INTERNSHIP REPORT

The Future of GenAI is Agentic

HSBC - Paris
2024

Abdessamad EL KABID

ÉCOLE POLYTECHNIQUE

INSTITUT POLYTECHNIQUE DE PARIS

HSBC

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my family for their unwavering support and constant encouragement. They remain, without question, one of the key pillars of my success.

I would also like to extend my heartfelt thanks to the ADA team for their warm welcome during my internship. I am particularly grateful to Mr. BENCHEKROUN for the insightful discussions we shared, and to Mr. Antoine BALL, my supervisor, for his attentive guidance, availability, and invaluable advice, all of which helped me achieve my goals. My sincere thanks also go to Mr. Mo REZKI for his mentorship, thoughtful recommendations, enriching interactions, and the invaluable knowledge he imparted to me.

I would also like to acknowledge the entire Finance-Analytics and ADA engineering teams, whose advice was instrumental in shaping my future decisions and making this internship a truly rewarding and memorable experience.

EXECUTIVE SUMMARY

My internship at HSBC marked a pivotal moment in my professional journey. I have developed a solid foundation and gained deep insights into the professional world. These prior experiences allowed me to approach this internship with an enriched perspective, further deepening my understanding of the field.

My primary goal was to enhance my skills in mathematics, algorithms, and machine learning, while observing their practical application in a real-world environment. The opportunity provided by the ADA team perfectly aligned with this ambition. My work involved reviewing specialized articles and implementing new methods within an internal application. The field of machine learning and natural language processing, which has always intrigued me, became an exciting area of exploration.

When I joined École Polytechnique, I was already drawn to these emerging subjects, but it was only through various projects and two years of study that I began to master the underlying mathematical theory. This internship significantly strengthened my expertise in deep learning, both in theory and practice.

Aiming for excellence in my career, I saw this internship as a unique opportunity to gain new and enriching experience. I was fortunate to work in an environment where engineers, programmers, traders, and quantitative researchers collaborated closely. I particularly appreciated the team spirit, camaraderie, and the friendly atmosphere that permeated the office. The team I worked with was international, with my supervisor based in Paris and other members in Hong Kong and London.

This experience allowed me to understand the connection between research and the corporate world, and to realize that my professional aspirations go beyond financial rewards or social recognition. It is essential to set goals and principles that align with the values of the company. I also gained firsthand experience in two crucial areas: the dynamics of relationships between colleagues in a research context, and the balance between rigor and comfort. Finally, this internship enabled me to critically evaluate the workings of a large organization and to recognize the importance of certain skills that I had previously underestimated.

CONTENTS

1 INTRODUCTION

As the saying goes, 'Practice makes perfect.' Indeed, it is through hands-on experience and perseverance that we become more proficient. Learning from the advice of more experienced individuals is also essential to pushing oneself further. My time at HSBC as an AI engineer reaffirmed this belief.

This field holds a particular appeal for me as it represents a fusion of mathematics and computer science—two disciplines I find both promising and fascinating. I was part of a team that allowed me considerable autonomy, encouraged research and the reading of academic articles, and gave me the opportunity to present my findings and methods to both a large audience and the digital team.

Even before my first day, my supervisor welcomed me warmly and answered all my questions.

This internship was not only a deep dive into the technical aspects of machine learning, including the structuring and programming of a large-scale project or application, but also an immersion into project management and the challenges and benefits of different team management styles from a technical expert's perspective. Beyond these technical aspects, it was also an opportunity to develop my interpersonal skills—fostering relationships, interacting with colleagues—and gaining valuable insights into the corporate world, which will undoubtedly be an asset to my future career.

I had the chance to refine my research skills, gain a deeper understanding of the challenges involved in applying theoretically sound ideas, and experience the complexity of explaining these concepts and convincing stakeholders of the potential of promising projects. Overall, it was an enriching experience within a major organization.

2
LLMS AND RAG

2.1 What is a Large Language Model

Large Language Models (LLMs) are a class of deep learning architectures based on *transformer networks*. A transformer model is a type of neural network that learns context and meaning by modeling relationships between elements in sequential data, such as words in a sentence. It uses *self-attention* mechanisms to capture dependencies between words, regardless of their position in the sequence.

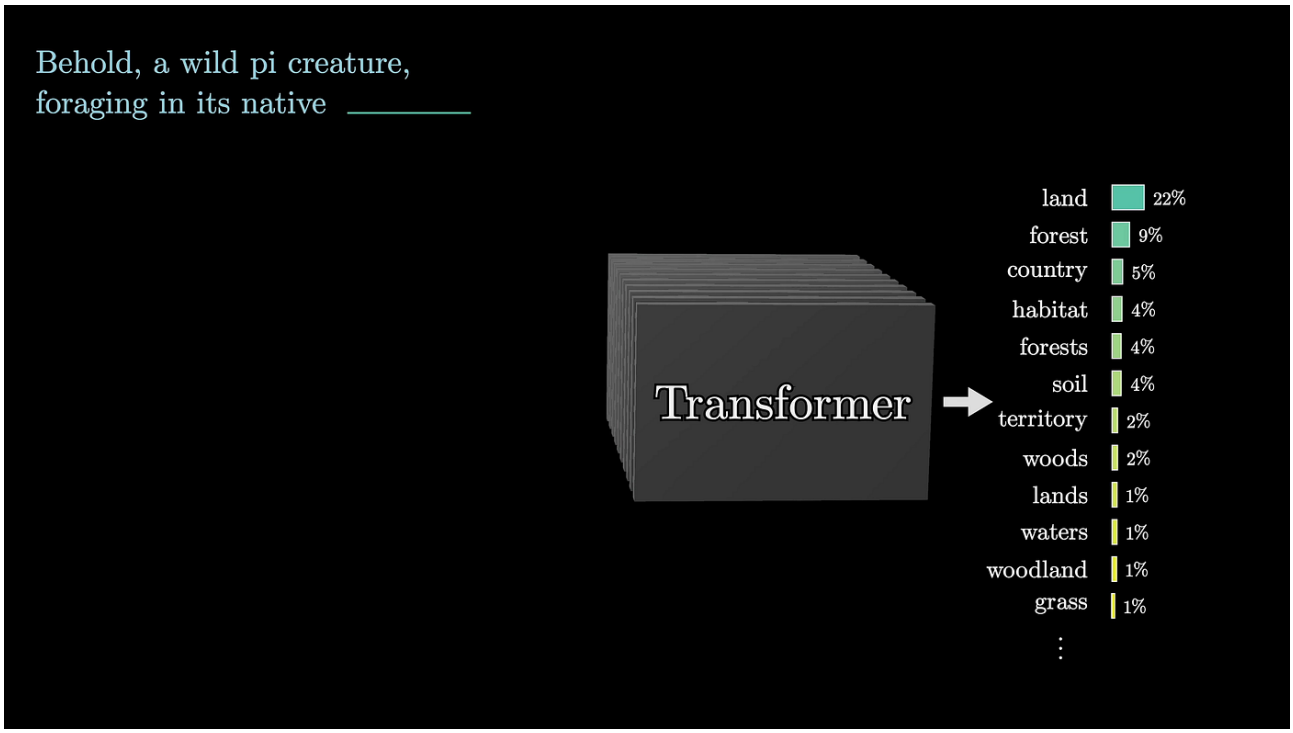

Figure 1: Predicting the probability of the next word

LLMs are designed to understand and generate human-like text by predicting the probability distribution over the next word in a sequence. By sampling from this probability distribution, the model can select the next word and continue generating coherent text.

2.2 Retrieval Augmented Generation

What if we want to use large language models on data they weren't trained on, such as confidential or private data? One way to achieve this is through **Retrieval-Augmented Generation** (RAG). RAG optimizes the output of an LLM by incorporating an external knowledge base, allowing the model to reference data outside of its training sources before generating a response.

LLMs are trained on vast amounts of data and leverage billions of parameters to perform tasks. RAG extends the capabilities of LLMs to *specific domains* or an organization's *internal knowledge* base without the need for retraining the model. This approach is cost-effective and ensures that LLM outputs remain relevant, accurate, and useful across different contexts, even when the required knowledge is not part of the model's original training data.

How it works:

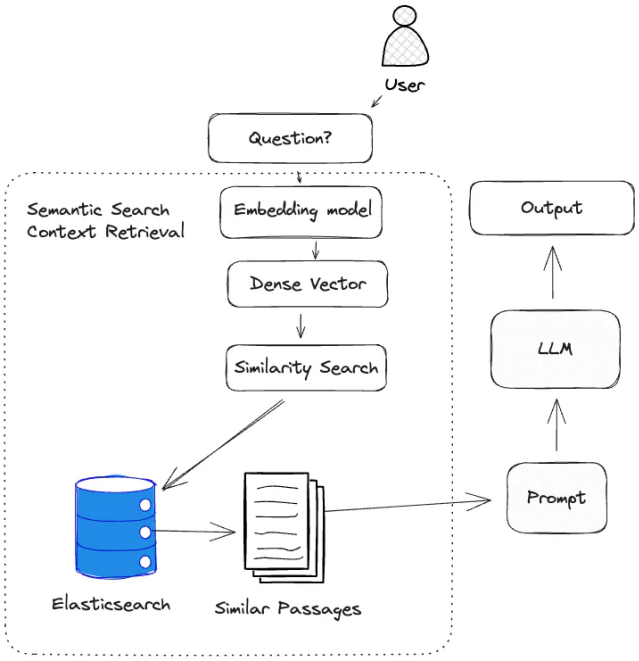

Figure 2: RAG pipeline

2.2.1 • Data Ingestion

- **Document Collection**: The first step involves collecting a large corpus of documents or knowledge sources. These sources can include text from internal databases, websites, or other relevant repositories.
- **Indexing**: The collected documents are indexed to facilitate efficient retrieval. Indexing methods include:
  - **Inverted Index**: Maps terms to their locations in the documents.
  - **Vector Embeddings**: Converts documents into dense vectors using models such as BERT or Sentence-BERT.
- **Embedding Generation**: Documents are transformed into vector embeddings, which capture semantic relationships and contextual information, and stored in a vector store. This step involves using embedding models to represent documents in a high-dimensional space. Directions in this space represent concepts and near vectors can have similar meanings or belong to the same category.

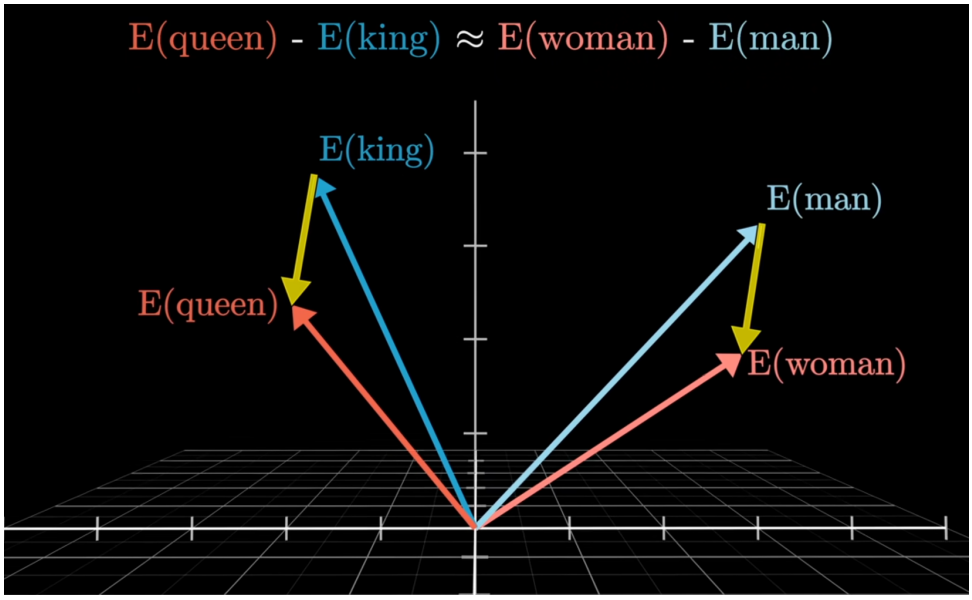

Figure 3: Example of how directions encode concepts

2.2.2 • Retrieval Phase

- **Query Embedding**: The input query is converted into a vector embedding using the same model as used for document embeddings.
- **Search Mechanism**: The query embedding is used to search the indexed documents. Retrieval methods include *Dense Retrieval*: Calculating similarity scores between the query embedding and document embeddings using methods such as cosine similarity.
- **Retrieval**: A set of top-N relevant documents or passages is retrieved based on similarity scores. These documents provide context or factual information relevant to the query.
- **Context Augmentation**: The retrieved documents are combined with the original query to form an augmented context. This context is then preprocessed to ensure it fits the input constraints of the generation model.

2.2.3 • Generation Phase

- **Input to the Generator**: The augmented context (query + retrieved documents) is fed into a generative model.
- **Attention Mechanism**: The generation model uses its attention mechanisms to integrate information from the query and retrieved documents, generating a contextually relevant response.
- **Generation**: The model generates text based on the augmented context, predicting the next words or sentences to produce a coherent and accurate response.

2.3 VectorStore

A vector store manages and queries data vectors. Vector stores allow for semantic search, making it easier to retrieve documents that are closely related to the query's intent rather than just matching exact words.

- **Vectorization**: Queries and documents are transformed into vectors using LLM embeddings.
- **Indexing**: Approximate Nearest Neighbor (ANN) algorithms are used for fast, efficient similarity searches.
- **Querying**: Allows the system to retrieve semantically similar documents based on the input vector.

3 PROJECT OVERVIEW

During the initial weeks of the project, my primary focus was on setting up the working environment and acquiring a comprehensive understanding of the tools, frameworks, and concepts that would be pivotal for my contributions. This involved securing the necessary system accesses on *ServiceNow*, configuring my development environment, and ensuring that my laptop was properly equipped to handle the project's technical demands. Concurrently, I dedicated time to ***reviewing academic papers*** on agentic workflows and specialized Large Language Models (LLMs). The goal was to understand how multi-agent systems could be effectively integrated with *Alice* -The chatbot i worked on- to deliver sophisticated, task-specific features that require high levels of reasoning and adaptability.
Experiments have demonstrated the power of agentic workflows and how an agentic pipeline can enhance the capabilities of any reasonably strong model:

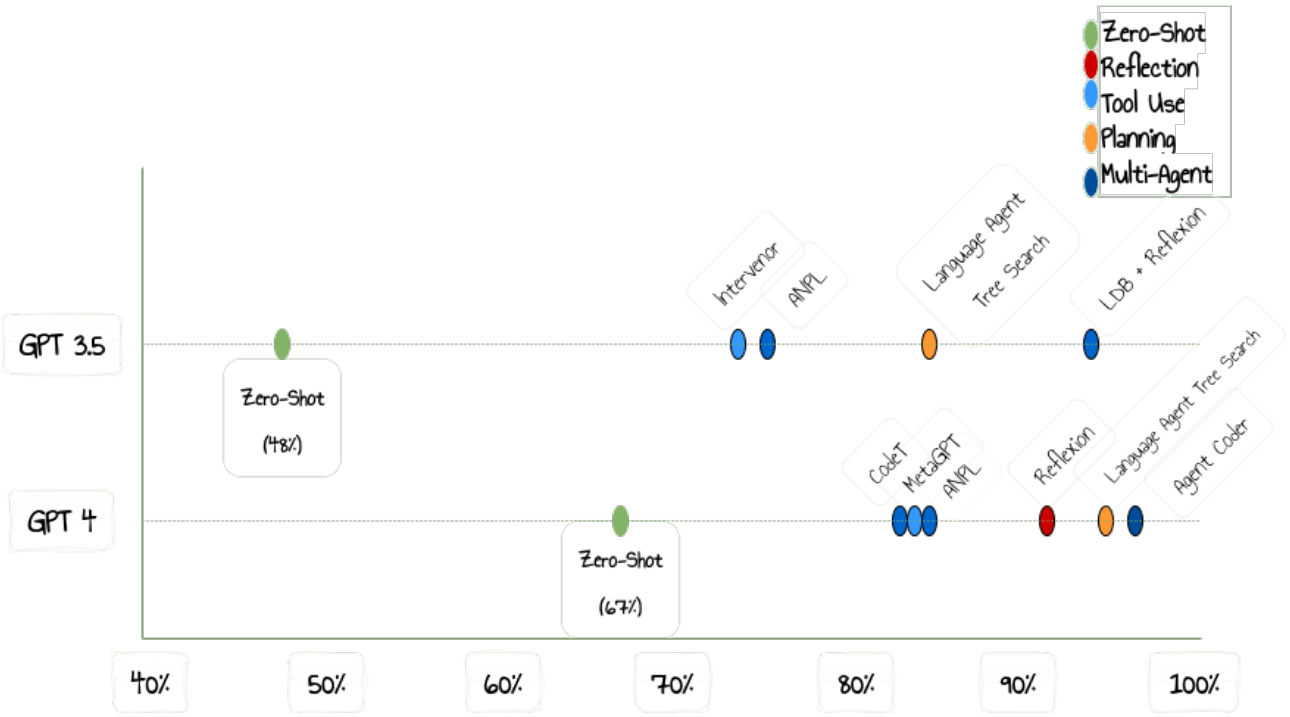

Figure 4: Power of agents

Additionally, I spent time reviewing the codebase to understand the architecture and approaches implemented by the team. This provided insight into the existing system and helped identify areas for improvement, particularly in the context of multi-agent workflows. Once my research phase was completed, I began implementing an action plan in collaboration with Mo to integrate new features and improvements.

The primary task was to design and implement a workflow that enables multiple agents to collaborate on generating, evaluating, and refining LLM outputs. I decided to implement a reflection method where one agent generates a draft response, and another agent critiques and suggests improvements. This creates a feedback loop where agents iteratively refine outputs to enhance the quality, accuracy, and reliability of the final result.

For this project, I chose to work with LangGraph—a powerful library that supports the creation of complex, stateful applications using multiple AI agents powered by LLMs. LangGraph offers a flexible framework to coordinate and execute tasks across agents, with robust features such as persistence, human-in-the-loop intervention, and streaming support.

3.1 LangGraph Overview

LangGraph provides a structured environment to manage interactions between multiple agents, ensuring smooth state transitions and dynamic control flows.

- **State Management**: The state is a central component in LangGraph, representing the real-time status of the application. It is continuously updated as agents complete their tasks, ensuring the system remains responsive to new inputs and maintains the context required for ongoing operations. Each time the graph is executed, the state is passed between nodes, allowing for adaptive and context-aware decision-making as the workflow progresses.
- **Nodes**: These are the core units of computation within LangGraph. Each node can represent a task performed by an agent, such as a computation or an LLM-based action. Nodes act as independent points in the system, and upon completing their task, they pass messages to other nodes, forming a chain of interactions. Nodes can be customized, allowing them to execute LLM-based operations or custom Python code.
- **Edges**: Edges define the flow of data between nodes, allowing for dynamic control flows. By leveraging edges, developers can introduce conditional paths and cycles within the graph, making it possible to build sophisticated workflows that handle complex decision-making processes or repetitive tasks.

Key Features:

- **Cycles and Branching**: Supports loops and conditionals for advanced decision-making.
- **Persistence**: Error recovery, human oversight, and resumption of tasks are built-in.
- **Human-in-the-Loop**: Human intervention is allowed at critical points in the workflow, enhancing control over agent behaviors.
- **Streaming Support**: LangGraph can stream token-by-token or intermediate step results for a dynamic, real-time user experience.

4
GRAPH DESIGN AND IMPLEMENTATION

To design the workflow, I began by sketching out the multi-agent interaction graph on paper. This allowed me to visualize the flow of data and tasks between agents before translating the design into code. The implementation required aligning my approach with the existing codebase while ensuring that the new features could integrate seamlessly with the front-end system. This phase of the project involved overcoming challenges related to understanding the codebase and ensuring consistency with the broader system architecture.

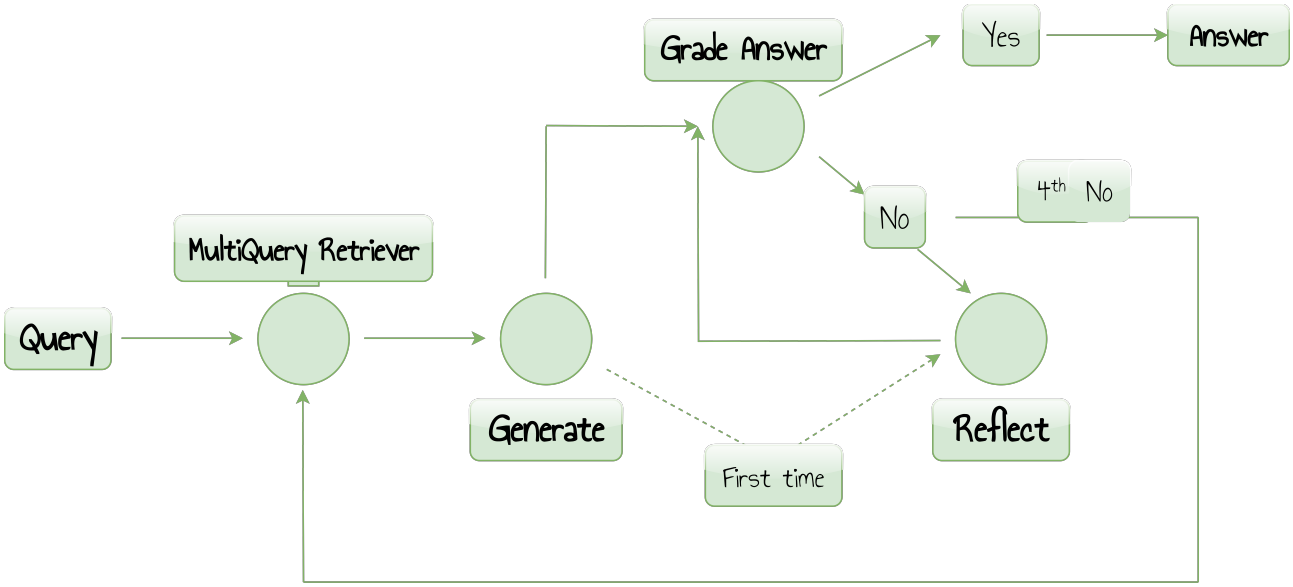

Figure 5:

4.1 State of the Graph

The state in my workflow contains the following elements:

- **Query**: The user's original query or task.
- **Generator's Answer**: The output from the Chain of Thought (CoT) Generator agent.
- **Critic Agent's Answer**: The Reflection Agent's response to the generator's answer.
- **Index Name**: The name of the vector index for document retrieval.
- **Memory**: The chat history, used to maintain context across multiple interactions.
- **Counter**: Tracks the number of interactions to control the graph's flow.
- **List of Thoughts**: A structured record of interactions between agents, capturing key decisions and actions.

4.2 Nodes

I designed a series of agents to form the backbone of the workflow:

- **Agent Parent Class**: A base class containing common functionality for all agents.
- **CoTGenerator**: Uses **Chain of Thought** (CoT) reasoning to generate structured responses.
- **Reflection Agent**: Reviews and critiques the generator's output, suggesting improvements.
- **Multi-query Retriever**: Expands on a user's query by generating multiple related questions to improve document retrieval accuracy.

- **Answer Grader**: Validates the reflection suggestions and final outputs to ensure alignment with the initial query and context.

Additionally, I initially experimented with a ***Hallucination Grader*** node, which aimed to detect inaccuracies in the LLM's outputs. However, it proved inefficient and time-consuming, so I opted to remove it from the workflow.

4.3 Node-specific Details

4.3.1 • CoT Generator

Breaks down the query and retrieved documents into logical steps, using a *Chain of Thought* prompting technique to produce intermediate reasoning steps, resulting in more accurate and interpretable responses.

4.3.2 • Reflection Agent

This agent evaluates the CoT generator's output and suggests possible improvements, creating a reflective loop that ensures the generated response is continuously refined.

4.3.3 • Answer Grader

This agent evaluates whether the final answer meets the required criteria by using a restricted output format. This restriction helps prevent hallucinations and enforces a structured response. The agent provides answers in a **yes or no** format based on the reflection, context, and query. Restricting the output format is necessary because large language models (LLMs) can sometimes generate hallucinations or responses in an unintended format.

LangChain offers a method called `with_structured_output()` for generating structured outputs from LLMs. This function simplifies the process of obtaining structured data, which is crucial for many applications.

- **Purpose**: The main goal of structured output is to ensure that LLMs return data in a predefined format, such as JSON or other schema-based formats. This is essential for integrating LLMs into applications where structured data is needed for further processing or analysis.
- **Implementation**: The `with_structured_output()` method is designed for models that support native APIs for structuring outputs. It allows developers to define a schema that specifies the names, types, and descriptions of the desired output attributes. This schema can be specified using a `TypedDict` class, JSON Schema, or a Pydantic class.

4.4 Multi-query Retriever

This node takes the user's input and generates multiple related questions to broaden the search scope. The top N relevant documents are selected from the vector store based on their similarity scores. This method improves retrieval precision by filtering and ranking the documents, especially when working with models that have a limited context length.

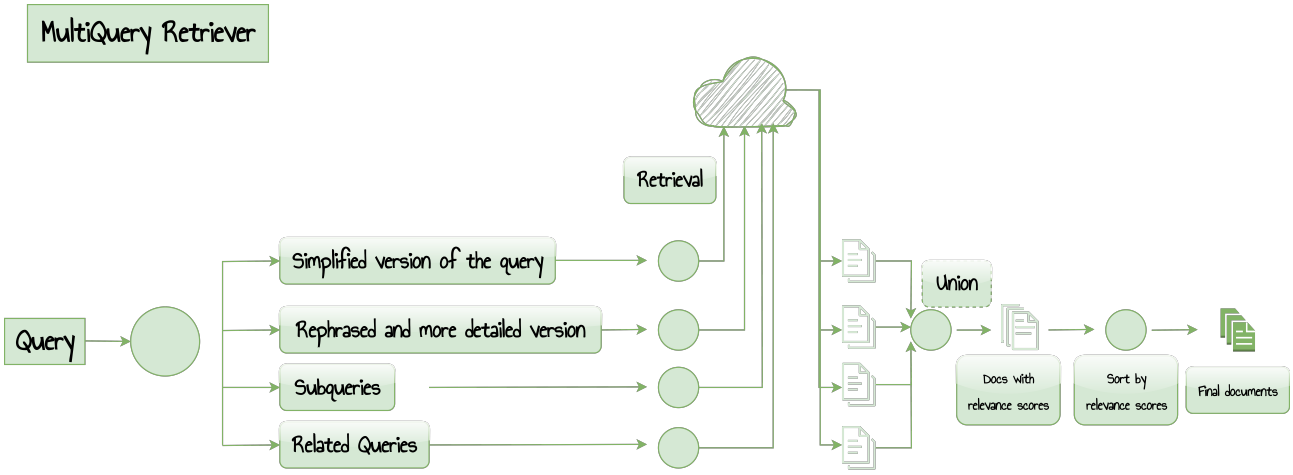

Figure 6: Multi-query retriever

4.5 Chat History

The chat history is designed to track user inputs and outputs but excludes internal agent interactions. I implemented Three versions:

- **Full Message History**: Captures all interactions in full detail. (`ConversationBufferMemory` class)
- **Partially Summarized History**: Condenses the older parts of the conversation to save space and provide a concise context.(`ConversationSummaryBufferMemory` class)
- **Summarized History**: Condenses the conversation.(`ConversationSummaryMemory` class)

5 FUTURE WORK

- **Tool Integration**: I explored using external tools such as code interpreters to allow agents to execute code or plot graphs. For security and scalability, we will need to run the code inside Docker containers.

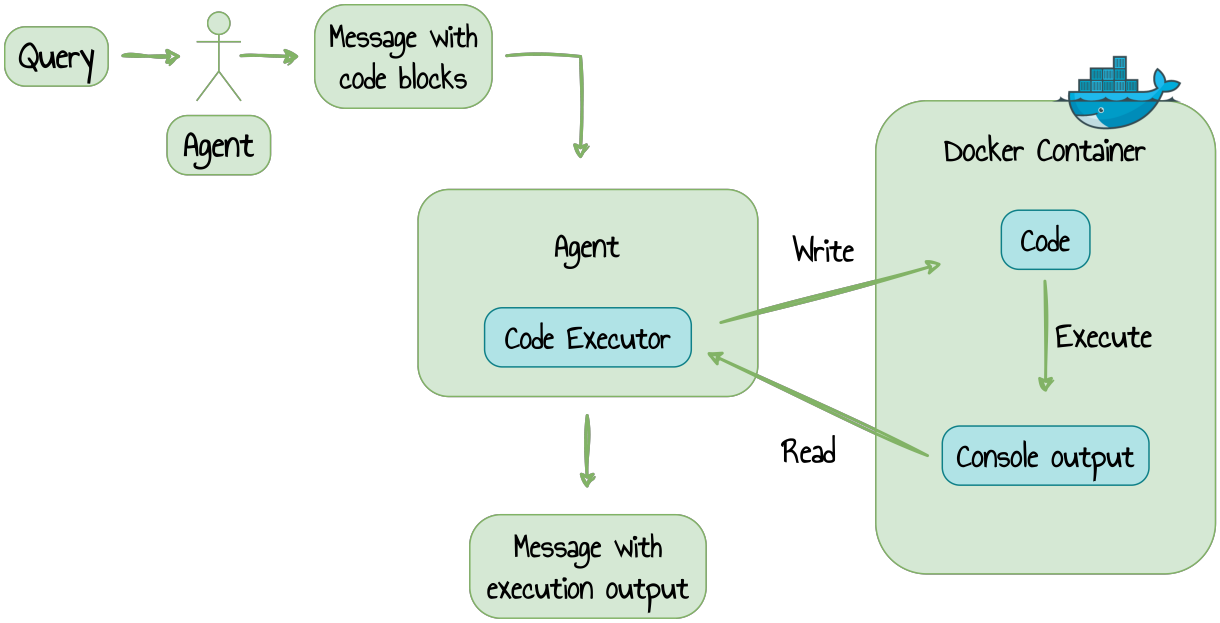

Figure 7:

- **Optimizing Multi-agent Collaboration**: There are ongoing efforts to improve how multiple agents work together. I plan to further explore toolkits like `autogen` or additional LangChain capabilities to streamline agent workflows.
- **Using Knowledge Graphs**: For better explainability. Works with structured data.

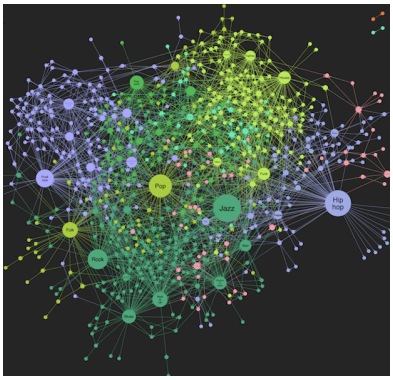

Figure 8: Knowledge graph

6 CONCLUSION

During this internship, I gained a deep understanding of Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG), and explored how these technologies can be used to enhance the performance and relevance of AI systems. By integrating external knowledge sources, we were able to improve how models handle specialized or confidential data.

A key focus of the project was developing a multi-agent workflow using LangGraph, which allowed us to build a system where multiple agents collaborate to generate, critique, and refine responses. This approach demonstrated how combining various agent capabilities can leverage the strengths of strong models while addressing their limitations.

Looking ahead, there are exciting opportunities for further improvement. Future work will involve integrating external tools to expand agent capabilities, optimizing how multiple agents collaborate, and exploring additional features of LangChain to refine workflows and boost system performance.

Overall, this internship provided valuable insights into applying advanced LLM techniques and agentic workflows, and highlighted how these technologies can be used to create more intelligent and adaptable systems.

REFERENCES

[1] Autogen: *https://arxiv.org/pdf/2308.08155*

[2] ChatDev: *https://arxiv.org/pdf/2307.07924*

[3] The Prompt Report: *https://arxiv.org/pdf/2406.06608*

[4] Self-RAG: *https://arxiv.org/pdf/2310.11511*

[5] Gorilla: Large Language Model Connected with Massive APIs : *https://arxiv.org/abs/2305.15334*

[6] Chain-of-Thought Prompting Elicits Reasoning in Large Language Models: *https://arxiv.org/abs/2201.11903*

[7] Introduction to neural networks:: *But what is a Neural Network?*