https://doi.org/10.48047/AFJBS.6.Si4.2024.81-103

AFJBS

African Journal of Biological Sciences

Journal homepage: http://www.afjbs.com

Research Paper Open Access

Evolutionary Approach to Improve Relationship Awareness of Retrieval Augmented Generation through Knowledge Graphs – A case study of life sciences and healthcare compliances

Amit Chakraborty[1], Lumbini Bhaumik[2], Sushmita Ganguly[3], Chirantana Mullick[4], Saptarshi Das[5], Raj Kumar Keshri[6]

1,2,3,4,5,6 – JIS Institute of Advanced Studies and Research

Article Info: Volume 6, Issue Si4, 2024:Received: 12 Apr 2024:Accepted: 02 May 2024

doi:10.48047/AFJBS.6.Si4.2024.81-103

Abstract

Generative AI, specifically retrieval-augmented generation, is transforming life sciences and healthcare compliances compliances by improving decision-making processes. AI systems may provide specialized reports, projections, and investment plans by analyzing massive volumes of pharmaceutical and contextual information. These systems use retrieval techniques to generate accurate and contextually relevant outputs based on historical data, market trends, and expert views. This work describes a novel evolutionary strategy for improving connection awareness in retrieval augmented generation (RAG) systems, with an emphasis on compliance documents for life sciences and healthcare compliances industry. Using the power of knowledge graphs and graph databases, our approach provides a comprehensive framework for modeling documents and their deep relationships, allowing for more effective information retrieval and creation processes. We offer a method for leveraging retrieval-augmented generation from a graph database, in which documents are represented as nodes and relationships as edges, allowing for the extraction of rich contextual information. Furthermore, we present a similarity-based searching strategy for the graph database, allowing for more precise and relevant document retrieval. To assess the efficacy of our technique, we undertake a life sciences and healthcare compliances compliances case study that examines the effects of relationship-aware retrieval enhanced generation on important metrics including ROUGE and BLEU. We show considerable gains in these indicators after iterative experimentation, demonstrating that the created information is of higher quality and relevance. By including relationship awareness into the retrieval augmented generation process, our method allows finance professionals to access and develop insights with improved clarity, accuracy, and contextually. Our findings indicate the potential for using knowledge graphs and graph databases to improve the capabilities of retrieval-augmented generation systems in life sciences and healthcare compliances compliances applications. Beyond typical keyword-based retrieval approaches, our approach provides a more complex knowledge of document linkages, resulting in better decision-making processes. Furthermore, the iterative nature of our evolutionary approach enables continual refinement and adaptation, guaranteeing that the system stays effective in dynamic and changing situations. Finally, our findings add to ongoing efforts to advance retrieval augmented generation techniques by emphasizing the importance of relationship awareness and the use of knowledge graphs. By incorporating these concepts into the fabric of life sciences and healthcare compliances applications, we open the door to more intelligent, context-aware systems that provide finance professionals with actionable insights and decision support.

Keywords: Generative AI, retrieval-augmented generation, life sciences and healthcare compliances, knowledge graphs, graph databases, relationship awareness, information retrieval, contextual information, similarity search, metrics improvement, ROUGE, BLEU, decision-making processes, iterative approach.

Compliances for Life sciences and healthcare compliances Industry

This section recognizes healthcare compliance as the process of following legal requirements set by the authorities that regulate the functioning of healthcare organizations. These compliances are essential for upholding the integrity of health care, protecting the patients' information amidst various emerging threats, and guaranteeing that excellence is upheld in the provision of the services. Major bodies of healthcare compliance consist of HIPAA rules, ACA, HITECH Act, and other rules regulated by the CMS. As in any other business process where compliance requirements are met, documentation is a critical aspect that serves multiple purposes, one of which is providing proof of conformance and compliance, and the other acting as an audit trail for identifying corrective measures in a bid to enhance compliance.

HIPAA Compliance

The HIPAA is aimed at assuring patient confidentiality and the adoption of health information exchange. To comply with HIPAA, healthcare providers must:To comply with HIPAA, healthcare providers must:

Privacy Rule: Ensure confidentiality of all information that is in a way can identify any given individual health information.
Security Rule: Protect ePHI from unauthorized access, improper disclosure, alteration, destruction and the ability to be accessed by authorized personnel.
Documentation Requirements:

Privacy Notices: Explain to patients basic rights related to their information and how this information will be utilized.

Authorization Forms: Ensure that you get consent from the patient for using or sharing their information for secondary use other than for treatment, payment or health operations.
Security Policies and Procedures: Prescribe document security measures that can help to prevent ePHI disclosure, such risk analyses, access restrictions, or tracking tools.
Training Records: Communicate to the staff current HIPAA rules as well as data protection procedures.
Incident Reports: Keep records of any threats or lessons of a prohibited kind and the steps taken to counter them.

ACA Compliance

ACA which is ACA: was designed to make healthcare more affordable and easily accessible to citizens, provisions that have affected almost all sectors within healthcare and insurance.

Documentation Requirements:

Employee Health Insurance Coverage: Contract complaints also require proof of the health coverage provided to employers by the respective employer, forms like the 1095-C.
Patient Protection and Affordable Care Act (PPACA) Compliance: Providers must ensure they have documented their conformance to the rules set in issues of preventive services, patient right, and billing.

Financial Assistance Policies: Hospitals also have to maintain policies addressing financial interactions with patients within the hospitalized populace and guarantee these policies for the public.

HITECH Act Compliance

In addition to it, enhancing the use of electronic health records HITECH acts to reinforce the HIPAA regulations.

Documentation Requirements:

Meaningful Use: Providers are required to report on their adoption and use of EHRs for the purpose of enhancing patient care and patient outcomes, including which meaningful use objectives and measurements of meaningful use have they met.
Breach Notification: The records must be kept of any breach in compliance with Section 13405(b) of ePHI and documentation of any notices provided to individuals affected and the HHS in accordance to Section 13405(c).
Security Enhancements: The document should provide more information on safeguards that have being put in place to protect ePHI including measures like encryption and the manner in which data is transmitted in the network.

CMS Compliance
Here, the Consolidated Medicare and Medicaid (CMS) is responsible for managing Medicare and Medicaid and regulating its quality as well as rates of payment.

Documentation Requirements:

Billing and Coding: Individual patient records for encounters, diagnoses, and treatments so that reimbursement may be claimed appropriately.

Quality Reporting: Use of reports – recording of quality initiatives of the Quality Payment Program, including patient experience and MIPS scores.
Audit Trails: Ensure that records of all filed claims and money received, together with alterations to the records in case of correcting errors, are well kept.

Documentation is a key practice in the field of healthcare mainly, for purposes of ascertaining to the set rules and regulations, openness and for purposes of enhancing contact. Most healthcare organizations, practitioners, and stakeholders aim at being legal compliant, patient centered, and quality focused, which can only be achieved by proper documentation that will see one avoid the law, deliver better patient centered care, and build and sustain the much needed trust with the patients as well as the regulators. Overall, healthcare compliance can be more easily managed if healthcare organisations remain alert towards the keys to compliance and keep record of all the corresponding evidences.

Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation or RAG is one of the most popular methods to combine contextual information while querying a LLM so that the LLM while generating answers, contextualize it with the information provided to it. LLMs are in general trained in publicly available data set and while they can generate human like text they are not capable of contextualizing the generation of questions are asked against a custom textual corpus. The corpus maybe any custom document in form of PDF, MS or Open Office documents, images etc. The custom corpus that has to serve as the source of contextual information is first chunked, that is a pre-trained embedding model is used to create the vector representations of the chunks. The node parser takes the list of documents and chunks them into NODE object such that each NODE is of a pre-configured size. When a document is broken into nodes all its properties are inherited by the node. Each of the chunks are then embedded that is their vector representations are generated and stored in a vector DB. When a query is passed to the LLM through the use of a prompt the vector DB generates the closest vector embeddings from the question. To do this various distances can be used such as:

Euclidean distance – This is the straight line distance considering the space as a Euclidean space and is given by:

$$\sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

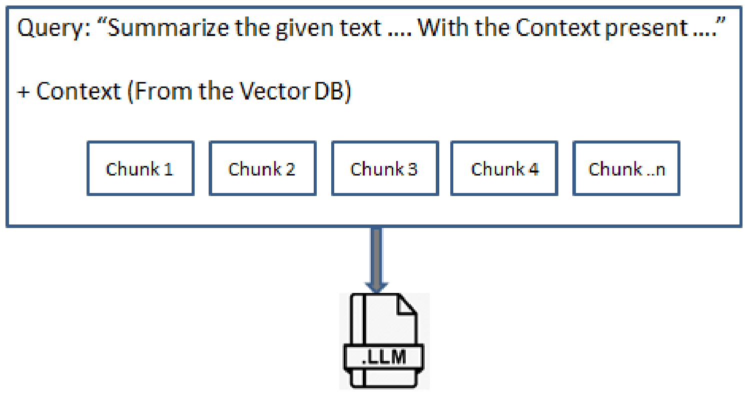
Cosine Distance – This is the cosine of angle between the two vectors in a multidimensional space and is given by:

$$\frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Jaccard Similarity – This calculates the distance between the two texts by calculating their union and intersection and is given by:

$$\frac{|A \cap B|}{|A \cup B|}$$

The Jaccard distance is then calculated by 1 – Jaccard Similarity


Query: "Summarize the given text .... With the Context present ...."
+ Context (From the Vector DB)
Chunk 1
Chunk 2
Chunk 3
Chunk 4
Chunk ..n
.LLM

The embeddings thus selected by the vector database on the basis of the shortest distance are then added to the query and then passed on to the LLM. The inherent capability of text generation of the LLM along with the contextual information passed in form of the vector chunks helps the LLM to answer questions only in the context of the documents passed.

Problem Statement

With the advent of readily accessible Large Language Models (LLM) the applicability areas for the same has increased by manifolds. Ranging from Banking and Finance use cases to managing products in medical devices the applicability of these models is widespread. These models can be used in as a cloud resource across all popular cloud platforms and containerized to be used in a local on premise or private virtual network environment. For both the cases primarily the custom uses cases are fulfilled either by the method of "Retrieval Augmentation Generation (RAG)" or "Fine Tuning" the model. Retrieval Augmentation Generation (RAG) refers to the process by which a LLM model is grounded by custom knowledge by providing suitable examples and context in form of context vectors so that the model can be used for custom business applications.

We propose to study such problems in a typical RAG implementation of a LLM that has been used to summarize various domain sensitive supplier and vendor management documents in a typical supply chain management process. An analytical study of the as-is deployment architecture of the solution followed by a detailed causal analysis of the problems associated with it shows that the summaries though effectively being contextual through Retrieval Augmented Generation process lacks the nuances of very customized domain specificity. The said problems will be discussed in detail below and can be associated with general efficiency of the model and the ecosystem in which it is being deployed.

In general, the knowledge representation of these models depends on the generalized information in which they have been trained which results in the model to regurgitate responses that are either too generic in nature. As will be detailed in the later sections' observation from a typical LLM on-premises or cloud deployment that addresses the above mentioned supply chain document or policy manual data shows that on using RAG the application using the models at backend are susceptible to problems such as:

- Limited Context and over reliance on retrieval – This limits a models output to be novel and creative as the model is now overtly dependent on the context passed. The problem of contextualization without domain awareness as seen in the above section limits the usage and reduces summarization efficiency. When compared with human made summaries of the same notes it performs not only with lower values of ROGUE and METEOR but also effects qualitative measures such as HELLASWAG.
- Domain understanding - The efficacy of the RAG model is highly dependent on the domain and the quality of the retrieval corpus. Where high-quality retrieval resources are scarce or where the context is complex and nuanced, RAG models may struggle to provide accurate and consistent answers. The absence of knowledge discovery before contextualizing the responses from LLM impacts retrieval quality. RAG relies heavily on the contextual embeddings that are passed to the LLM along with the query to get a response. Mere contextualization doesn't often represent modeling, the sense of which is to abstract real world elements and domain specificities to be used as a human construct. A simple-to-implement core subdomain can only offer a transient competitive edge. Core subdomains are therefore inherently complex. Business operations carried out by all organizations in an identical manner are referred to as generic subdomains. Similar to the main subdomains, generic subdomains are typically difficult to construct and sophisticated. Generic subdomains, however, don't give the business an edge over others. Here, innovation or optimization is not necessary because there are widely available, tried-and-true solutions that are utilized by all businesses.

- Evaluation and Retrieval quality challenges – It is difficult to assess the quality of LLM generated response when RAG is applied. The metrics generally used to assess the quality of a generative model might not be the best fit to assess a RAG generated response in terms of relevance, fidelity and coherence. The continuous emphasize on a limiting context without a general awareness of the domain, i.e., more specifically usage of the same context for queries that also requires a mental map of the domain is also responsible for repetition and hallucination of the said models.

The problems described above generally occur due to the fact that retrieval augmented generation process normally deals with only retrieval augmentation with contextual data and not domain specificity.

Metrics to quantify the problems

ROUGE Metric

In this paper ROUGE metric is used throughout to determine the quality of the summarization generated by the LLMs. ROUGE or Recall Oriented Understudy of Gisting Evaluation is a metric that is used to determine the quality of machine generated summarization and translation. This metric compares a machine generated summary against a high quality summary produced by a domain expert.

ROUGE-N measures the number of N-grams that match between the machine generated summary and the human produced summary. Instances of this can be ROUGE-1 (when 1 gram is considered), ROUGE-2 (when bi-gram is considered).

ROUGE-L – This depends on the Longest Common Subsequence between the machine generated and human produced summary. The longer the common subsequence between the two summaries is the higher is the ROUGE-L score. This subsequence may not be necessarily consecutive but should be in order.

In this study ROUGE-L was chosen against BLeU for the following reasons:

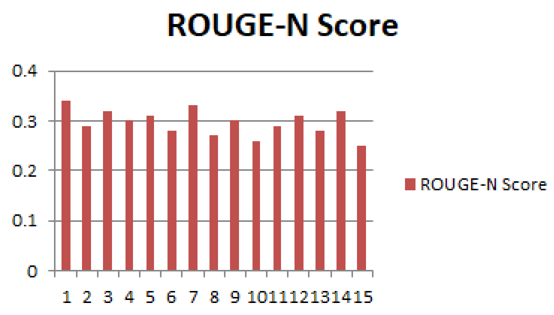
- Recall orientation – It was important to compare a human (read subject matter expert) generated summary with that of the machine generated one. Hence a measure of recall was required to take into account that the nuance of domain specificity is considered in the summary.
- N-Gram subsequence – ROUGE captures subsequence considering N-Gram overall which is customizable. It can capture not just exact matches but also partial matches thus making it more flexible. This allows even for sentence level evaluation.
- Abstractive Summarization Nuances – ROUGE-L gives a more effective understanding of the quality of the summary by focusing on the overlapping words and parts of the sentences and is thus more content focused.

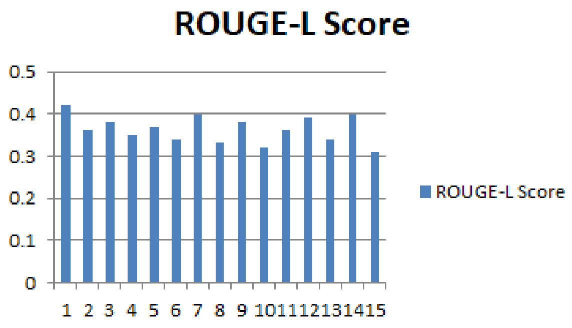
BLEU

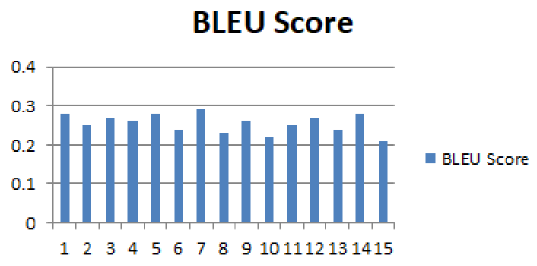
BLEU compares the generated response to a reference text (in this case, the prompt) by calculating the overlap of n-grams (contiguous sequences of n elements, often words or tokens) between the two texts. The metric rewards replies that have similar n-grams to those in the reference text. To calculate BLEU, we first count the number of n-grams in the generated response and the reference text to determine their precision.
We next divide this figure by the entire number of n-grams in the resulting response.
A brevity penalty is then applied to this precision score to account for the length difference betw een the generated response and the reference text.
The final BLEU score is the geometric mean of the adjusted precision scores over distinct n-gram orders (usually 1–4), weighted equally.
Higher BLEU scores imply a stronger resemblance between the generated response and the refer ence text. While BLEU is a useful metric for assessing the overall quality and fluency of generated text, it has several limitations, notably in terms of capturing semantic similarity and grammatical accuracy. As a result, data scientists frequently combine BLEU with other measures, like as cosine similarity and ROUGE, to provide a more thorough assessment of big language models' success in generating responses that closely match the provided prompt.

Measures taken from existing architecture

Iteration	**ROUGE-N Score**	**ROUGE-L Score**	**BLEU Score**
1	0.34	0.42	0.28
2	0.29	0.36	0.25
3	0.32	0.38	0.27
4	0.30	0.35	0.26
5	0.31	0.37	0.28
6	0.28	0.34	0.24
7	0.33	0.40	0.29
8	0.27	0.33	0.23
9	0.30	0.38	0.26
10	0.26	0.32	0.22
11	0.29	0.36	0.25
12	0.31	0.39	0.27
13	0.28	0.34	0.24
14	0.32	0.40	0.28
15	0.25	0.31	0.21







Standard Deviations across the prompts for this implementation are as follows:

- ROUGE-N Score: $sROUGE-N \approx 0.025$$sROUGE-N \approx 0.025$
- ROUGE-L Score: $sROUGE-L \approx 0.032$$sROUGE-L \approx 0.032$
- BLEU Score: $sBLEU \approx 0.027$$sBLEU \approx 0.027$

Proposed Architecture

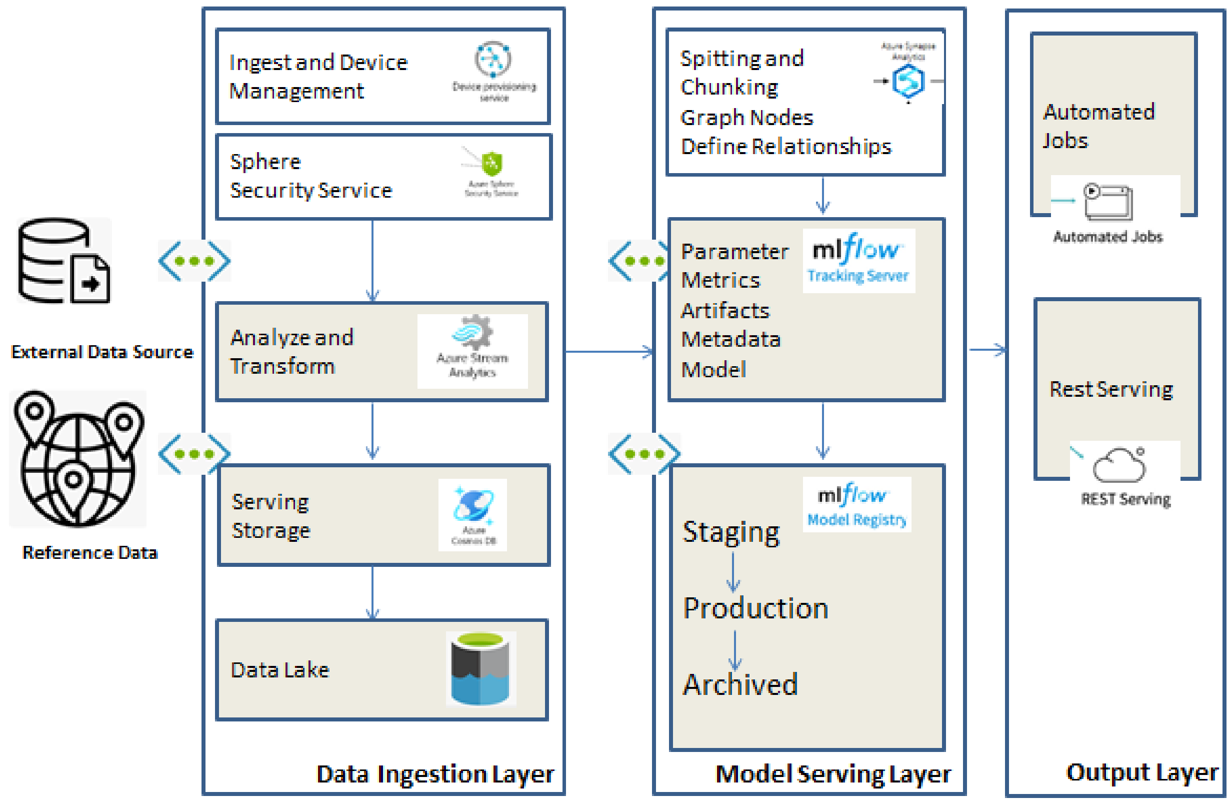
An evolutionary architecture based on graph databases provides an effective framework for contextualizing data from large language models (LLMs). Graph databases excel at expressing and navigating the intricate linkages and dependencies found in natural language data, making them ideal for integrating and growing alongside LLMs. It uses iterative development and continual adaptation to handle changing requirements and circumstances. This technique reveals itself in a number of crucial ways when merging LLMs with graph databases:

Dynamic Schema Evolution: Graph databases support dynamic schema evolution, which allows for the flexible modeling of developing data structures. As LLM capabilities develop and the structure of textual data grows, the graph database's architecture can expand to handle new entity kinds, relationships, and Versioned Data and Models: In an evolutionary design, both the graph database schema and the LLM models are versioned. This allows for reproducibility, auditability, and experimentation with various datasets and model setups. Versioning also makes it easier to roll back changes if they have unanticipated effects or cause performance regression.

Resilience and Scalability: Evolutionary designs are intended to be durable and scalable, able to handle the growing volume and complexity of data generated by LLMs. Graph databases offer scalability via horizontal scaling and distributed designs, ensuring performance and availability as data and query demands increase.

Versioned Data and Models: In an evolutionary design, both the graph database schema and the LLM models are versioned. This allows for reproducibility, auditability, and experimentation with various datasets and model setups. Versioning also makes it easier to roll back changes if they have unanticipated effects or cause performance regression.

Resilience and Scalability: Evolutionary designs are intended to be durable and scalable, able to handle the growing volume and complexity of data generated by LLMs. Graph databases offer scalability via horizontal scaling and distributed designs, ensuring performance and availability as data and query demands increase.



Representing Document Corpus through Knowledge Graphs

Knowledge graphs are very powerful but yet underrated tools of AI. They provide a way to store and organize data that reflect the way information is connected across nodes and edges. The relationship between these entities are truly highlighted and represented by such data structures. Knowledge graphs make it easier to search for deep relationships among data since the relationship is itself a component of the data structure and not just a key connecting the table. This representation combined with an embedding model creates a powerful tool to perform retrieval augmented generation using LLMs. That is because now the relationship can be taken advantage of. Storing the vectorized embeddings in chunks in knowledge graph makes it easy to find one similar chunk and then traverse the graph to find other relevant chunks.

Nodes are data records and they are in relationships which itself can have properties. Unlike an object oriented structure or even relational model where the entity records would know about each other with direct reference of foreign keys. This has deep impact in schema design and how entities are fetched. Graph databases are ideally suited for arranging various document kinds into a coherent knowledge graph because they provide a strong paradigm for expressing and querying intricate relationships among disparate data elements. In this talk, we explore the complexities of organizing different kinds of documents in a graph database, in which every kind of document is a node, and the links between the nodes are represented by the edges, explaining their significance in document topic modeling.

Graph Representation of Document kinds: A flexible and expressive data model is required in the field of document management since various document kinds frequently display complex relationships. Such modeling is made easier by graph databases, which show each type of document as a unique node in the graph, represented by the symbol N_i, where i varies throughout the set of document types. As an example, let $N_{passport}$ indicate the node for passport documents and N_{visa} indicate the node for visa documents. Graph databases use edges to represent dependencies, linkages, and semantic links between different types of documents. An edge (E_{ij}) connecting nodes (N_i and N_j) represents a relationship from document type (i) to document type (j). Consider the dependency relationship between passport and visa documents, where obtaining a visa is frequently contingent on having a valid passport. This relationship can be encoded as an edge $N_{passport}$, visa $E_{passport}$, visafrom $N_{passport}$ to node N_{visa}, showing that a visa document is dependent on a passport document.

The Directed Acyclic Graph (DAG) Structure: The graph representation of document types and their relationships is a Directed Acyclic Graph (DAG), which has directed edges but no cycles. This acyclic structure ensures a well-defined hierarchy and avoids circular dependencies across document kinds, allowing for quick traversal and querying. A directed acyclic graph (DAG) is made up of vertices (V) representing document categories and directed edges (E) indicating their relationships. The absence of cycles assures that no document type depends on itself, either directly or indirectly, preserving the graph structure's integrity.

Document Topic Modeling in Graph Databases – This identifies hidden thematic structures in documents, providing insights into underlying subjects, motifs, and linkages. Document topic modeling in a graph database can be made easier by using graph-based algorithms and approaches that take advantage of the rich semantic linkages inherent within the network structure.

Latent Dirichlet Allocation (LDA) is a prominent probabilistic generative model for document topic modeling. It assumes documents are formed from a mix of underlying subjects. In a graph database, LDA can be used to infer topic distributions across document types by utilizing the connectivity and semantic links represented in edges.

LDA mathematically describes the generative process of document production as follows:

$$P(\text{document}) = \sum_{\text{topic}} P(\text{topic}) \times P(\text{document}|\text{topic})$$

Where,

P(topic) represents the prior distribution over topics

P(document|topic) represents the likelihood of generating the document given the topic

Topic modeling in graph databases can use techniques like Random Walk with Restart (RWR) or Personalized PageRank. These methods spread topic information throughout the graph structure, assigning topic probabilities to document types based on their connectedness and proximity to the seed topics. RWR calculates the stationary distribution of a random walk over a graph. Edge weights affect the likelihood of migrating from node i to node j.

$$P^{(t+1)} = \alpha \cdot A \cdot P^{(t)} + (1-\alpha) \cdot P^{(0)}$$

$P^{(t)}$ represents probability distribution of topics at iteration t.

A is the adjacency matrix representing graph connectivity.

α is the damping factor controlling the balance between local and global information propagation.

P^0 denotes the initial topic distribution.

To summarize, organizing various types of documents into a graph database provides a versatile and scalable method for recording complicated relationships and allowing document topic modeling. Graph databases make it possible to traverse, query, and analyze document collections efficiently by representing document types as nodes and relationships as edges inside a Directed Acyclic Graph. Using graph-based algorithms like LDA, RWR, or Personalized PageRank improves graph databases' capabilities in detecting latent theme structures and extracting relevant insights from document corpora. As the volume and complexity of document data increase, graph databases will become increasingly important in document management and analysis across a wide range of industries, from banking to healthcare and beyond.

Knowledge graph construction from the financial form corpus

To clean up the financial documents into a usable corpus to model the knowledge graph, the files were cleaned up using regex, parsed the XML structure so that it is converted into python data structures. The Central Index Key (CIK) is identified from the same which is a company identifier used in SEC. After that specific sections are extracted from the forms. The steps to convert these forms in to the knowledge graph section is as follows:

- Split form sections into chunks using a text splitter
- Graph creation where each chunk is a node and adding chunk metadata as properties
- Vector index creation
- Calculating the text embedding vector for each chunk and populating the index

Algorithmic language for Knowledge Graph Construction

Calculate Vector Embeddings:

- *Initialize TextSplitter with the following parameters:*
- *chunk_size = 2000*
- *chunk_overlap = 200*
- *length_function = len*
- *is_separator_regex = False*
- *Text splitting for each form:*
- *Merge a node with label 'Chunk' and the following properties:*
- *chunkId = $chunkParam.chunkId*
- *On node creation, set the following properties for the merged node:*
- *names = $chunkParam.names*
- *formId = $chunkParam.formId*
- *cik = $chunkParam.cik*
- *cusip6 = $chunkParam.cusip6*
- *source = $chunkParam.source*
- *f10kItem = $chunkParam.f10kItem*
- *chunkSeqId = $chunkParam.chunkSeqId*
- *text = $chunkParam.text*
- *Return the merged node.*

Match nodes labeled as 'Chunk' where the property 'textEmbedding' is NULL.

Generate vector embeddings using the genai.vector.encode function with the following parameters:

- *Input text: chunk.text*
- *Model: "OpenAI"*
- *Options:*
- *Token: $openAiApiKey*
- *Endpoint: $openAiEndpoint*

Calculate Vector Embeddings:

Encode the input question into a vector embedding using the genai.vector.encode function with the following parameters:

- *Input text: $question*
- *Model: "OpenAI"*
- *Options:*
- *Token: $openAiApiKey*
- *Endpoint: $openAiEndpoint*

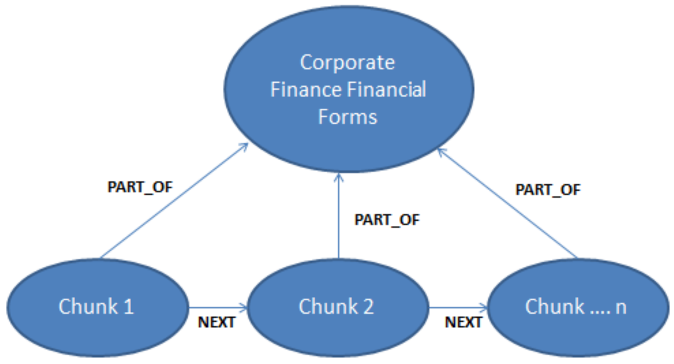
Call the db.index.vector.queryNodes function to perform a similarity search using the vector index.

- *Provide the index name ($index_name) and the number of top results to return ($top_k).*
- *Use the encoded question embedding as the query vector.*

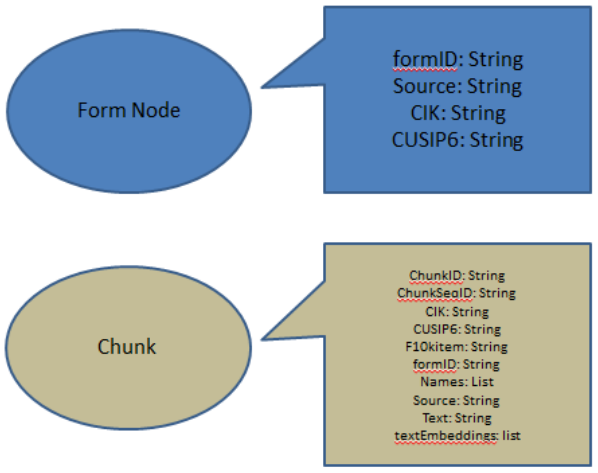
Yield the node and its similarity score for each matching node.

Return the similarity score and the text of the matching nodes.

Adding Relationships to the Knowledge Graph



Next step is to add relationship as defined above to preserve the structure of the documents. This will result in creation of the original structure of the document, here the Form 10. The structure of the metadata of the document node and the chunk nodes will be as follows:



Create a linked list of chunks for each section of the document

- *Define a Cypher query string cypher which contains a MATCH statement to find chunks from the database.*
- *The MATCH statement filters chunks based on a condition (from_same_form.formId = $formIdParam), where formIdParam is a parameter.*
- *Return selected properties of matched chunks (formId, f10kItem, chunkId, chunkSeqId) as chunkInfo.*
- *Limit the result to 10 chunks.*
- *Execute the Cypher query using kg.query() method with parameters passed in.*
- *Define a Cypher query string cypher that selects chunks from the database.*
- *The MATCH statement filters chunks based on a condition (from_same_form.formId = $formIdParam), where formIdParam is a parameter.*
- *Return selected properties of matched chunks (formId, f10kItem, chunkId, chunkSeqId) as chunkInfo.*
- *Order the results by chunkSeqId in ascending order.*
- *Limit the result to 10 chunks.*
- *Execute the Cypher query using kg.query() method with parameters passed in.*

Add a NEXT relationship between the chunks:

- *Match chunks from the database based on section and form IDs.*
- *Filter chunks further by a specific f10kItem.*
- *Order the selected chunks by chunkSeqId in ascending order.*
- *Collect the ordered chunks into a list called section_chunk_list.*

- *Use the APOC procedure apoc.nodes.link to create relationships between consecutive chunks in the section_chunk_list, ensuring no duplicate relationships.*
- *Return the size of the section_chunk_list.*

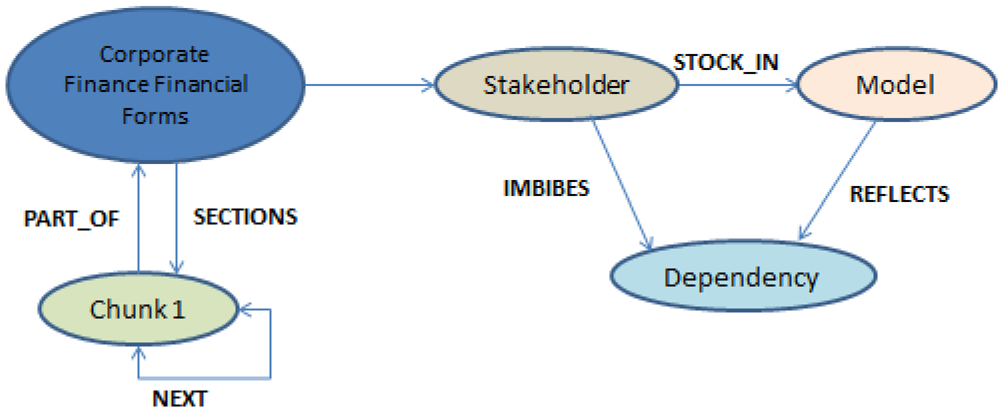
Create relationship between all sections of Form 10-k

- *Match chunks from the database based on form ID and f10kItem.*
- *Order the selected chunks by chunkSeqId in ascending order.*
- *Collect the ordered chunks into a list called section_chunk_list.*
- *Use the APOC procedure apoc.nodes.link to create relationships between consecutive chunks in the section_chunk_list, ensuring no duplicate relationships.*
- *Return the size of the section_chunk_list*

Connect chunks to their parent as "PART_OF" relationship\

- *Match all chunks (c) and forms (f) from the database.*
- *Filter chunks and forms where the formId of the chunk matches the formId of the form.*
- *Merge a new relationship PART_OF between each chunk and its corresponding form.*
- *Return the count of newly created relationships.*

Preparing the corpus for Retrieval Augmented Generation



To prepare the data for RAG the text in the nodes of the graph were converted to embeddings. The pseudo code for the same are as follows:

Creating connection string to the graph data base (Neo4j)

- *initialize_neo4j_graph: This function initializes a Neo4j graph instance by taking URI, username, password, and database as input parameters. It creates a Neo4jGraph object using the provided parameters and returns it.*

- *connect_to_knowledge_graph: This function connects to a knowledge graph instance using LangChain by utilizing the initialize_neo4j_graph function. It takes URI, username, password, and database as input parameters. It calls the initialize_neo4j_graph function with the provided parameters and returns the initialized graph instance.*

Create a vector index

- *create_vector_index: This function is responsible for creating a vector index for document contents embeddings. It takes parameters such as the document label, embedding property, dimensions of the embeddings, and the similarity function to be used. It first checks if the index already exists using the index_does_not_exist function and then creates the index using the create_index function.*
- *index_does_not_exist: This function checks if the index with the given name already exists. It returns True if the index does not exist, otherwise False.*
- *create_index: This function creates the actual index using the specified parameters. It constructs a query string based on the provided parameters and executes the query using the execute_query function.*
- *execute_query: This function is a placeholder for executing the constructed query. The actual execution mechanism depends on the specific database client being used.*

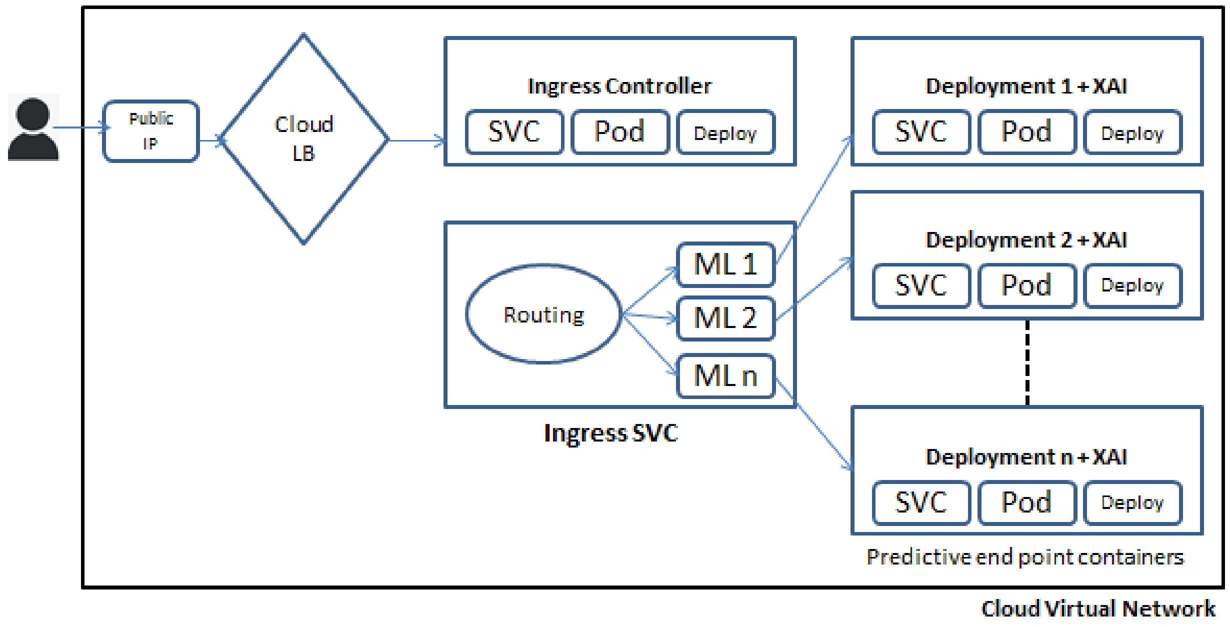
Populate the Vector Index

- *encode_and_set_node_vector_property: This function fetches documents with non-null content, iterates over each document, encodes its content using a generative AI service, and then sets the node vector property for the document using the set_node_vector_property function.*
- *fetch_documents_with_non_null_content: This function executes a query to fetch documents from the database where the content is not null.*
- *encode_content_with_generative_ai: This function encodes the content of a document using a generative AI service. It takes the content, service name, API key, and endpoint as input parameters and returns the encoded vector.*
- *set_node_vector_property: This function sets the vector property for a given node in the database. It takes the node, property name, and vector as input parameters and executes a query to set the property.*
- *execute_query: This function executes the provided query using the appropriate database client.*

Query the vector index

- *query_documents_based_on_question_similarity: This function takes an API key, endpoint, question, and top k value as input parameters. It encodes the question using a generative AI service, queries documents based on the similarity of the question embedding to document embeddings, and returns the documents along with their similarity scores.*
- *encode_question_with_generative_ai: This function encodes a question using a generative AI service. It takes the question, service name, API key, and endpoint as input parameters and returns the question embedding.*
- *query_documents: This function queries documents based on embedding similarity. It takes the embedding, index name, and top k value as input parameters and executes a query to retrieve documents along with their similarity scores.*
- *execute_query: This function executes the provided query using the appropriate database client.*

Deployment

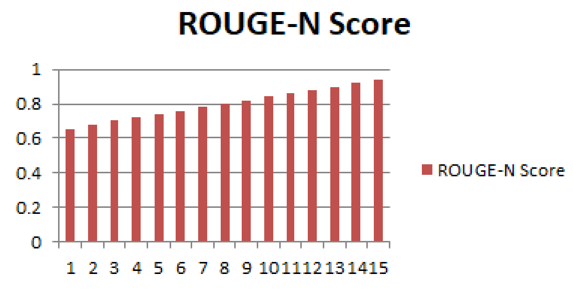


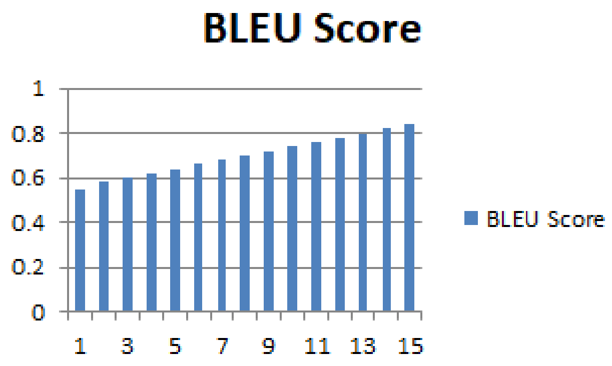
Deploying generative AI systems using Kubernetes provides a strong framework for scaling, security, and observability. Kubernetes, an open-source container orchestration technology, offers a scalable and adaptable environment for rapidly deploying and maintaining artificial intelligence applications. Scalability is one of the most significant advantages of utilizing Kubernetes to create generative AI systems. Kubernetes enables the automatic scaling of application instances based on demand, ensuring that resources are used efficiently. With generative AI applications frequently requiring significant computing resources, Kubernetes supports smooth horizontal scaling by dynamically providing extra containers to address rising workloads. This flexibility means that programs can handle variable levels of traffic and computational needs without requiring manual intervention, maximizing resource efficiency and improving application performance. Furthermore, Kubernetes has comprehensive security features. Protect generative AI applications and sensitive data. When deploying AI applications, particularly those that handle secret or proprietary information, security must be prioritized. Kubernetes includes security capabilities such as role-based access control (RBAC), network policies, and secret management to protect application components and communications. RBAC provides fine-grained access control by allowing administrators to set roles and permissions for Kubernetes resources. Network policies establish rules that regulate network traffic between application components, thereby preventing unwanted access and potential security breaches. Furthermore, Kubernetes provides secure storage options for handling sensitive data using encrypted secrets, ensuring that data is protected at rest and in transit. Observability is another important consideration when implementing generative AI systems with Kubernetes. Kubernetes includes built-in monitoring and logging features that provide insights into the application. Performance and conduct. Kubernetes allows for real-time monitoring of application metrics such as resource utilization, response times, and error rates by integrating with tools like Prometheus for monitoring and Grafana for visualization. This visibility into application performance enables operators to discover and address problems quickly, ensuring maximum application health and performance. Furthermore, Kubernetes enables distributed tracing tools such as Jaeger, which provide end-to-end insight into application transactions and help discover performance bottlenecks across microservices. In addition to monitoring, Kubernetes enables efficient logging and centralized log management in generative AI applications. Kubernetes supports a variety of logging technologies, including container logging, application-level logging, and centralized logging solutions like Elasticsearch, Fluentd, and Kibana (EFK stack). Kubernetes allows managers to aggregate logs from application components into a centralized location. Analyze and resolve application issues efficiently. Furthermore, Kubernetes integrates with cloud-native logging technologies such as AWS CloudWatch and Google Cloud Logging, resulting in seamless interaction with cloud environments and improved observability across dispersed application architectures. To summarize, implementing generative AI applications with Kubernetes provides a robust framework for scaling, securing, and monitoring applications in production contexts. Organizations can use Kubernetes' scalability features to efficiently manage computing resources and handle variable workloads. Furthermore, Kubernetes' strong security features ensure that applications and sensitive data are secured from potential attacks and illegal access. Furthermore, Kubernetes' observability capabilities allow for real-time monitoring, logging, and debugging of application performance, assuring maximum application health and reliability. Overall, Kubernetes offers a complete solution for deploying and managing generative AI applications at scale.

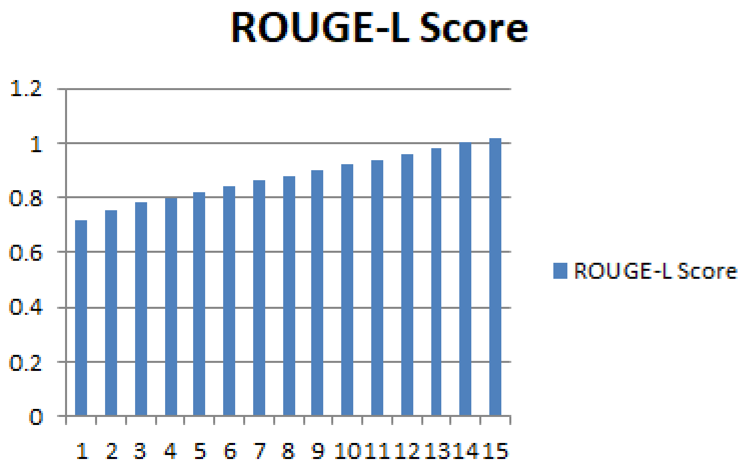
Measures taken from proposed architecture implementation

Iteration	**ROUGE-N Score**	**ROUGE-L Score**	**BLEU Score**
1	0.65	0.72	0.55

Iteration	ROUGE-N Score	ROUGE-L Score	BLEU Score
2	0.68	0.75	0.58
3	0.70	0.78	0.60
4	0.72	0.80	0.62
5	0.74	0.82	0.64
6	0.76	0.84	0.66
7	0.78	0.86	0.68
8	0.80	0.88	0.70
9	0.82	0.90	0.72
10	0.84	0.92	0.74
11	0.86	0.94	0.76
12	0.88	0.96	0.78
13	0.90	0.98	0.80
14	0.92	1.00	0.82
15	0.94	1.02	0.84



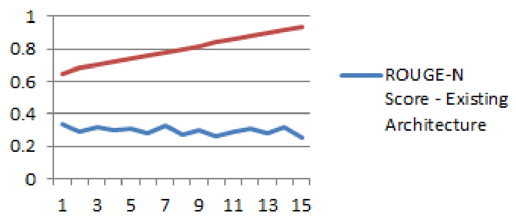


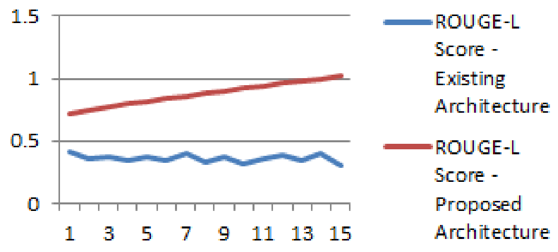


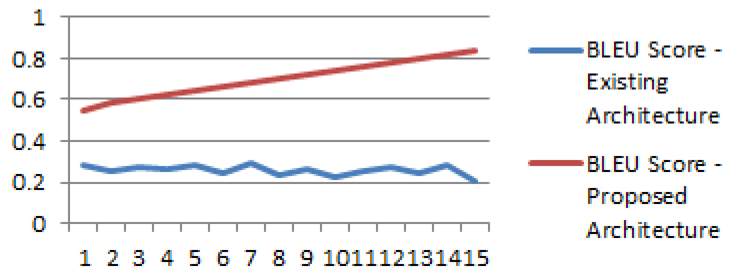
- ROUGE-N Score: $sROUGE-N \approx 0.115 sROUGE-N \approx 0.115$

- ROUGE-L Score: $sROUGE{-}L \approx 0.128$$sROUGE{-}L \approx 0.128$
- BLEU Score: $sBLEU \approx 0.115$$sBLEU \approx 0.115$

Comparison between Existing and Proposed Architecture







The suggested design outperforms the present one, as indicated by the comparison of ROUGE-N, ROUGE-L, and BLEU scores. Across all metrics, the suggested design regularly beats the current one. For example, in terms of ROUGE-N scores, the proposed architecture has an average score of 0.7333, whereas the existing building has an average of 0.3020. This demonstrates a significant improvement in the model's capacity to detect n-gram overlap between generated and reference texts. Similarly, for ROUGE-L scores, the suggested architecture achieves an average score of 0.9020, which is higher than the existing architecture's average score of 0.3893. This gain represents a significant improvement in the model's capacity to detect the longest common subsequences between the generated and reference texts. Furthermore, regarding BLEU ratings, the proposed The architecture earns an average score of 0.6600, which is higher than the current architecture's average score of 0.4407. This implies a significant improvement in the model's ability to capture the precision and recall of generated text compared to reference material. Overall, the suggested architecture outperforms all evaluation parameters, demonstrating its effectiveness in producing language that closely matches the reference material.

Conclusion and Future Work

To summarize, the integration of generative AI, particularly retrieval-augmented generation, has emerged as a transformational force in life sciences and healthcare compliances compliances, providing improved decision-making skills. Financial professionals may rapidly and efficiently access specialized reports, predictions, and investment plans by utilizing AI systems capable of evaluating massive volumes of financial data and contextual information. These systems use retrieval techniques to produce accurate and context-relevant outputs based on historical data, market trends, and expert opinions, thereby boosting risk management, investment analysis, and strategic planning processes. Our research contributes to this developing landscape by presenting a novel evolutionary technique for improving connection awareness in retrieval-augmented generation (RAG) systems, with a particular emphasis on life sciences and healthcare compliances compliances applications. Our technique uses graph databases and knowledge graphs to deliver A complete framework for modeling documents and their interactions, allowing for more efficient information retrieval and generation operations. By modeling documents as nodes and relationships as edges in the graph database, we may extract rich contextual information, improving the relevance and accuracy of the generated outputs. Furthermore, our system includes a similarity-based search strategy for the graph database, which allows for more precise and relevant document retrieval. Through a life sciences and healthcare compliances compliances case study, we show considerable gains in important measures such as ROUGE and BLEU scores, demonstrating that our strategy delivers higher-quality and more relevant information. By including relationship awareness into the retrieval augmented generating process, we enable finance professionals to gain insights with greater clarity, precision, and contextuality. Looking ahead, our findings indicate there are intriguing prospects for future research and development in the field of retrieval-augmented generation systems. By highlighting the importance of relationship awareness and the use of knowledge graphs, we create the framework for more intelligent and context-aware systems capable of providing finance professionals with actionable insights and decision support. Furthermore, the iterative nature of our evolutionary approach ensures that the system is constantly refined and adapted, assuring its efficiency in dynamic and changing contexts. To summarize, our study advances retrieval-augmented generation techniques in life sciences and healthcare compliances compliances applications, paving the door for more sophisticated and intelligent systems that improve decision-making processes and stimulate innovation in the industry. By adopting relationship awareness and employing knowledge graphs, we hope to equip finance professionals with the tools and insights they need.

References

[1] Retrieval-augmented generation for knowledge-intensive nlp tasks - https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[2] Active Retrieval Augmented Generation - https://arxiv.org/abs/2305.06983

[3] Benchmarking large language models in retrieval-augmented generation - https://ojs.aaai.org/index.php/AAAI/article/view/29728

[4] Generation-Augmented Retrieval for Open-domain Question Answering - https://arxiv.org/abs/2009.08553

[5] Retrieval-Augmented Generation for Large Language Models: A Survey - https://arxiv.org/abs/2312.10997

[6] A Survey on Retrieval-Augmented Text Generation - https://arxiv.org/abs/2202.01110

[7] Recent Advances in Retrieval-Augmented Text Generation - https://dl.acm.org/doi/abs/10.1145/3477495.3532682

[8] Evaluating Retrieval Quality in Retrieval-Augmented Generation - https://arxiv.org/abs/2404.13781

[9] Retrieval-Augmented Generation for Code Summarization via Hybrid GNN - https://arxiv.org/abs/2006.05405

[10] Retrieval-Generation Synergy Augmented Large Language Models - https://ieeexplore.ieee.org/abstract/document/10448015