Adaptive Biomedical Knowledge Graph Querying: Orchestrating Multi-Agent AI for Complex Data Retrieval

A Project Report

submitted by

RAHUL KHARKWAL

MT23249

in partial fulfilment of the requirements

for the award of the degree of

MASTER OF TECHNOLOGY

Department of Computational Biology

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

NEW DELHI- 110020

August 20th, 2025

Certificate

This is to certify that the thesis titled **"Adaptive Biomedical Knowledge Graph Querying: Orchestrating Multi-Agent AI for Complex Data Retrieval"** being submitted by **Rahul Kharkwal** to the Indraprastha Institute of Information Technology, Delhi, for the award of the Master of Technology, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Dr Debarka Sengupta

Department of Computational Biology

Indraprastha Institute of Information Technology Delhi

New Delhi 110020

Place: New Delhi

Date: 20th August 2025

Acknowledgements

I sincerely want to thank everyone who stood by me throughout the process of completing my thesis. First and foremost, I'm deeply grateful to my advisor, **Dr. Debarka Sengupta**, for his steady guidance and constant encouragement. His insights and expertise helped shape my work and pushed me to maintain high standards throughout. Working under his mentorship has been an enriching experience.

I'd also like to express my heartfelt thanks to **Swarnava Samanta**, my senior and mentor, for being so generous with his time and knowledge. His technical guidance and support were especially valuable during the tough phases of this project.

A special thanks goes to my friend**, Kartikay Joshi**, for always being a source of motivation and support when I needed it most. Lastly, I'm thankful to everyone else who contributed in any way—whether by offering suggestions, sharing resources, or simply encouraging me along the way. Your support has truly meant a lot, and I'm grateful to have had such a strong network during this journey.

Rahul Kharkwal
MT23249

Abstract

This thesis explores the development and evaluation of a smart, multi-agent system designed to retrieve information efficiently from a diverse biomedical knowledge graph. The graph was carefully built using BioKG data in TSV format and structured in Neo4j, with special attention given to properly representing multi-valued properties as lists.

Early attempts to use embedding-based models—such as Nomic Atlas v1, BioBERT, and PubMedBERT—for semantic search presented several obstacles. The main issues stemmed from the highly varied nature of the data, repeated terms that introduced bias, and the models' limited ability to process structured key-value information effectively. Due to these limitations, a BM25 retriever was initially used for keyword-based node extraction. While it served as a practical starting point, its dependency on exact keyword matches proved restrictive.

To address these shortcomings and enhance retrieval accuracy, a layered multi-agent system was built using the LangChain supervisor agent framework, with GPT-4o Mini at its core. This system includes several specialized agents: one for handling query expansion and rewriting (including web search via Tavily), another for initial node retrieval using BM25, and a graph traversal agent that navigates the graph using Cypher queries and generates comprehensive responses.

Together, these components form a robust solution for querying complex biomedical datasets. The system not only improves over basic retrieval methods but also illustrates the potential of agent-based architectures in exploring large, heterogeneous knowledge sources.

Contents

Chapter 1: Introduction

1.1 Background

Over the past two decades, scientific disciplines have been flooded with data of ever–growing size and complexity. Nowhere is this more apparent than in biomedicine, where breakthroughs, clinical records, and high-throughput experiments churn out information on genomics, proteomics, metabolomics, drug interactions, disease mechanisms, and more. Faced with this torrent of heterogeneous data, researchers must tackle two main challenges: how to organize such diverse information and how to draw useful insights from it.

Traditional relational databases excel at handling uniform, tabular data—but they struggle when you need to capture the rich, evolving relationships that underpin biological systems. In response, knowledge graphs have emerged as a more flexible alternative. By treating entities (genes, proteins, diseases, drugs) as nodes and their various connections ("drug–drug interactions," "protein–protein interactions," etc.) as edges, knowledge graphs make complex networks explicit and queryable. This approach not only reveals direct links but also uncovers hidden pathways, enabling scientists to infer new connections and accelerate discoveries.

Just representing data, however, is only half the battle. Effective information retrieval (IR) techniques are equally vital for navigating these large graphs. For years, simple keyword search served us well, but the increasing intricacy of biomedical data demands more nuanced methods. Embedding models—turning words and phrases into numerical vectors—promise "semantic" understanding, where a system can grasp the intent behind a query rather than just spotting exact terms. More recently, advances in large language models and multi-agent architectures have opened the door to dynamic retrieval systems that can interpret complex queries, break them down into subtasks, and even engage in multi-step reasoning across heterogeneous data sources.

Together, knowledge graphs and modern IR techniques are reshaping how we integrate diverse biomedical datasets, build detailed disease models, speed up drug discovery, and move toward truly personalized medicine.

1.2 Problem Statement

Despite the recognized utility of knowledge graphs in biomedical informatics, their practical implementation and effective querying pose significant challenges that this thesis aims to address.

Firstly, the process of **knowledge graph construction** from raw, often semi-structured, biomedical data sources is inherently complex. The BioKG dataset, utilized in this research, epitomizes this challenge. Sourced in Tab Separated Value (TSV) format, its conversion into a Neo4j graph database required meticulous parsing and schema mapping. A particularly nuanced aspect encountered was the representation of **multi-valued properties for a single key** – a common occurrence in biomedical data where, for instance, a disease might have numerous associated symptoms. Storing such data faithfully within the Neo4j property graph model, specifically by representing these multiple values as lists, was a foundational technical challenge addressed in the graph construction phase.

Secondly, and perhaps the most critical hurdle, lay in developing an **efficient and intelligent information retrieval system** capable of querying this complex and heterogeneous biomedical knowledge graph. Our initial investigations explored the direct application of state-of-the-art **embedding models** (including the general-purpose Nomic Atlas Embedding Model v1 and

domain-specific BioBERT and PubMedBERT) for semantic search. The hypothesis was that these models could convert textual node and relationship properties into dense vector spaces, enabling flexible, meaning-based retrieval. However, these attempts proved to be "not beneficial" due to a confluence of factors:

- **High Data Heterogeneity:** The BioKG data is exceptionally diverse, encompassing a wide spectrum of entity types (Drugs, Genetic Disorder, Protein, Pathways, and Disease) and a broad range of textual descriptions. This inherent heterogeneity hindered the ability of embedding models to learn a unified and discriminative semantic space, leading to ambiguous or inaccurate vector representations.
- **Bias from Frequently Repetitive Terms:** Biomedical literature and databases often feature highly specialized and frequently repeated terminology, specifically the key property (e.g., "Name," "Class," "ID"). While crucial to the domain, the prevalence of these terms introduced a significant bias in the embeddings, causing many distinct entities to appear semantically similar simply because they shared common, yet non-discriminative, vocabulary. This reduced the precision of semantic matching.
- **Architectural Mismatch with Key-Value Pair Structures:** A fundamental limitation was that these embedding models are predominantly optimized for sequential textual data (sentences, paragraphs, documents). They are not inherently designed to understand or effectively encode the structured nature of **key-value pairs**, especially when values are aggregated as **lists** within a property graph model. Directly embedding such structured properties often led to a loss of the precise semantic relationship between the property key and its list of values, diminishing retrieval accuracy. Ad-hoc preprocessing attempts, such as removing numbers or highly frequent words, paradoxically "disturbed the data" by eliminating crucial contextual information or identifiers, thereby further degrading embedding quality.

Consequently, the project shifted its focus to the **BM25 retriever** as a more robust keyword-based mechanism for initial information extraction. While BM25 proved effective for direct keyword matches, this approach introduced its own "first challenge": its inherent "limitation to use specific words in our query to get good results". This meant that natural language queries requiring semantic understanding or multi-hop reasoning were poorly served, highlighting the need for an intelligent layer that could adapt queries and orchestrate multi-step retrieval.

This research, therefore, addresses the pressing need for a sophisticated and adaptable information retrieval system that can efficiently query complex and heterogeneous biomedical knowledge graphs, overcoming the limitations of both direct embedding applications and constrained keyword-based methods.

1.3. Objectives

This thesis aims to address the aforementioned challenges by achieving the following specific objectives:

1. **To construct a robust and semantically rich biomedical knowledge graph** from BioKG data in TSV format using Python and Neo4j, with particular attention to accurately representing and storing multi-valued properties as lists.
2. **To systematically evaluate the applicability and limitations of various embedding models** (Nomic Atlas Embedding Model v1, BioBERT, PubMedBERT) for generating meaningful vector representations of heterogeneous biomedical knowledge graph data,

explicitly documenting the observed challenges related to data heterogeneity, bias, and structural mismatch.

3. **To design and implement a sophisticated multi-agent information retrieval system** utilizing large language models (GPT-4o Mini) and the LangChain supervisor agent framework, specifically tailored for querying the constructed biomedical knowledge graph and orchestrating complex retrieval workflows.
4. **To integrate and orchestrate diverse information retrieval tools** within the multi-agent architecture, including a BM25 retriever for initial node extraction, a web search tool (Tavily Search) for real-time information augmentation, and Cypher query generation via function calling for adaptive graph traversal.
5. **To demonstrate the efficacy and advantages of the developed multi-agent system** in performing complex information retrieval tasks, encompassing intelligent query expansion, rewriting, and adaptive graph traversal, thereby offering a more flexible, comprehensive, and accurate querying experience compared to standalone methods.

Chapter 2: Literature Review

This chapter surveys the key ideas and prior work that underpin our intelligent retrieval system for biomedical knowledge graphs. We begin by tracing how knowledge representation has evolved, then explore advances in information retrieval methods, examine emerging graph-based Retrieval-Augmented Generation (RAG) approaches, and discuss the rise of multi-agent architectures driven by large language models. By highlighting gaps in today's solutions, this review sets the stage for the novel contributions of this thesis in AI and biomedical informatics.

2.1 Knowledge Graphs and Biomedical Data Modeling

Knowledge graphs (KGs) have gained prominence as a way to organize complex, interlinked information. Stemming from the semantic-web movement, they treat real-world entities—genes, proteins, diseases, drugs—as nodes, and the relationships between them as edges. This graph structure naturally captures rich semantics and supports powerful, flexible queries.

In biomedicine, KGs help bring together data from many sources—scientific articles (e.g., PubMed), protein databases (UniProt), ontologies (Gene Ontology, Disease Ontology), and clinical records—to paint a unified picture of biological systems. However, integrating such diverse datasets raises challenges: data formats and terminologies vary, new discoveries continually update our understanding, and some properties (for instance, a gene linked to several diseases) require multi-valued representations. Graph databases like Neo4j address this by allowing list-style properties for storing multiple values cleanly.

2.2 Information Retrieval Techniques

Information retrieval (IR) focuses on finding the most relevant information from large data collections in response to user queries. Over the years, IR has matured from straightforward keyword matching to more nuanced, meaning-oriented methods.

2.2.1 Traditional Keyword-Based Retrieval

Early IR systems depend on matching query keywords against documents and ranking results using statistical scoring. A classic example is Okapi BM25, which refines TF-IDF by adding term-frequency saturation (so that additional repeats of a word have diminishing returns) and normalizing for document length. BM25 remains a go-to method for many search applications because of its strong performance and simplicity.

Yet BM25's strength—relying on exact or stemmed word matches—is also its weakness. It struggles to grasp the underlying intent of a user's question when synonyms are used or when the phrasing is more conversational. In specialized domains like biomedicine, where terminology is varied and complex, purely keyword-based approaches can miss relevant connections that lie just beyond exact term overlap.

2.2.2. Embedding-Based Semantic Retrieval

The advent of deep learning has revolutionized IR through the development of **embedding models**. These models, typically neural networks, learn to represent words, phrases, or entire documents as dense numerical vectors (embeddings) in a high-dimensional space, where semantic similarity is approximated by vector proximity. General-purpose models like those supporting Nomic Atlas Embedding Model v1 learn broad semantic relationships from diverse text corpora. For specialized domains like biomedicine, pre-trained models such as **BioBERT** and **PubMedBERT** have emerged, fine-tuned on vast collections of biomedical literature (e.g., PubMed abstracts), aiming to capture domain-specific semantic nuances. These models promise to enable semantic search, allowing systems to retrieve information based on meaning rather than just keywords. However, their effectiveness can be hampered by highly heterogeneous data, biases from frequently repetitive terms, and inherent difficulties in representing structured data like key-value pairs, especially when values are lists, as demonstrated in our initial exploration.

2.3. Graph-Based Retrieval Augmented Generation (RAG) Systems

The limitations of standalone LLMs, such as hallucination and outdated knowledge, have led to the development of **Retrieval Augmented Generation (RAG)**[2]. RAG systems augment LLMs by retrieving relevant information from external knowledge bases before generating a response, thereby grounding the LLM's output in factual data. A powerful extension of this paradigm is **Graph-based RAG**, which leverages knowledge graphs as the external knowledge base.

Graph-based RAG offers several advantages:

- **Structured Context:** KGs provide highly structured and interconnected context, enabling LLMs to understand complex relationships.
- **Multi-hop Reasoning:** KGs naturally support multi-hop reasoning, allowing the retrieval of information that spans multiple entities and relationships.
- **Reduced Hallucination:** By grounding responses in the KG, the LLM is less prone to generating inaccurate or fabricated information.

Several approaches to Graph-based RAG have been proposed. Some pipelines focus on transforming unstructured documents into graph structures for enhanced retrieval, as exemplified by the original **GraphRAG pipeline** [3].Others outline methods to develop GraphRAG systems from scratch, emphasizing the use of knowledge graphs alongside vector databases for comprehensive retrieval [4]. Frameworks like LangChain also offer integrations and conceptual patterns for GraphRAG, providing tools and abstractions to build such systems [5]. While this thesis draws inspiration and conceptual understanding from these GraphRAG paradigms, its core multi-agent architecture is designed independently, focusing on orchestrating existing tools rather than building a novel graph-construction-from-text pipeline. The use of knowledge graph data from BioKG, distinct retrieval methods, and a unique agent orchestration distinguishes this work.

2.4 Multi-Agent Systems Powered by Large Language Models

Recent breakthroughs in large language models (LLMs), such as GPT-4o Mini, have paved the way for AI solutions capable of sophisticated reasoning, planning, and tool integration. A particularly promising approach is to organize these capabilities into multi-agent architectures, where distinct “agents” collaborate under the coordination of a supervisor. Each agent is assigned a focused role—whether it’s handling query interpretation, executing database calls, or performing external

web searches—and the supervisor orchestrates their interactions, passing information and control as needed.

Frameworks like the LangGraph Supervisor make it straightforward to define these hierarchical workflows. You register each agent with its specialized skill set, specify how they exchange messages, and lay out the steps they follow to solve a complex task. This modular design boosts resilience (agents can fail or retry independently), flexibility (new agents can be plugged in), and overall system capability—allowing the ensemble to tackle multi-step problems that a single model or monolithic tool could not manage alone.

Typical components in such setups include:

- **Web Search Agents**
  These agents use tools like Tavily Search to pull in up-to-date information from the internet, enriching the internal knowledge base with recent findings or real-world data.
- **Database Query Agents**
  Charged with speaking directly to structured repositories, these agents generate and execute queries—such as Cypher commands against a Neo4j instance—to retrieve precisely the nodes and relationships relevant to the user's request.
- **Query Processing Agents**
  Focused on refining the user's input, these agents can expand or rewrite queries to improve recall, manage synonyms, or translate conversational language into exact terms for downstream tools.

By combining these specialized agents under a unifying supervisor, multi-agent systems harness the deep language understanding of LLMs alongside targeted retrieval engines like BM25 and real-time web searches. In this thesis, we adopt the LangGraph Supervisor framework to assemble a layered retrieval pipeline, overcoming the constraints of isolated methods and enabling a more adaptable, end-to-end solution for querying large, heterogeneous biomedical knowledge graphs.

Chapter 3: Methodology

In this chapter, I outline the step-by-step process used to build our biomedical knowledge graph and to design the intelligent, multi-agent retrieval system that runs on top of it. We begin by describing the data sources and how they were transformed into a graph format. Next, we discuss our first experiments with simple retrieval methods. Finally, we dive into the full architecture of our multi-agent framework, detailing each component and how they work together.

3.1 Building the Knowledge Graph

The first task was to turn raw biomedical data into a rich, navigable graph in Neo4j. This involved:

Gathering and cleaning data
We started with the BioKG dataset, which comes as a series of TSV files. Each row either lists an entity (for example, a gene or a drug) along with its properties, or describes a relationship between two entities, complete with metadata.

Parsing and normalization
Custom Python scripts read these TSV files, handled missing or malformed entries, and standardized names and identifiers. We also detected when an attribute had multiple values—say, a protein tied to several pathways—and prepared those as list properties in the graph.

Populating Neo4j
Once the data was cleansed and structured, we used the Neo4j Python driver to batch-create nodes for each entity and relationships (edges) for each triplet. Special care was taken to ensure multi-valued properties (like "associated_diseases" or "known_interactions") were stored as lists, preserving the full semantic richness of the original data.

3.1.1 Data Source and Preprocessing: BioKG

The BioKG collection served as our foundation. Here's how we handled it:

- **TSV ingestion:** We treated each file as a table—reading it line by line to separate entities from relationship records.
- **Entity extraction:** Whenever a row described a standalone entity (e.g., a drug with its name, ID, and other tags), we mapped its columns to node labels and properties.
- **Relationship extraction:** For rows that formed triplets (entity A → relationship → entity B), we extracted the source and target IDs plus any attached metadata (such as evidence scores or publication references).
- **Data cleaning:** We applied filters to remove duplicates, corrected inconsistent naming, and harmonized identifiers against external references (e.g., UniProt IDs for proteins).
- **Handling lists:** Whenever a column contained multiple comma- or semicolon-separated values, we split them into Python lists so they could be stored as multi-valued properties in Neo4j.

With this preprocessing pipeline in place, our Neo4j database was populated with a semantically enriched graph ready for querying—laying the groundwork for the next phase: experimenting with retrieval methods and, ultimately, building the multi-agent system.

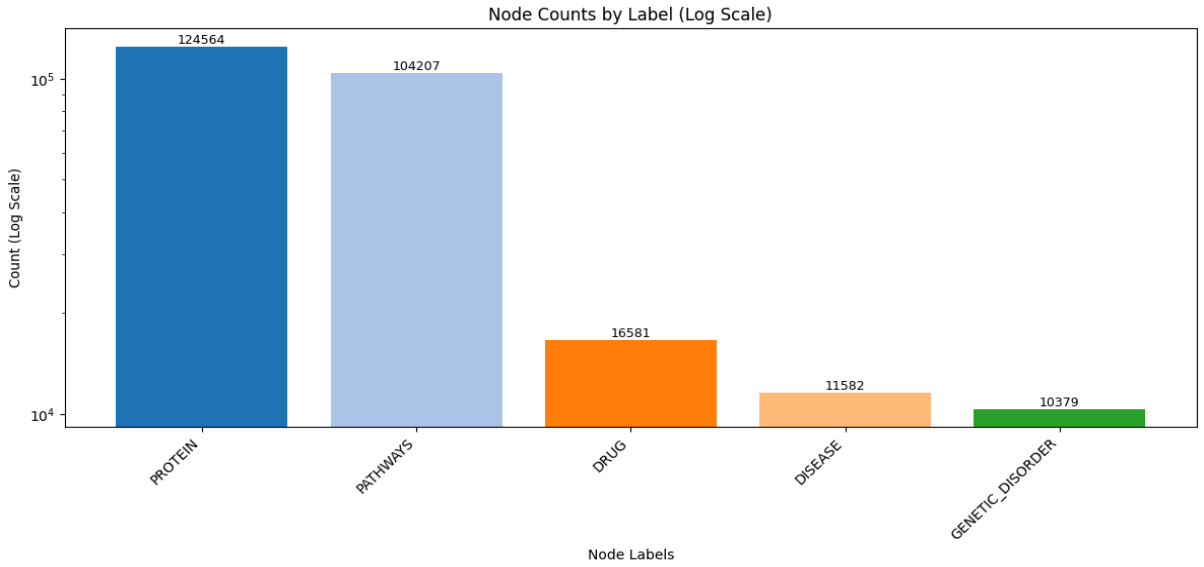

Fig 3.1. Number of Nodes with Respective Labels

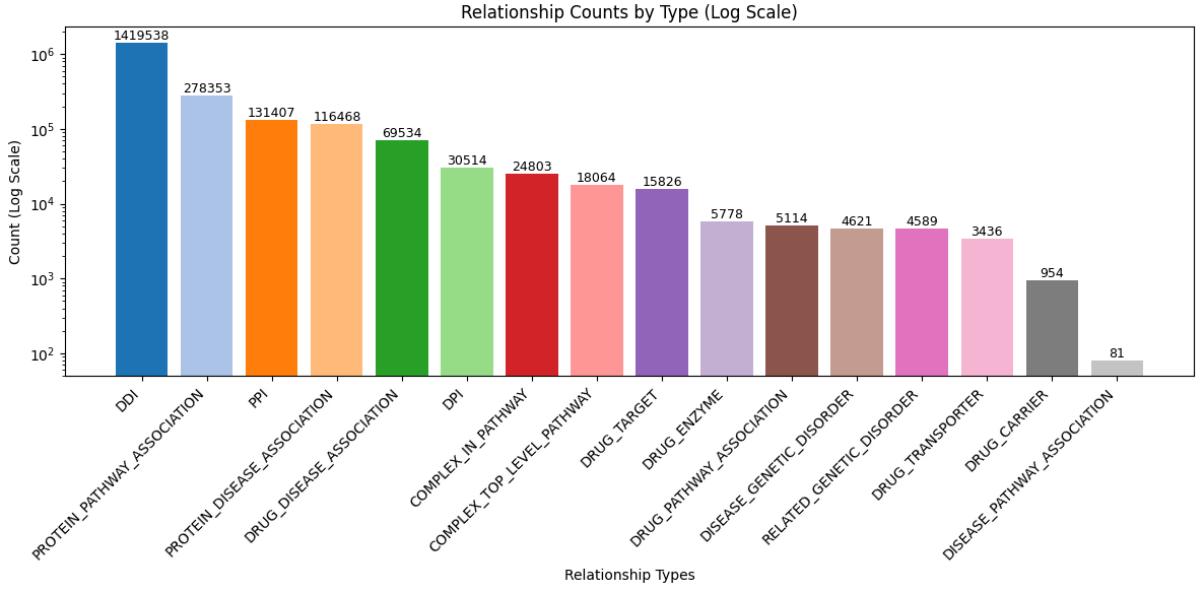

Fig 3.2 Number Of Relationships

3.1.2. Neo4j Integration and Schema Design

Neo4j was selected as the graph database management system due to its native graph storage and processing capabilities, powerful Cypher query language, and robust Python driver. The database instance was hosted locally (bolt://localhost:7690) under the database name myproject, accessed with standard authentication credentials.

A critical design consideration during integration was the faithful representation of **multi-valued properties**. The BioKG dataset often features scenarios where a single property key can have multiple associated values (e.g., a "Gene" node might have multiple "AssociatedDiseases"). To preserve this rich information without creating redundant nodes or complex intermediate structures,

these multi-valued properties were directly inserted into Neo4j nodes as **list (array) properties**. This approach leverages Neo4j's property graph model capabilities, allowing for efficient storage and retrieval of all values under a single key. For instance, node.symptoms = ["fever", "cough", "headache"] was used instead of individual symptom properties.

The graph schema was designed to reflect the inherent structure of biomedical data:

- **Nodes:** Entities were assigned distinct labels (e.g., Gene, Protein, Disease, Drug, Pathway) to categorize them. Each node possessed a NAME property for human readability and unique identifiers such as uniprot_id, drug_id, mesh_id, pathway_id, and mim_id for precise programmatic lookup.
- **Relationships:** Connections between nodes were represented by semantically meaningful relationship types (e.g., DDI, Drug_Enzyme_Association, PPI, DPI).

3.1.3. Dynamic Schema Extraction

To ensure that the language model-powered agents could interact with the knowledge graph effectively and generate accurate Cypher queries, a dynamic schema extraction mechanism was implemented using the kgschema.py script. This script connects to the live Neo4j instance and programmatically retrieves:

- All distinct **node labels** present in the graph along with their associated **property keys**.
- All unique **relationship types**, specifying the labels of their start and end nodes.

This extracted schema is then formatted as a JSON object and saved to kg_schema.json. This dynamic availability of the graph's structure is crucial for agents, particularly the rewrite_expansion_agent, allowing them to validate entities and properties mentioned in user queries against the actual graph schema.

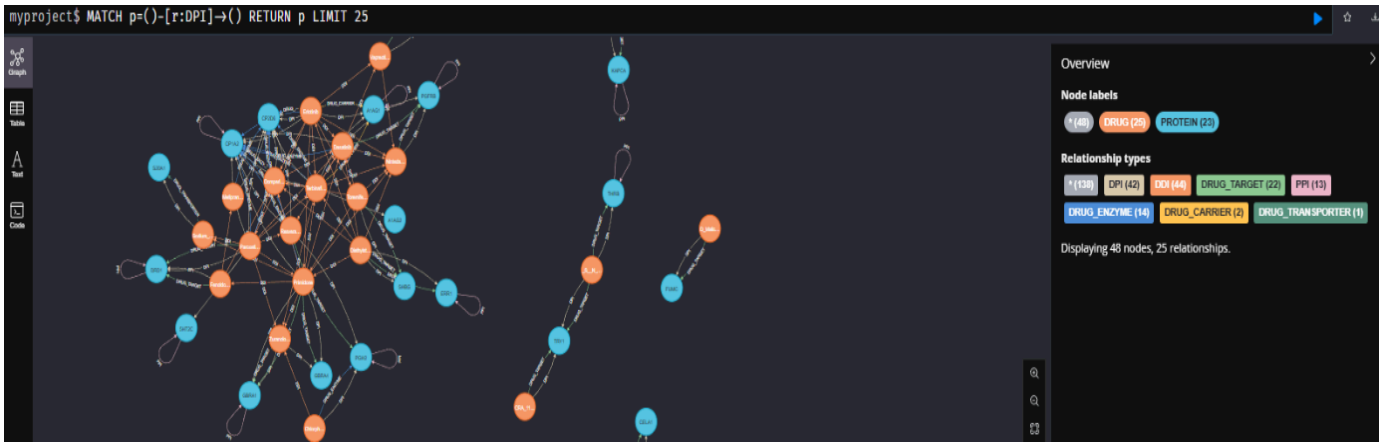

Figure 3.3: Screenshot of the Knowledge Graph Structure in Neo4j Browser

3.2. Initial Retrieval Approaches: Embedding Model Exploration

Prior to settling on BM25, a significant effort was invested in exploring embedding models for semantic information retrieval, given their prominence in modern NLP.

3.2.1. Models Explored

The following embedding models were investigated:

- **Nomic Atlas Embedding Model v1:** A general-purpose embedding model chosen for its reported performance across various domains.
- **BioBERT:** A BERT-based model pre-trained specifically on biomedical text, expected to capture domain-specific semantic nuances.
- **PubMedBERT:** Another BERT variant fine-tuned exclusively on PubMed abstracts, aiming for high relevance in medical literature.

3.2.2. Challenges and Rationale for Discarding

Despite their individual strengths, these embedding models proved "not beneficial" for directly enabling semantic search over the constructed BioKG knowledge graph due to several critical limitations:

- **High Data Heterogeneity:** The immense diversity of entity types and textual content within BioKG (e.g., gene sequences, disease descriptions, drug side effects) created a highly fragmented and inconsistent semantic space, hindering the models' ability to generate coherent and universally comparable embeddings.
- **Bias from Frequently Repetitive Terms:** Common biomedical terms or identifiers frequently appeared across disparate entities. The embedding models tended to assign similar vectors to these repetitive terms, leading to a "bias" where distinct entities might appear semantically close simply due to shared, but non-discriminative, vocabulary. This reduced the precision required for accurate entity resolution.
- **Architectural Mismatch with Key-Value Pairs:** Fundamentally, these models are designed for continuous, sequential text processing (sentences, paragraphs). They lack an intrinsic understanding of the structured nature of knowledge graph properties, especially when values are stored as **lists** for a single key. Attempts to flatten or preprocess these key-value structures for embedding often resulted in a loss of critical contextual meaning and the precise relationships between keys and their values.

Attempts to preprocess the data, such as removing numbers (believed to be meaningless noise) or frequently occurring words (to combat bias), further "disturbed the data" by stripping away essential identifiers or context, leading to degraded embedding quality rather than improvement. Consequently, a pivot to a more direct, keyword-based retrieval mechanism was deemed necessary.

3.3. Core Retrieval Component: BM25 System

Given the limitations of embedding models, the Okapi BM25 algorithm was adopted as the core component for initial node extraction due to its proven effectiveness in keyword-based information retrieval.

3.3.1. BM25 Algorithm Overview

BM25 is a ranking function that calculates the relevance of documents to a given query. It is a bag-of-words model that considers term frequency, inverse document frequency, and document length normalization to provide a relevance score. Its simplicity and robust performance make it a standard for many search applications where keyword matching is critical. In this system, BM25 functions as an intelligent retrieval tool, providing ranked lists of relevant nodes.

3.3.2. BM25 Indexing and Implementation

The implementation of the BM25 retriever is handled by the bm25_class.py module. The process involves:

1. **Node Loading:** All nodes from the Neo4j database that possess a NAME property are retrieved.
2. **TextNode Conversion:** Each Neo4j node is converted into a llama_index.core.schema.TextNode object. The NAME property of the Neo4j node is used to populate the text field of the TextNode (which BM25 indexes), while all other Neo4j properties are stored in the metadata field. This metadata is crucial for extracting unique identifiers like uniprot_id, drug_id, mesh_id, pathway_id, or mim_id later.
3. **Persistent Caching:** To optimize performance, these processed TextNode objects are serialized and saved to bm25_nodes.pkl. This caching mechanism ensures that nodes are loaded from disk on subsequent runs, significantly reducing the overhead of repeated database queries.
4. **BM25 Retriever Initialization:** A llama_index.retrievers.bm25.BM25Retriever instance is initialized from these TextNode objects. It is configured with similarity_top_k=2, meaning it returns the top two most relevant nodes, and utilizes an English stemmer to improve matching by reducing words to their root form.

3.3.3. Retrieve_with_bm25 Tool

The retrieve_with_bm25 function, exposed as a tool (tools.py), wraps the BM25 retriever's functionality. It takes a query string (or list of strings) and, after executing the BM25 retrieval, processes the results to return a simplified list of dictionaries. Each dictionary contains the node_label and the most relevant unique node_id (prioritizing uniprot_id, drug_id, mesh_id, pathway_id, or mim_id) for the retrieved nodes. This structured output is vital for subsequent agents to work with precise identifiers.

3.3.4. Limitations of Standalone BM25

While robust for keyword matching, BM25, when used in isolation, presented a significant "first challenge": its "approach limited us to use specific words in our query to get good results". This dependency on exact keyword matches meant that natural language queries or those requiring semantic inference beyond direct term matching would often yield suboptimal results, necessitating a more intelligent and adaptable retrieval system.

3.4. Multi-Agent Retrieval System Architecture

To overcome the inherent limitations of standalone retrieval methods and to facilitate complex query understanding and graph traversal, a sophisticated **multi-agent system** was designed and implemented.

3.4.1. LangChain Supervisor Framework

The multi-agent system is built upon the **LangGraph Supervisor library**. This framework provides a robust and flexible architecture for orchestrating multiple AI agents in a hierarchical manner. The core principle is that individual agents, each with a specialized function, operate under the direction of a supervisor. These supervisors, in turn, can be managed by a top-level supervisor, enabling a cascading decision-making process and clear control flow. This modularity allows for complex workflows to be broken down into manageable sub-tasks, enhancing system robustness, flexibility, and maintainability.

3.4.2. Language Model Integration

Every agent in our system relies on GPT-4o Mini as its reasoning core. This model interprets incoming queries, evaluates results from the different tools, decides which agent should handle each step, and weaves those pieces into a clear, cohesive response. Its deep understanding of natural language and strong problem-solving abilities are what let the system adapt on the fly and tackle even complex retrieval tasks.

3.4.3. Agent and Supervisor Roles

The final_agents.py script defines the intricate hierarchy and interaction logic of the agents and supervisors. The system comprises three main supervisors, all reporting to a top-level supervisor.

3.4.3.1. Query Processor Supervisor

The query_processor supervisor (query_processor in final_agents.py) is responsible for the initial understanding and refinement of the user's query. It guides the agents under its purview to prepare the query for effective downstream processing. Its primary objective is to classify the query's intent (web search, rewrite, or expansion) to optimize the subsequent retrieval steps.

- **Agent 1 (Query Router Agent):** Named agent1 in the implementation, this agent acts as the initial router based on the complexity and content of the user's query. Its logic dictates:
    - If the query contains **two or more node entities/IDs**, it routes to the rewrite_expansion_agent for **expansion**.
    - If the query contains **a single node entity/ID**, it routes to the rewrite_expansion_agent for **rewriting**.
    - If the query explicitly asks for external, current, or time-sensitive biomedical information, it routes to the web_search_agent.
- **Rewrite and Expansion Agent:** This agent (rewrite_expansion_agent) directly refines the query.
    - **Rewriting:** For single-entity queries, it aims to make the query more precise for BM25 retrieval without altering the core keywords.
    - **Expansion:** For multi-entity queries, it breaks down the query into distinct sub-queries, each focusing on a single entity, facilitating individual lookup and subsequent relationship discovery. This agent leverages the formatted_schema (from kg_schema.json) to validate node properties and ensure query consistency with the graph's structure.

- **Web Search Agent:** This agent (web_search_agent) is invoked when external information is required.
    - **Tool:** It exclusively uses the tavily_web_search tool to perform targeted web searches.
    - **Purpose:** Its role is to retrieve the latest or complementary information about a biomedical entity that might not be present in the static knowledge graph. It then returns a cleaned version of the query, stripping vague words, without generating answers itself.

3.4.3.2. Initial Node Extractor Supervisor

The initial_node_extractor supervisor (initial_node_extractor) focuses on pinpointing the most relevant starting nodes within the knowledge graph.

- **BM25 Retriever Agent:** This agent (bm25_agent) is the sole agent under this supervisor.
    - **Tool:** It uses the retrieve_with_bm25 tool (described in Section 3.3.3) to execute BM25 searches.
    - **Responsibility:** It receives the processed query from the query_processor and leverages BM25 to extract initial nodes. Its output is strictly limited to the node_label and a unique node_id (e.g., mesh_id, drug_id, uniprot_id) of the retrieved nodes, ensuring precision for the next stage.

3.4.3.3. Graph Traverser Supervisor

The graph_traverser supervisor (graph_traverser) is responsible for exploring the relationships between the identified nodes and synthesizing the final answer based on the graph's structure.

- **Graph Traversal Agent:** This agent (graph_traverser_agent) navigates the knowledge graph.
    - **Tool:** It uses the find_paths tool (from tools.py), which executes Cypher queries against the Neo4j database.
    - **Responsibility:** It takes the unique node IDs identified by the bm25_agent and attempts to find paths and relationships connecting them. The find_paths tool is configured to explore paths up to **3 hops** and returns a maximum of **2 paths** to manage computational complexity and result relevance. It directly returns the raw path information from the Cypher query.
- **Adaptive Retrieval Agent:** This agent (adaptive_retrieval_agent) is responsible for formulating the user-facing response.
    - **Responsibility:** It receives the discovered nodes and relationship information from the graph_traverser_agent. It then constructs a coherent explanation of the found connections. For direct relationships, it states the connection simply. For multi-hop connections, it describes the path step-by-step. A critical constraint on this agent is that it must strictly adhere to the information provided by the previous agents, **without hallucinating or adding external knowledge**. If no path is found, it explicitly states "No valid path found."

3.4.3.4. Top-Level Supervisor

The top_level_supervisor acts as the master orchestrator of the entire multi-agent system. It defines and enforces the high-level workflow, ensuring a seamless progression from initial query processing to final answer generation.

- **Workflow Orchestration:** Its prompt clearly dictates the sequence: user query goes to query_processor, its output is forwarded to initial_node_extractor, and the resulting node IDs are passed to graph_traverser. The top_level_supervisor ensures that outputs are passed unmodified between these main stages and that the final answer originates solely from the graph_traverser (specifically the adaptive_retrieval_agent).

3.4.4. Tool Implementation and Function Calling

The seamless operation of the multi-agent system relies heavily on the integration and intelligent use of various tools via **function calling**. Function calling allows the LLMs to dynamically invoke external functions (Python methods in tools.py) based on their understanding of the query and the current state of the workflow. The key tools and their functions are:

- retrieve_with_bm25: For BM25-based initial node retrieval.
- tavily_web_search: For real-time web information retrieval.
- find_paths: For executing complex Cypher queries to traverse the Neo4j knowledge graph and find relationships. This tool is instrumental in allowing agents to programmatically query the graph structure.

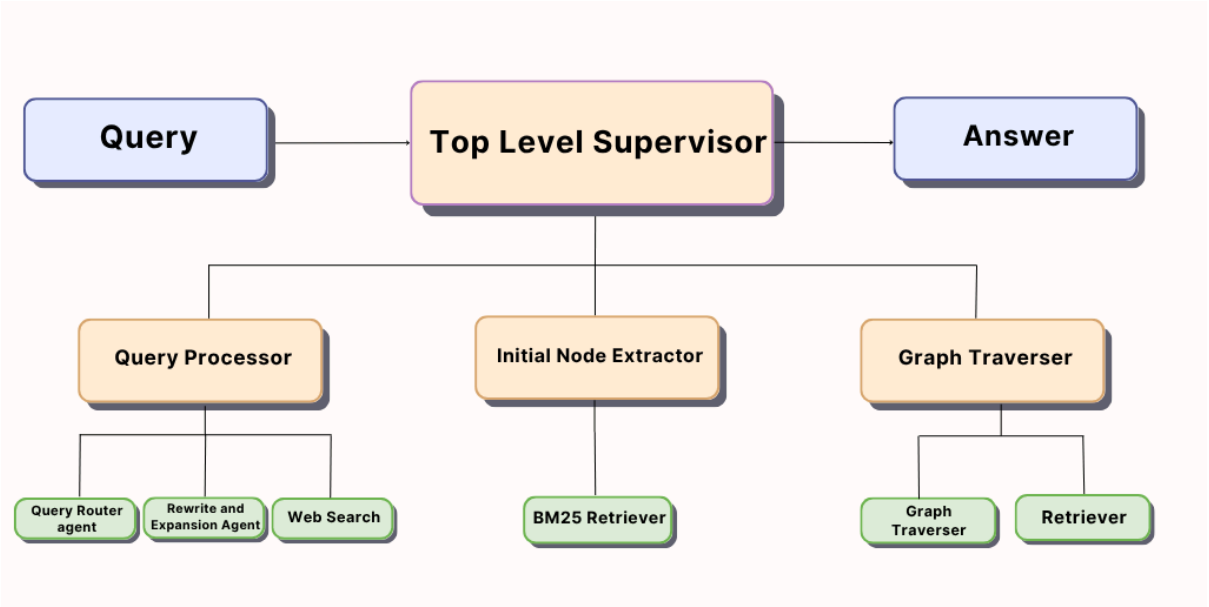

Fig 3.4 Flowchart of Langchain Supervisor Agent

By weaving together specialized agents under a central supervisor, our system can pick the right tool at each step—whether that's expanding a query, pulling nodes with BM25, traversing the graph with Cypher, or fetching fresh data from the web. This dynamic handoff means we can tackle multi-stage, nuanced questions that no single model or standalone retriever could manage on its own

3.5 Experimental Setup

We built and tested everything in a Python 3.10+ environment, relying on these main packages:

- **LangChain & LangGraph** for orchestrating agents

- **OpenAI's openai library** to access GPT-4o Mini via ChatOpenAI
- **Neo4j Python driver** to load and query our local graph database
- **llama_index** (BM25 implementation) plus **Stemmer** for text preprocessing
- **tavily-python** for integrated web searches

All components ran on a single machine, with Neo4j serving as our back-end store. This setup let us measure performance end-to-end—from user prompt to final answer—under realistic conditions.

Chapter 4: Results and Discussion

In this chapter, we evaluate how our multi‑agent retrieval system performs in practice. Through a series of illustrative case studies, we showcase its ability to interpret complex biomedical questions, extract relevant nodes from the knowledge graph, and traverse relationships to synthesize accurate, comprehensive answers. We compare these outcomes against simpler, standalone retrieval methods—highlighting where traditional approaches fall short and how our layered agent architecture overcomes those limitations. By examining both the reasoning steps each agent takes and the end results they produce, we demonstrate the system's effectiveness and its potential to advance intelligent querying over heterogeneous biomedical data.

4.1. Experimental Setup

The multi-agent system was implemented and tested within a Python 3.9+ environment. The core components included:

- **Graph Database:** Neo4j (version 5.22.0), running locally, hosting the constructed biomedical knowledge graph.
- **Language Model:** GPT-4o Mini, accessed via the OpenAI API, serving as the reasoning engine for all agents.
- **Agent Framework:** LangGraph Supervisor, orchestrating the hierarchical multi-agent workflow.
- **Key Libraries:** neo4j for database interaction, llama_index for BM25 retrieval, tavily-python for web search, and langchain/langgraph for agent development.

Queries were posed to the top_level_supervisor, which then managed the internal workflow and tool calls to produce a final response.

4.2. Qualitative Evaluation of System Performance

The system's performance was evaluated by submitting a variety of natural language queries designed to test its different functional layers. The results demonstrate its ability to adapt to query complexity and leverage appropriate tools.

4.2.1. Intelligent Query Processing

The query_processor supervisor, through its specialized agents, significantly enhanced the system's capability to understand and refine diverse user inputs.

- **Query Rewriting (Single Entity Example):**
  - **User Query:** "Tell me about gene EGFR."
  - **query_router_agent Action:** Identifies a single entity ("EGFR") and routes to rewrite_expansion_agent.
  - **rewrite_expansion_agent Output:** "properties of gene EGFR" or similar, making it more precise for keyword matching. This ensures that BM25 receives a well-formed query.
- **Query Expansion (Multiple Entities Example):**
  - **User Query:** "What is the relationship between drug 'DB00636' and 'D006930'

- **query_router_agent Action:** Identifies multiple entities (‘DB00636’, "D006930’") and routes to rewrite_expansion_agent.
- **rewrite_expansion_agent Output:** Splits into sub-queries like: ‘DB00636’, ‘D006930’ or internal representations that specify the entities for the bm25_agent. This allows for independent initial lookup of each entity.

- **Web Search Integration Example:**
  - **User Query:** "What are the latest clinical trials for 'Diabetes drug X'?"
  - **query_router_agent Action:** Recognizes the need for current information and routes to web_search_agent.
  - **web_search_agent Action:** Calls tavily_web_search with a refined query like "latest clinical trials Diabetes drug X".
  - **web_search_agent Output:** Returns a cleaned query for further processing, or if directly answering (not in current implementation), would have provided search results, demonstrating its capability to handle real-time or external data requirements.

4.2.2. Initial Node Extraction Accuracy

The initial_node_extractor supervisor, leveraging the bm25_agent and the retrieve_with_bm25 tool, consistently provided accurate initial nodes crucial for graph traversal. This demonstrated effective mitigation of the limitations observed with direct embedding model usage.

- **Example Query:** "Find information about drug clofibrate
- **bm25_agent Action:** After query processing, the retrieve_with_bm25 tool is called.
- **retrieve_with_bm25 Tool Output (Illustrative):**

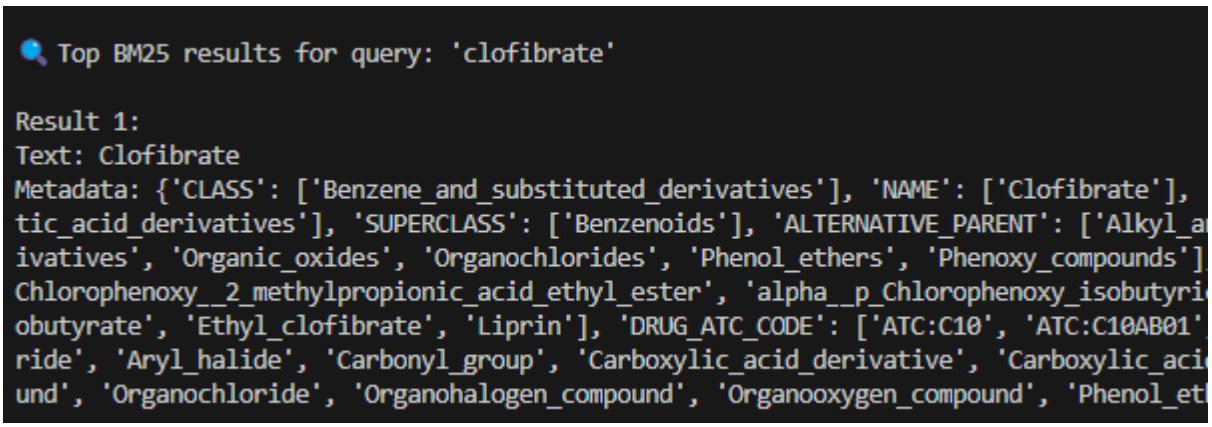

Fig 4.1 Output of BM25 Retriever tool

This output confirms the successful identification and extraction of precise node identifiers, which are then passed to the graph traversal stage. The BM25's ability to accurately retrieve nodes based on unique identifiers or well-formed names, even after LLM-driven query pre-processing, proved reliable for establishing anchor points in the graph.

4.2.3. Comprehensive Graph Traversal and Answer Generation

The graph_traverser supervisor, with its graph_traverser_agent and adaptive_retrieval_agent, showcased the system's strength in discovering and explaining relationships within the knowledge graph.

- **Direct Relationships Example:**
  - **User Query:** "Show me the connection between drug 'Doxorubicin' and disease 'Breast Cancer'."
  - **System Internal Steps:** query_processor refines, initial_node_extractor finds nodes for Doxorubicin and Breast Cancer.
  - **graph_traverser_agent Action:** Calls find_paths tool with the IDs of Doxorubicin and Breast Cancer.
  - **find_paths Tool Output**

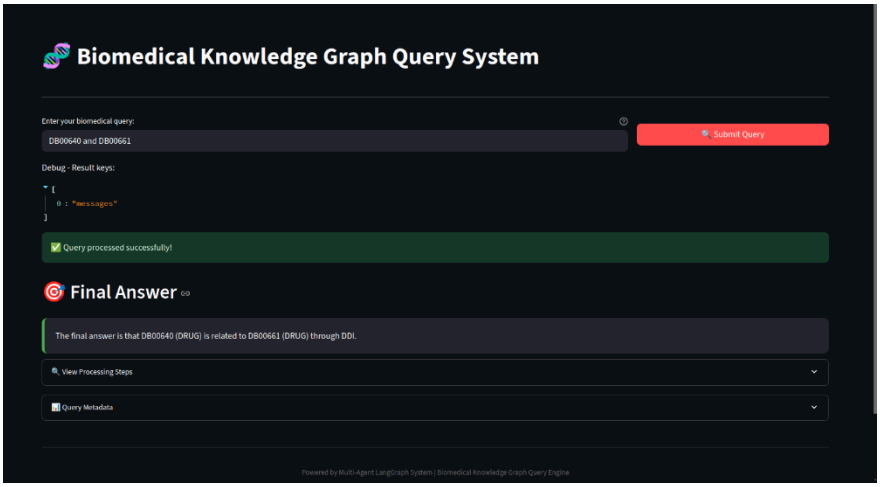

Fig 4.2 Raw Output of Graph Traversal Tool

  - **adaptive_retrieval_agent Final Answer:** "Drug DB00640 is related to DB00661 through DDI."
- **Multi-hop Relationships Example (up to 3 hops):**
  - **User Query:** "How is Gene 'MIM:267010 ' related to 'Q12797'?" (assuming a multi-hop path through a pathway or another protein)
  - **System Internal Steps:** query_processor and initial_node_extractor find the IDs for MIM:267010 and Q12797.
  - **graph_traverser_agent Action:** Calls find_paths with the respective IDs.
  - **find_paths Tool Output**

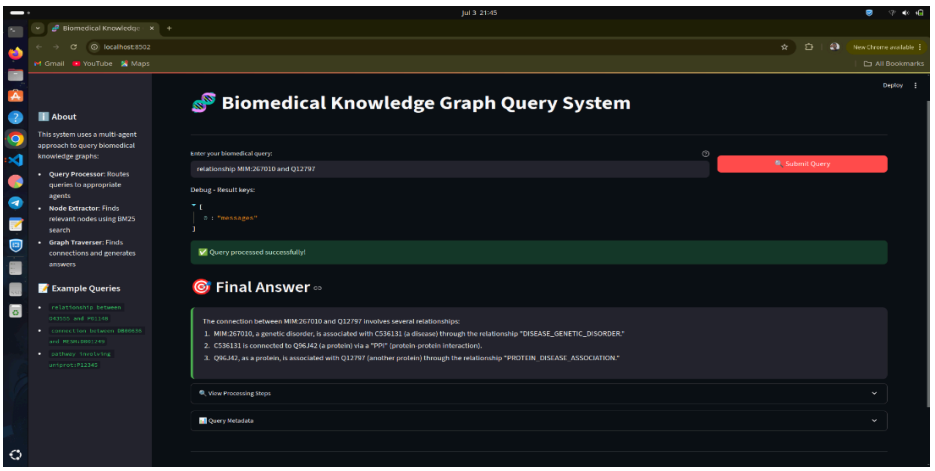

Fig 4.3 Multihop Graph Traversal Output

- **Adaptive_retrieval_agent Final Answer:** The connection Between MIM:267010 and Q12797 involve Several Relationship:
  - MIM:267010 a genetic Disorder is associated with C536131 (a disease) through a relationship “DISEASE_GENETIC_DISORDER”
  - C536131 is connected to Q96J42 (a protein) via “PPI” (protein protein interaction)
  - Q96J42, as a protein is associated with Q12797 through the relationship “PROTEIN_DISEASE_ASSOCIATION”

- **Handling "No Path Found" Example:**
  - **User Query:** "Connect gene 'XYZ' and drug 'ABC' (where no path exists in KG)."
  - **System Internal Steps:** query_processor and initial_node_extractor retrieve node IDs. graph_traverser_agent calls find_paths.
  - **find_paths Tool Output:** {"message": "No valid path found between the provided node IDs."}
  - **adaptive_retrieval_agent Final Answer:** "No valid path found between the provided node IDs." This shows the system's robust handling of negative results, preventing hallucination.

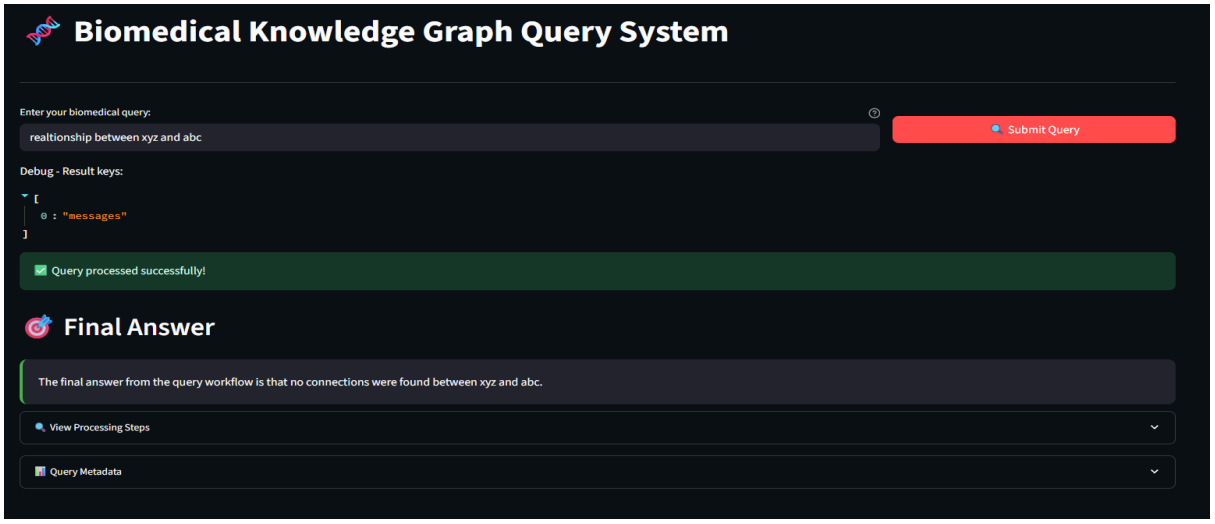

Fig 4.4 Graph Traversal showing NO relationship

Result with APOC:

APOC adds a set of useful tools to Neo4j that make working with graphs much easier and faster. It helps find paths between nodes more efficiently, even when there are multiple steps involved—something that can be slow with the usual queries. Besides pathfinding, APOC offers handy features for cleaning and transforming data, which helps get better results from the graph without writing complicated code.

Here are some results:

Query: shortest path between P41392 and R-CEL-5654219

Answer: The shortest path from P41392 to R-CEL-5654219 is through the protein Q11184, with the following relationships:

1. PPI (Protein-Protein Interaction) between P41392 and Q11184.
2. PROTEIN_PATHWAY_ASSOCIATION between Q11184 and R-CEL-5654219.

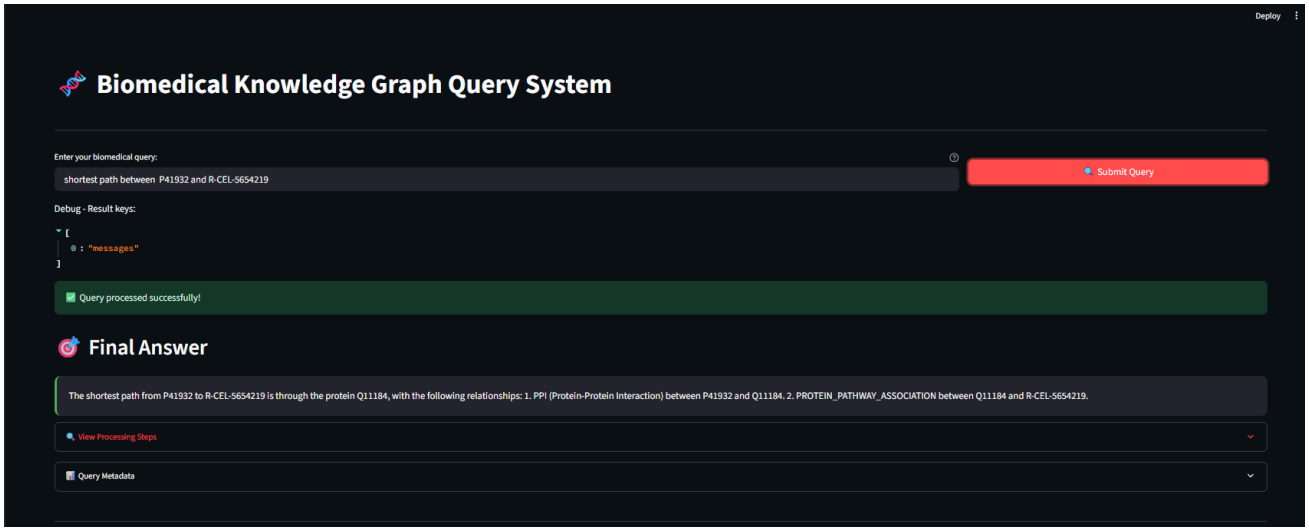

Fig 4.5 Shortest Path Between Two Nodes

Query-: Path between P41932 and Q11184 R-CEL-5654219 Q10656
Answer-: **P41932** (14-3-3-like protein 1) interacts with **Q11184** (let-756) through a protein-protein interaction.
Q11184 is associated with the pathway **R-CEL-5654219** and can link to protein **Q10656** (egl-15) through multiple associations with other pathways (**R-CEL-5654689**, **R-CEL-5654699**, **R-CEL-190371**, **R-CEL-1257604**, **R-CEL-6811558**).
This indicates that there are multiple routes from **P41932** to **Q10656** through the intermediary **Q11184** and the respective pathways.

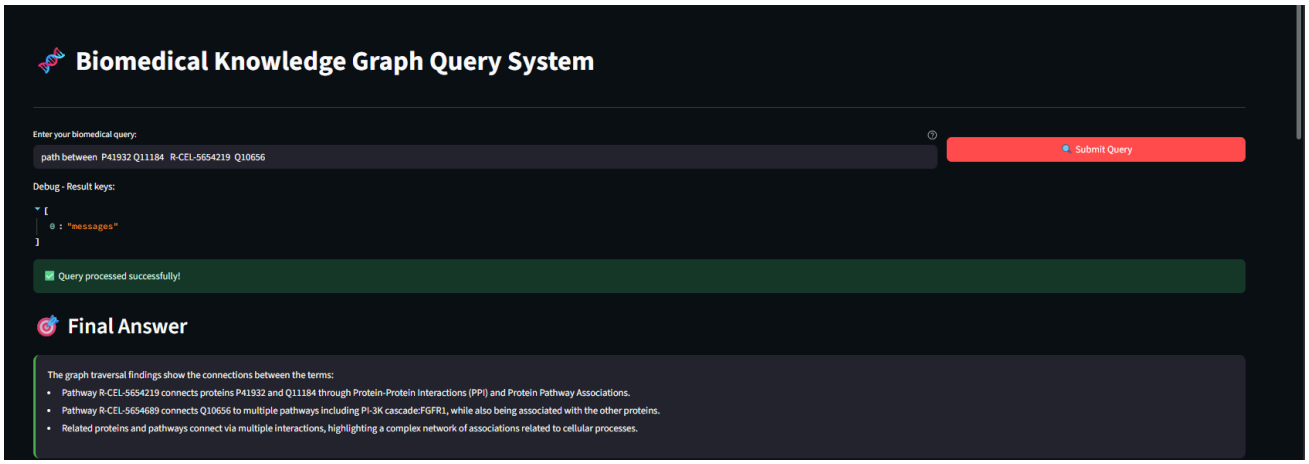

Fig 4.6 Path between Two nodes with Intermediate Nodes

Query-: Shortest Path between P41932 and Q11184 R-CEL-5654219
Answer-: The shortest path between the nodes is as follows:

P41932 (14-3-3-like protein 1 in *Caenorhabditis elegans*) is connected to **Q11184** (LET-756) through a protein-protein interaction (PPI).
Q11184 (LET-756) is associated with **R-CEL-5654219** (Phospholipase C-mediated cascade: FGFR1) through a protein-pathway association.
Thus, the sequence is: **P41932** → **Q11184** → **R-CEL-5654219**.

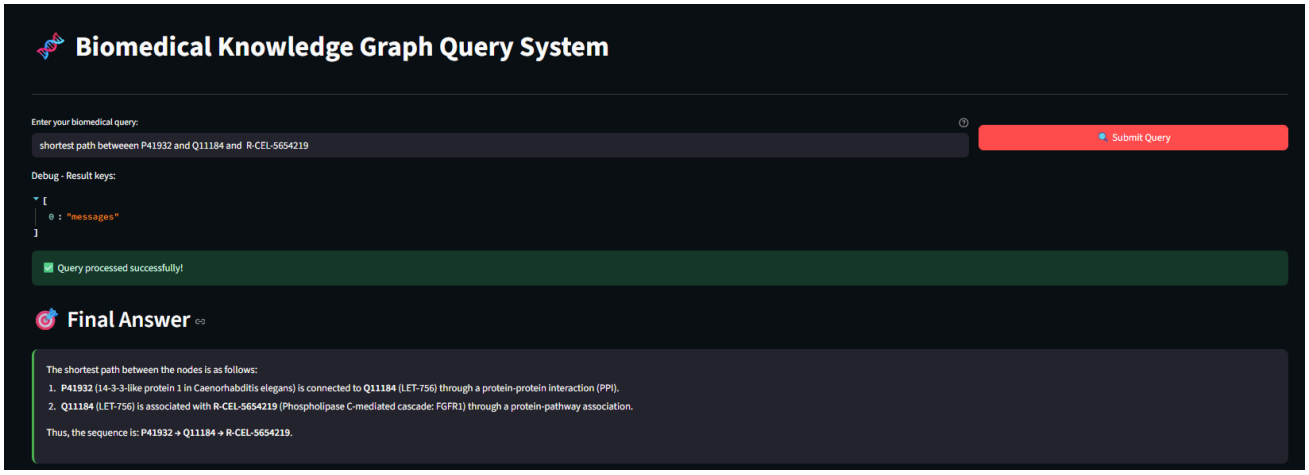

Fig 4.8 Shortest Path between Two Nodes with Intermediate Nodes

4.3. Discussion of Advantages

The results presented above underscore several key advantages of the developed multi-agent system:

- **Enhanced Robustness and Flexibility:** The modular architecture, with specialized agents for distinct tasks (query processing, retrieval, traversal), makes the system highly robust. It can gracefully handle variations in user queries, adapt its strategy based on query complexity, and orchestrate multiple tools seamlessly. This significantly improves flexibility compared to a single-model or single-tool approach.
- **Effective Mitigation of BM25's Limitations:** By integrating an LLM-powered query_processor supervisor, the system directly addresses the "specific words" limitation of BM25. Queries are intelligently rewritten or expanded, ensuring that the BM25 retriever receives optimized input, thus improving initial node extraction accuracy even from natural language questions.
- **Scalability and Maintainability:** The LangGraph Supervisor framework provides a clear separation of concerns. This modularity allows for easier debugging, updates, and scaling. New functionalities or improved tools can be integrated by simply adding or modifying specific agents without disrupting the entire system workflow.
- **Complex Multi-Step Reasoning:** The system effectively performs multi-step reasoning by chaining agent actions. From understanding a nuanced query, to retrieving initial entities, to traversing complex graph paths, and finally synthesizing a coherent answer, each step builds upon the previous one, guided by the LLM's intelligence and the supervisor's orchestration. This ability to perform complex, multi-hop reasoning over a knowledge graph is a significant strength.
- **Factual Accuracy:** By grounding the final answer in the explicit relationships and properties retrieved directly from the Neo4j knowledge graph and strictly prohibiting hallucination by the adaptive_retrieval_agent, the system ensures high factual accuracy in its responses.

4.4. Comparison with Prior Approaches

The successful deployment and performance of this multi-agent system highlight a critical advancement over the initial approaches explored in Chapter 3.

- **Beyond Embedding Models:** Unlike the direct application of embedding models, which struggled with the BioKG's data heterogeneity, repetitive term bias, and especially the structured nature of key-value pair lists, the multi-agent system circumvents these issues. It achieves semantic understanding through LLM-powered query processing and combines it with a reliable keyword-based retriever (BM25) and explicit graph traversal, demonstrating that a hybrid, orchestrated approach is more effective for this specific problem domain.
- **Augmenting BM25:** While BM25 alone is limited, its integration into the multi-agent system transforms it from a rigid keyword matcher into a highly effective initial node extractor. The LLM agents prepare the input for BM25 and then intelligently utilize its output for further graph reasoning, turning a basic retrieval tool into a powerful component of a larger, intelligent system.

Chapter 5: Conclusion and Future Work

5.1. Conclusion

This thesis successfully addressed the complex challenge of effectively retrieving information from a heterogeneous biomedical knowledge graph constructed from BioKG data in Neo4j. The initial phase involved the meticulous conversion of TSV data into a graph structure, specifically managing multi-valued properties by storing them as lists within node properties. This foundational work laid the groundwork for robust data representation.

Early attempts to leverage state-of-the-art embedding models (Nomic Atlas v1, BioBERT, PubMedBERT) for semantic retrieval faced significant limitations. The inherent heterogeneity and potential biases of the biomedical data, coupled with the models' lack of native support for structured key-value pairs, rendered these approaches largely ineffective. This led to the strategic pivot to BM25 as the primary initial retrieval mechanism, acknowledging its efficacy in keyword-based matching but also recognizing its inherent limitation of requiring precise query terms.

The core innovation and contribution of this work lie in the design and implementation of a sophisticated **multi-agent information retrieval system**, built upon the LangGraph Supervisor framework and powered by GPT-4o Mini. This hierarchical system effectively mitigates the challenges posed by both data complexity and the limitations of individual retrieval tools. The system's agents, organized under dedicated supervisors (Query Processor, Initial Node Extractor, Graph Traverser), seamlessly orchestrate:

- **Intelligent Query Pre-processing:** Agents for query rewriting, expansion, and conditional web search (via Tavily Search) intelligently adapt user queries, making them suitable for the underlying BM25 retriever and augmenting information with real-time data when necessary.
- **Robust Initial Retrieval:** The BM25 retriever, as an "AI tool," reliably extracts initial nodes, serving as crucial anchors for subsequent graph exploration.
- **Adaptive Graph Traversal:** Agents specifically designed for graph traversal leverage dynamic Cypher query generation (via function calling) to discover complex relationships and paths within the knowledge graph.
- **Contextualized Answer Generation:** A dedicated agent synthesizes the retrieved graph information into clear, accurate, and structured responses, ensuring factual consistency by avoiding hallucination.

The collaborative and modular nature of the multi-agent system not only delivered a robust and flexible solution for biomedical information retrieval but also demonstrated a significant advancement over isolated retrieval techniques. By combining the strengths of keyword-based retrieval with the reasoning and orchestration capabilities of large language models, this system provides a powerful framework for navigating and extracting insights from complex biomedical knowledge graphs, paving the way for more intelligent biomedical discovery and analysis.

5.2. Limitations

Despite its successful implementation, the developed system has certain limitations that provide avenues for future research:

- **BM25 Dependency and Query Specificity:** While the multi-agent system significantly mitigates BM25's limitation, the core reliance on keyword matching for initial node extraction still means that entities without a well-defined NAME property might be harder to retrieve directly. The system's performance is sensitive to how well the query maps to existing node names or identifiers.
- **Predefined Hop Limit:** The find_paths tool has a fixed maximum hop limit (currently 3). While this design choice helps manage computational complexity and response time, some deeper, more indirect, but potentially relevant relationships within a very large graph might not be discovered.
- **Computational Resources for LLMs:** While GPT-4o Mini is more cost-effective than larger models, the continuous invocation of LLMs for decision-making and response generation still incurs computational and financial overhead, particularly for very high query volumes.
- **Static Knowledge Graph:** The current knowledge graph is static after its initial construction. Biomedical knowledge is constantly evolving, and manual updates or a more dynamic data ingestion pipeline would be required for the graph to remain current without relying solely on web search for up-to-date information.
- **Limited Interpretability of LLM Decisions:** While the supervisor-agent structure provides a traceable workflow, the internal reasoning and specific choices made by the GPT-4o Mini within each agent (e.g., how it chooses to rewrite a query or interpret traversal results) remain largely opaque, representing a degree of "black box" behavior.
- **Error Handling in Neo4j Queries:** While the system attempts to handle errors from Cypher queries, unexpected data structures, edge cases in graph topology, or highly complex query patterns could still lead to less than optimal retrieval or agent failures.

5.3. Future Work

Building upon the foundations laid by this thesis, several promising directions for future research and development emerge:

- **Hybrid Retrieval with Improved Embeddings:** Revisit embedding models with more sophisticated techniques tailored for handling heterogeneous key-value data in knowledge graphs. This could involve:
    - Developing custom embedding strategies that intelligently concatenate or combine key-value pairs, including list properties, into semantically meaningful inputs for embedding models.
    - Exploring Graph Neural Networks (GNNs) for node and relationship embeddings, which can intrinsically capture graph structure and relational information, potentially offering richer semantic representations for biomedical KGs.
    - Implementing a hybrid retrieval approach that combines the strengths of BM25 for keyword matching with embedding-based semantic search for re-ranking or initial candidate generation.
- **Dynamic Knowledge Graph Updates:** Develop a more automated and scalable pipeline for continuously updating the knowledge graph with new biomedical data. This could involve:
    - Integrating with real-time data streams or regularly updated scientific literature databases.
    - Leveraging LLMs or advanced NLP techniques for automated entity and relationship extraction from new publications, enabling incremental enrichment of the graph.
- **Advanced Graph Reasoning and Inference:** Enhance the graph_traverser and adaptive_retrieval_agent to perform more complex reasoning beyond simple pathfinding, such as:

- Inferring new, implicit relationships based on existing patterns and logical rules within the graph.
- Enabling the system to answer more complex "why" or "how" questions that require multi-step logical deduction across multiple subgraphs or entities.
- Incorporating probabilistic reasoning or uncertainty modeling into graph traversals to provide confidence scores for inferred relationships.

- **User Interface and Interactive Query Refinement:** Develop a user-friendly graphical interface that allows for intuitive interaction with the multi-agent system. This interface could provide visual representations of the retrieved graph paths, allow users to refine queries interactively, and offer explanations for the agent's reasoning steps.

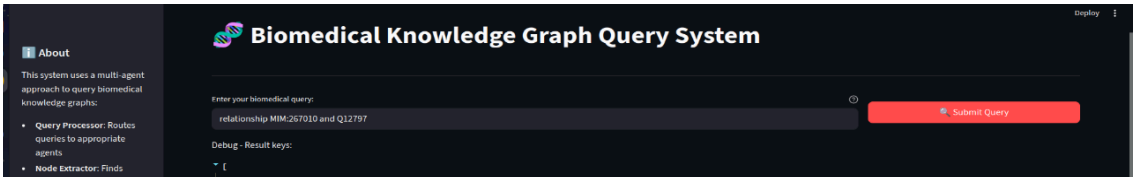

Fig 5.1: Interactive User Interface for Visualizing Knowledge Graph Expansion Outcomes

- **Explainable AI (XAI) for Agent Decisions:** Explore methods to make the LLM-powered agents' internal decision-making processes more transparent and explainable. This could involve prompting agents to provide their rationale for query routing, rewriting choices, or the selection of specific tools and paths, increasing user trust and understanding.
- **Broader Domain Applicability:** Evaluate the adaptability and performance of this multi-agent architecture in other complex, heterogeneous knowledge graph domains beyond biomedicine, such as finance, legal, supply chain management, or materials science, demonstrating its generalizability.

By systematically addressing these limitations and exploring these future directions, the robust foundation established in this thesis can be extended to create even more powerful, versatile, and intelligent knowledge graph-driven systems, pushing the boundaries of automated knowledge discovery and analysis.

The complete source code for this work is publicly available on GitHub at: [link].

References

[1] “BM25 Retriever - LlamaIndex.” Accessed: Jul. 06, 2025. [Online]. Available: https://docs.llamaindex.ai/en/stable/examples/retrievers/bm25_retriever/

[2] D. Edge *et al.*, “From Local to Global: A Graph RAG Approach to Query-Focused Summarization,” Feb. 19, 2025, *arXiv*: arXiv:2404.16130. doi: 10.48550/arXiv.2404.16130.

[3] “Getting Started - GraphRAG.” Accessed: Jul. 06, 2025. [Online]. Available: https://microsoft.github.io/graphrag/get_started/

[4] S. Hedden, “How to Implement Graph RAG Using Knowledge Graphs and Vector Databases,” Towards Data Science. Accessed: Jul. 06, 2025. [Online]. Available: https://towardsdatascience.com/how-to-implement-graph-rag-using-knowledge-graphs-and-vector-databases-60bb69a22759/

[5] “Graph RAG | 🦜🔗 LangChain.” Accessed: Jul. 06, 2025. [Online]. Available: https://python.langchain.com/docs/integrations/retrievers/graph_rag/

[6] “langchain-ai/langgraph-supervisor-py.” Accessed: Jul. 06, 2025. [Online]. Available: https://github.com/langchain-ai/langgraph-supervisor-py

[7] T. Procko and O. Ochoa, “Graph Retrieval-Augmented Generation for Large Language Models: A Survey,” *Social Science Research Network, Rochester, NY*: 4895062. Accessed: Jul. 06, 2025. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4895062

[8] “Tavily Search | 🦜🔗 LangChain.” Accessed: Jul. 06, 2025. [Online]. Available: https://python.langchain.com/docs/integrations/tools/tavily_search/