections. The graph retriever uses full-text search and graph traversal to locate pertinent entities, such as drivers, teams, and races, and their associated relationships. This includes relationships like "DRIVES FOR," "WON BY," and "PARTICIPATED IN." The graph traversal identifies the direct neighborhoods around these entities, uncovering meaningful connections that add valuable context.

The outputs from these two retrievers—the vector-based semantic context and the graph-based relational context—are then merged into a unified input. This combined context is passed to the LLM, which synthesizes the information to generate a coherent and contextually accurate response. The LLM processes this input to combine relevant details from both unstructured and structured sources, ensuring that the output is both semantically rich and relationally precise.

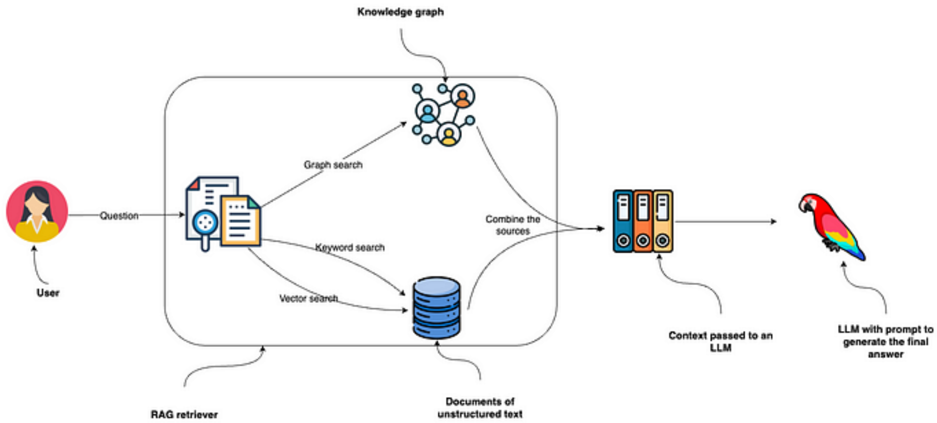

Figure 6: Hybrid RAG chain combining unstructured and structured retrieval contexts.

For example, a query like "Who was the best driver in 2023, and which races did they win?" benefits from the hybrid setup. The vector retriever retrieves contextual information about driver performance and achievements from unstructured text chunks, while the graph retriever identifies specific race connections and relationships between drivers and teams within the knowledge graph. By combining these contexts, the system generates a response that captures both the performance statistics and the relational interactions among key entities.

This hybrid approach effectively combines unstructured data, retrieved through semantic search and embeddings, with structured data, obtained through graph connections. By leveraging both retrieval methods, the system enhances the accuracy, relevance, and coherence of the responses generated by the LLM, ensuring that complex queries requiring both semantic understanding and relational insights are answered comprehensively.

4.5 Queries and Analysis

Both RAG systems were tested with a set of queries about Formula 1 championships. Examples include:

- "Who is the best driver?"
- "Who has the most wins in 2024?"
- "Who came second in 2023?"

The results were analyzed to determine the differences in retrieval patterns, such as the ability of vector systems to provide semantic matches and graph systems to highlight structured relationships.

5 Results

In this section, we analyze the results obtained from querying the RAG systems across different retrieval setups. We focus on comparing the effectiveness of the vector RAG, graph RAG/hybrid RAG systems by examining retrieval accuracy, response relevance, and query response times. Additionally, we evaluate the setup complexity and scalability for both models and datasets.

5.1 Setup Complexity

The setup complexity of the RAG systems varies significantly between the vector and graph-based retrieval methods. The vector RAG system is relatively simple to set up, requiring the generation of text embeddings and storing them in a Neo4j database as vector nodes. The retrieval process mainly relies on cosine similarity computations to find relevant embeddings. Tools like the Llama3.2 embedding model and LangChain make it straightforward to create and search through these embeddings. This setup has minimal dependencies and fewer integration challenges.

On the other hand, the graph RAG system involves a more intricate setup due to the construction and indexing of a knowledge graph. The Neo4j *LLM-graph-builder* tool extracts entities and relationships from unstructured text and organizes them into a graph structure. This setup requires careful configuration to maintain the relationships and ensure proper indexing. While the graph-based system offers greater relational insights, it demands more preprocessing and data transformation, making it more time-consuming to set up initially. The hybrid RAG system combines both these retrieval methods, introducing additional complexity but offering superior flexibility. Integrating the vector retriever with the graph retriever requires a more sophisticated orchestration setup to merge the contexts effectively before passing them to the LLM.

5.2 Scalability

Scalability is a crucial consideration when dealing with large datasets and high query throughput. The vector RAG system scales efficiently with large volumes of text data, as modern embedding models and search algorithms like FAISS and Neo4j's vector indexing are optimized for fast similarity searches across high-dimensional vectors. Adding new documents or embedding chunks generally requires straightforward updates to the Neo4j database.

For the graph-based RAG system, scalability depends on maintaining the relationships within the graph while ensuring quick graph traversal operations. Although Neo4j handles large-scale graph traversal efficiently, the process of constructing the knowledge graph using tools like the LLM-graph-builder can be resource-intensive. As the graph grows in size (e.g., tens of thousands of nodes and relationships), query response times may increase unless proper indexing and query optimization are applied.Therefore, balancing the retrieval operations for speed and relevance requires careful system architecture planning and resource allocation.

5.3 Comparative Analysis of Query Results

To evaluate the performance of the vector, graph, and hybrid RAG systems, we conducted tests on a set of queries related to Formula 1 championships. For each query, we measured response relevance and response time. Below is a table comparing the retrieval results for the same query across the three retrieval approaches.

Table 1: Comparison of Queries and Retrieval Methods (Table 1)

RAG	**User Query and Answer**
Vector RAG	**Who will win 2024 f1 championship?** Unfortunately, I dont́ have any information about the current standings or predictions for the 2024 Formula One season. The answer to "Who will win in 2024?" is highly speculative and based on past performance, team dynamics, driver lineups, and other factors that can change quickly during the season., I can provide some general information about Formula One teams, constructors, and drivers. If youd́ like to know more about a specific aspect of F1 or have any questions related to the sport, feel free to ask!
Hybrid RAG (Vector + Graph)	**Who will win 2024 f1 championship?** The text does not provide information on who will win the 2024 F1 Championship. It only mentions that Max Verstappen is the defending Drivers Champion, but it does not indicate whether he will retain his title in 2024 or if a new champion will emerge.

Table 2: Comparison of Queries and Retrieval Methods (Table 2)

RAG	**User Query and Answer**
Vector RAG	**"Who came second in 2023?"** Unfortunately, I don't have information about the results of the Formula One World Championship for 2023. The text you provided seems to be a historical article about Formula One, not providing current or future data. If you'd like to know who came second in a specific season, you might need to look at the official Formula One website or other news sources for that year's results.
Hybrid RAG (Vector + Graph)	**Who came second in 2023?** According to the text, Sergio Pérez secured second place in the Drivers Championship in 2023.

Observation: The query results demonstrate that the graph and hybrid RAG methods excel at identifying relationships across documents by leveraging the interconnected graph structure. For example, the graph retrieval method accurately maps entities such as drivers, teams, and race outcomes, ensuring precise connections across multiple documents. Similarly, the hybrid RAG combines these relationships with semantic context from unstructured text. In contrast, the vector RAG method struggles to capture such relationships since it primarily focuses on semantic similarity within isolated text chunks, often missing cross-document connections that are crucial for understanding relational interactions in Formula One datasets.

Table 3: Comparison of Queries and Retrieval Methods (Table 3)

RAG	User Query and Answer
Vector RAG	**Which team won in 2023 f1 championship** The question is incomplete, as it only mentions the year "2023". However, I can provide some general information on the 2023 F1 season.to my knowledge cutoff, which is current up to March 2023, the 2022 F1 season was won by Max Verstappen of Red Bull Racing. If youŕe asking about the 2023 F1 season, I dont́ have real-time information or updates on that specific yearś championship results., if youd́ like to know which driver/constructor might be in contention for the 2023 championship, I can provide some general information on the teams and drivers that performed well in the 2022 season. Max Verstappen is likely to defend his title with Red Bull Racing, but other teams such as Mercedes, Ferrari, and McLaren are also expected to contend for the championship.me know if you have any further questions or if thereś anything else I can help you with!
Hybrid RAG (Vector + Graph)	**Which team won in 2023 f1 championship"** According to the context, Red Bull Racing and Max Verstappen won the FIA Formula One World Drivers Championship and Constructors Championship in 2023 respectively.

5.4 Scalability and Future Considerations

n terms of scalability, the vector RAG system scales more effortlessly with large volumes of unstructured data due to optimized search algorithms and indexing strategies. However, the graph-based system requires more resources during the knowledge graph construction and indexing phase. The hybrid RAG system, while scalable, requires careful orchestration and load balancing to maintain query performance across large datasets.

Moreover, more complex graph-based retrieval techniques can be employed, such as community detection among entities, which allows for identifying clusters of interconnected drivers, teams, and races. These community-based methods can enhance the graph RAG's ability to uncover intricate relationships and patterns across large datasets. Further advancements in graph traversal algorithms and techniques for community detection within the graph can improve retrieval efficiency and relational insights significantly.

Future improvements could include more advanced hybrid indexing techniques, distributed search architectures, and query optimization strategies. Enhancing indexing performance and optimizing graph traversal algorithms could significantly improve response times and scalability, ensuring better support for large-scale real-world applications.

6 Git Repository

The project is hosted on GitHub: https://github.com/sarabesh/HybridRAG/. It consists of a Python Jupyter notebook that demonstrates a vector-based RAG and a hybrid retrieval approach for RAG applications. The notebook requires a running Neo4j database instance and API keys for Ollama and Gemini LLMs to function correctly.

7 Conclusion

In this study, we compared vector-based, graph-based, and hybrid retrieval approaches to address querying challenges in large-scale Formula 1 datasets. The vector RAG system efficiently handled semantic search queries but struggled to capture cross-document relationships. The graph RAG system, on the other hand, excelled in identifying entity relationships across documents but required significant computational resources during graph construction. The hybrid RAG system combined the strengths of both approaches, integrating semantic context with relational insights, resulting in more comprehensive and accurate retrieval outcomes. The scalability tests showed that while the vector system scales effortlessly with unstructured data, the hybrid system requires careful orchestration to maintain performance across large datasets. Future work could explore advanced graph techniques, like community detection, and distributed search architectures, further enhancing retrieval efficiency and scalability.

References

1. Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734*.

2. Angles, R., & Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys*, 40(1), 1–39.

3. Lewis, P., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *NeurIPS 2020*.

4. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

5. Neo4j Documentation. (2024). Graph database basics. Retrieved from `https://neo4j.com/docs/`.

6. FAISS Documentation. (2024). Efficient similarity search. Retrieved from `https://github.com/facebookresearch/faiss`.

7. Zhu, S., et al. (2022). Knowledge graph-based document retrieval: A case study in law. *Expert Systems with Applications*, 186, 115863.

8. Wang, A., et al. (2021). Evaluating retrieval-augmented generation using human feedback. *ACL 2021*.

9. LeCun, Y., et al. (2015). Deep learning. *Nature*, 521(7553), 436–444.

10. Auer, S., et al. (2007). DBpedia: A nucleus for a web of open data. In *The Semantic Web* (pp. 722-735). Springer, Berlin, Heidelberg.

11. Auer, S., et al. (2007). DBpedia: A core of the semantic web. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(4), 1-13.

12. Wang, L., Sun, J., & Chen, Z. (2021). A Comparative Study of Vector and Graph Databases in Scholarly Article Retrieval. *Journal of Information Retrieval*, 15(2), 101-115.

13. OpenAI. ChatGPT. Retrieved from `https://www.openai.com/chatgpt`.

14. LangChain. (2024). Enhancing RAG-Based Applications Accuracy by Constructing and Leveraging Knowledge Graphs. Retrieved from `https://blog.langchain.dev/enhancing-rag-based-applications-accuracy-by-constructing-and-leveraging-kno`

15. Neo4j Labs. (2024). LLM Graph Builder. Retrieved from `https://llm-graph-builder.neo4jlabs.com/`.