Chapter 1

Introduction

In recent years, the field of natural language generation (NLP) [15] has experienced remarkable progress, largely due to the advancements in large language models (LLMs) [13]. These models, such as GPT-4 [1], have revolutionized the way machines understand and generate human language. They have shown exceptional capabilities in producing coherent, contextually relevant text across a variety of applications, from answering questions to creative writing and conversational agents. However, as these models have grown in sophistication, they have also revealed inherent limitations, particularly in their ability to provide accurate, up-to-date information and to handle specific queries that extend beyond the data they were trained on.

Traditional LLMs are primarily trained on vast corpora of text, which encapsulate the knowledge available up to a certain point in time. This static nature of training data poses a significant challenge when these models are expected to respond to queries that require current or highly specific information. For instance, in applications such as virtual assistance, customer support, and decision-making, the need for accurate, timely responses is paramount. Yet, LLMs, by design, are limited to the knowledge they acquired during their training phase and cannot independently access or retrieve new information post-training. This limitation becomes particularly problematic when dealing with dynamic or niche domains where the availability of real-time data is critical.

To address these challenges, the research community has turned to innovative approaches that combine the generative power of LLMs with information retrieval techniques. One of the most promising developments in this area is the Retrieval-Augmented Generation (RAG) framework [16]. RAG integrates LLMs with retrieval mechanisms that allow the model to access external databases or knowledge sources at the time of query processing. By doing so, it enhances the model's ability to generate responses that are both contextually relevant and informed by the most current data available.

Despite the improvements introduced by RAG, the complexity of integrating retrieval systems with generative models presents several challenges. The effectiveness of a RAG model depends heavily on the efficiency of the retrieval process, the relevance of the retrieved information, and the seamless integration of this information with the LLM's generative capabilities. In response to these challenges, an advanced variant of RAG, known as Multi-Retrieval-Augmented Generation

(Multi-RAG) [17], has been developed. Multi-RAG enhances the retrieval process by employing multiple retrieval mechanisms, thereby increasing the diversity and relevance of the information accessed during generation.

However, the deployment of these models in practical applications is not without its difficulties. Implementing and fine-tuning RAG and Multi-RAG models requires sophisticated infrastructure, including robust vector databases and development environments tailored to handle the integration of retrieval and generation processes. Furthermore, the need to monitor and optimize these systems for cost, performance, and accuracy adds another layer of complexity. Technologies like ChromaDB [12], a vector database, are essential for managing and retrieving large-scale embeddings, while environments such as LangFlow [5] facilitate the development and orchestration of these models. Additionally, tools like LangWatch [7] are critical for tracking metrics, visualizing costs, and debugging, ensuring that the models perform optimally in real-world scenarios.

Given the current landscape, this thesis seeks to explore and benchmark the performance of different approaches to natural language generation, specifically focusing on the comparison between a traditional LLM, a RAG model, and the Multi-RAG variant. The study will utilize two distinct LLMs: OpenAI's ChatGPT-4o-mini and Gemini-1.0-pro [2], each representing a different implementation of language models with varying capabilities. By conducting this comparison, the research aims to provide insights into how the integration of retrieval systems can enhance the quality and accuracy of responses generated by LLMs, and to identify the strengths and weaknesses of each approach in a controlled experimental setting.

1.1 Objectives

The primary objective of this thesis is to benchmark and compare three distinct approaches to natural language generation: a traditional large language model (LLM), a Retrieval-Augmented Generation (RAG) model, and the advanced Multi-Retrieval-Augmented Generation (Multi-RAG) model. This comparison will be conducted using a specific dataset designed to evaluate the models' ability to generate accurate and contextually relevant responses. The study will employ two LLMs, ChatGPT-4o-mini and Gemini-1.0-pro, as the foundational models for this comparison.

The objectives of the thesis are as follows:

- Evaluate the Performance of Traditional LLMs: Assess the capabilities and limitations of traditional LLMs in generating accurate and contextually relevant responses, particularly in scenarios where the required information extends beyond the model's training data.

- Analyze the Impact of Retrieval-Augmented Generation (RAG): Investigate how the integration of retrieval mechanisms in the RAG model enhances the model's ability to access up-to-date information, and evaluate the effectiveness of this approach compared to the traditional LLM.

- Benchmark Multi-RAG Against RAG and Traditional LLMs: Examine the advantages of the Multi-RAG model, which employs multiple retrieval mechanisms, in improving the diversity and relevance of retrieved information. The study will compare Multi-RAG's performance against both the traditional LLM and the standard RAG model.

- Utilize and Assess Advanced Technologies: Implement the comparison using ChromaDB as the vector database, LangFlow as the development environment, and LangWatch for monitoring and visualization. The thesis will evaluate how these technologies contribute to the performance, efficiency, and scalability of the models.

- Provide Practical Insights and Recommendations: Based on the findings, the thesis aims to offer practical insights into the strengths and weaknesses of each approach, and provide recommendations for their application in various NLP tasks. This includes considerations for response time, accuracy, and the complexity of deployment in real-world scenarios.

Through these objectives, the thesis intends to contribute to the ongoing development of more effective and reliable natural language generation systems, offering a deeper understanding of how retrieval-augmented techniques can be optimized and applied in conjunction with large language models.

Chapter 2

State of the Art

This chapter offers a comprehensive overview of the current state of the art in the field of Retrieval-Augmented Generation (RAG), a cutting-edge technique that enhances the capabilities of language models by integrating them with external data retrieval systems. While large language models (LLMs) play a significant role in RAG, the primary focus of this thesis is on exploring and advancing RAG methodologies. The chapter is structured to provide insights into the platforms that support RAG implementations and to review key scientific paper that has shaped the development of RAG techniques.

2.1 Platforms

The *Platforms* section delves into the tools and frameworks that are essential for developing and deploying RAG systems. These platforms not only support the integration of LLMs with retrieval mechanisms but also provide the necessary infrastructure for testing, monitoring, and optimizing RAG-based applications.

Each platform is evaluated based on its features, strengths, and limitations in the context of RAG.

2.1.1 Langsmith

Langsmith [6] is an advanced platform specifically designed to streamline the development and deployment of large language models (LLMs). As shown in **Fig.** 2.1, it offers a comprehensive suite of DevOps tools tailored for LLM applications, ensuring that models can be effectively tested, monitored, and maintained throughout their lifecycle.

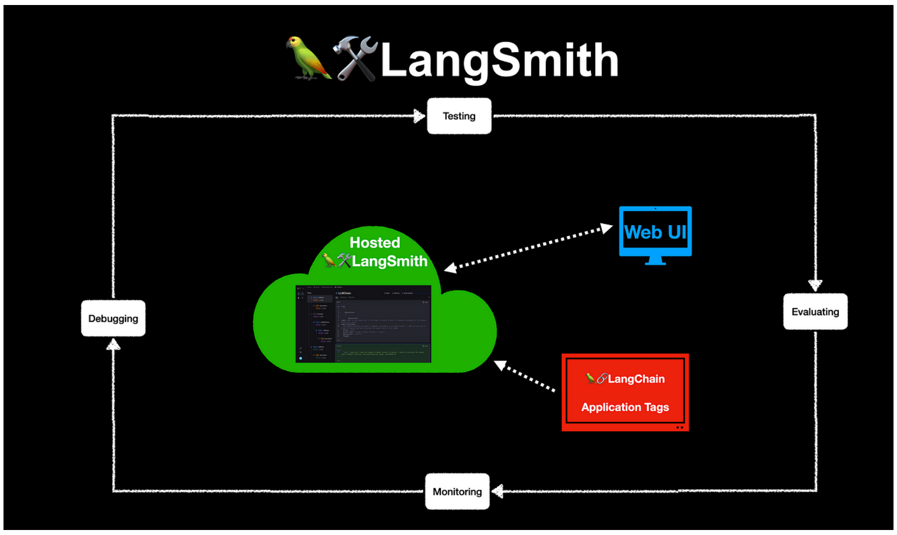


Figure 2.1: The diagram illustrates the various phases and tools involved in the testing, debugging and monitoring processes and evaluation, as well as integration with LangChain.

A key feature of Langsmith is its comprehensive set of tools for testing and debugging LLMs, ensuring that models perform reliably once deployed and addressing potential issues before they impact production environments. The platform also provides real-time performance monitoring, enabling developers to track the performance of their models and quickly resolve any emerging issues.

In addition to these tools, Langsmith includes detailed evaluation capabilities that are crucial for assessing the effectiveness of LLM applications. These tools offer valuable insights into model performance and identify areas where improvements can be made, ensuring that models are truly production-ready.

Another significant advantage of Langsmith is its ability to host LangChain, a powerful integration that allows users to leverage advanced retrieval and workflow capabilities, further enhancing the platform's overall functionality.

Advantages

- Comprehensive Tool Suite: Langsmith provides an extensive set of tools for testing, debugging, and monitoring, enhancing the reliability and effectiveness of LLM models.
- Real-Time Monitoring: The platform offers continuous performance monitoring, facilitating quick issue resolution.
- Detailed Evaluation Tools: Provides in-depth analysis of model performance, enabling targeted improvements.

- Integration with LangChain: Users can benefit from LangChain's advanced retrieval and workflow capabilities, expanding the potential of Langsmith.

Disadvantages

- Steep Learning Curve: The extensive feature set and integration with LangChain can be complex, presenting challenges for new users.
- High Costs: The platform's advanced functionalities and comprehensive tools may incur higher costs, which could be a limiting factor for smaller projects or organizations with constrained budgets.

This revised version maintains clarity while focusing on the essential features and functionalities of the Langsmith platform, ensuring that it is well-integrated within the context of your thesis.

2.1.2 LangChain

LangChain [4] is a versatile framework designed to facilitate the development of applications that leverage language models. It excels in integrating large language models (LLMs) with various data sources and environments, enabling the creation of complex and scalable workflows.

The overview provides a visual representation of LangChain's architecture. It highlights how the framework integrates with multiple data sources and tools, showcasing the flow of information between components. This overview demon-

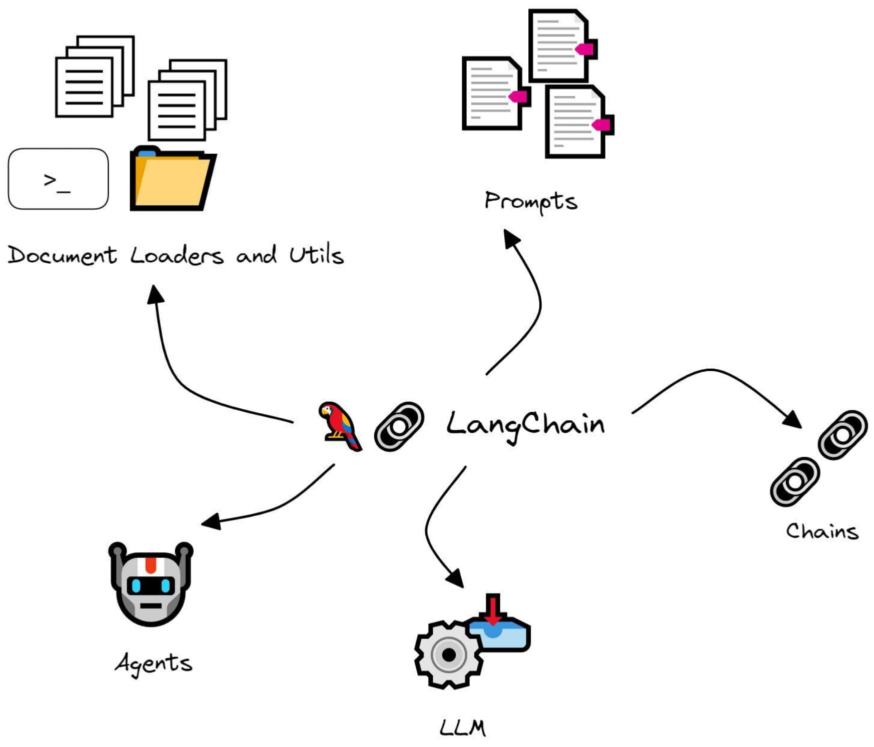


Figure 2.2: Overview of LangChain: This figure illustrates the framework's architecture and its integration with various data sources and tools, showcasing its capability to support complex, scalable workflows.

strates LangChain's capability to manage complex interactions and data handling, illustrating its role in facilitating scalable and efficient language model applications.

LangChain supports Retrieval-Augmented Generation (RAG), a technique that enhances the ability of LLMs to access and utilize external data sources. This feature allows LangChain to generate more effective and contextually relevant responses. The framework is highly customizable, providing the flexibility to tailor solutions to specific use cases.

Additionally, LangChain stands out for its ability to integrate with a wide range of tools and platforms, which enhances its utility across diverse scenarios. This adaptability makes LangChain a powerful choice for developers aiming to build sophisticated applications.

Advantages

- Versatile Integration: LangChain excels at integrating LLMs with various data sources and environments, supporting the development of complex and scalable workflows.
- Retrieval-Augmented Generation (RAG): Improves the LLM's capability to access and utilize external data sources, resulting in more effective and contextually relevant responses.
- High Customizability: Provides significant flexibility to tailor solutions for specific use cases, accommodating a wide range of applications.
- Wide Range of Integrations: Capable of working with a diverse set of tools and platforms, enhancing its functionality across different scenarios.

Disadvantages

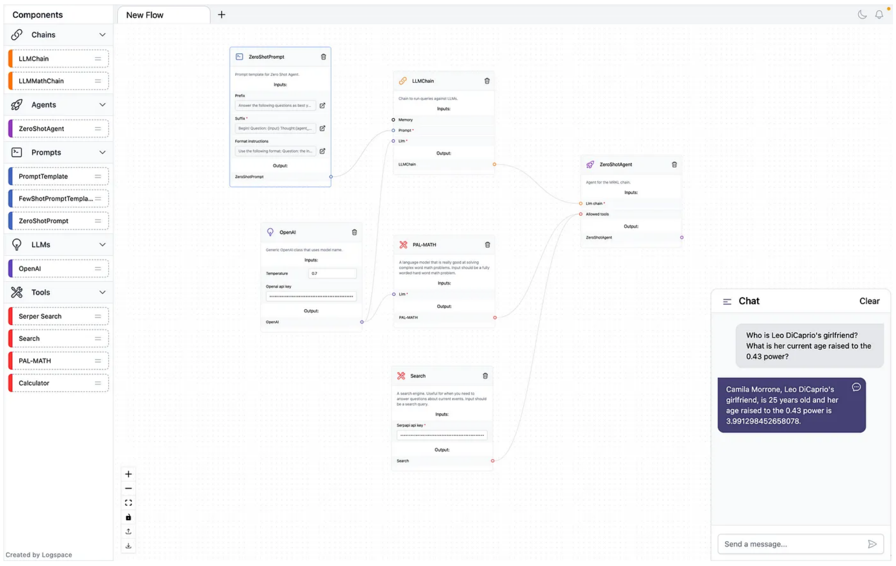
- Complex Setup: The extensive integration capabilities and customization options can make the initial setup and configuration complex.

- Learning Curve: The versatility and flexibility of LangChain may result in a steep learning curve for new users.
- Potential Performance Overhead: Integration with multiple data sources and tools might introduce performance overheads that need to be managed.

2.1.3 Langflow

LangFlow [5] represents a significant advancement in the field of AI application development. It extends the capabilities of LangChain by providing a user-friendly interface specifically designed for orchestrating complex language model workflows. LangFlow's state-of-the-art design offers a visual drag-and-drop environment where users can connect various nodes representing different functionalities, such as LLMs, data processors, and input/output modules. This modular approach allows for the rapid creation and deployment of complex language processing pipelines.

One of LangFlow's most remarkable features is its ability to integrate multiple LLMs and tools within a single workflow. This enables the creation of applications that can perform a wide range of tasks—from text generation and summarization to sentiment analysis and question answering. The platform also supports fine-tuning models on custom datasets, providing users with the flexibility to tailor language models to their specific needs.



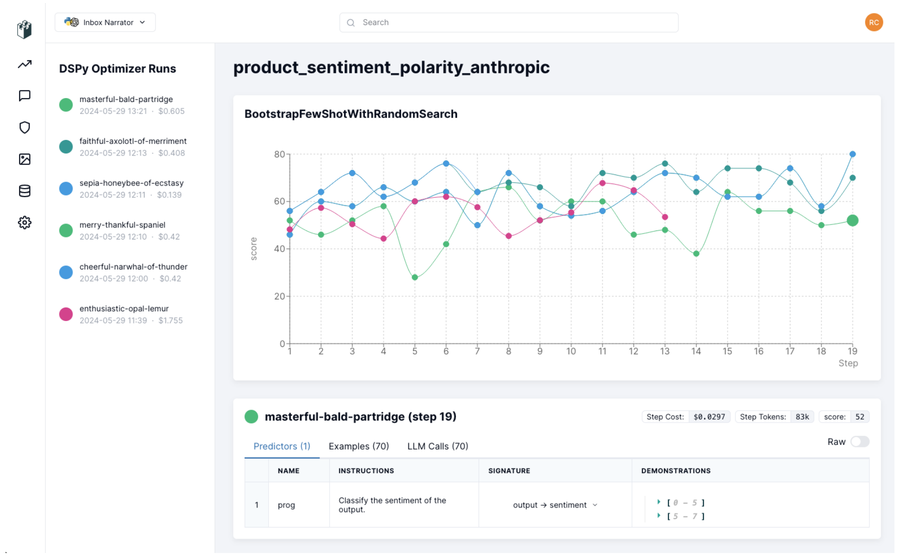
Who is Leo DiCaprio's girlfriend? What is her current age raised to the 0.43 power?

Camila Morrone, Leo DiCaprio's girlfriend, is 25 years old and her age raised to the 0.43 power is 3.991298452658078.

Figure 2.3: Exapmle of Langflow Workspace

LangFlow's interface is not only intuitive but also richly interactive, offering real-time feedback and visualization. Users can dynamically adjust parameters and inspect outputs at each stage of their workflow, simplifying the process of debugging and optimization. Designed with scalability in mind, LangFlow supports cloud-based deployment options, making it suitable for handling large volumes of data and serving numerous end-users.

Moreover, LangFlow integrates with popular version control systems and continuous integration/continuous deployment (CI/CD) pipelines, ensuring seamless integration into modern software development workflows. As an open-source platform with an active community, LangFlow continues to evolve, incorporating the



DSPy Optimizer Runs

- masterful-bald-partridge 2024-05-29 13:21 · $0.605
- faithful-axolotl-of-merriment 2024-05-29 12:13 · $0.408
- sepia-honeybee-of-ecstasy 2024-05-29 12:11 · $0.139
- merry-thankful-spaniel 2024-05-29 12:10 · $0.42
- cheerful-narwhal-of-thunder 2024-05-29 12:00 · $0.42
- enthusiastic-opal-lemur 2024-05-29 11:39 · $1.755

product_sentiment_polarity_anthropic

BootstrapFewShotWithRandomSearch

masterful-bald-partridge (step 19) Step Cost: `$0.0297` Step Tokens: 83k score: 52

Predictors (1) Examples (70) LLM Calls (70)

	NAME	INSTRUCTIONS	SIGNATURE	DEMONSTRATIONS
1	prog	Classify the sentiment of the output.	output → sentiment	[0 – 5] [5 – 7]

Figure 2.4: Example of Langwatch Dashboard

latest advancements in AI and machine learning technologies.

2.1.4 LangWatch

LangWatch [7] represents the forefront of AI operations by offering a comprehensive suite of tools designed specifically for monitoring and optimizing Large Language Models (LLMs). As organizations increasingly rely on LLMs for critical applications—ranging from customer support and content generation to data analysis and decision-making—LangWatch provides the necessary infrastructure to ensure these systems operate at peak efficiency.

The platform excels in providing granular visibility into the performance of LLMs, a capability that traditional monitoring tools often lack due to the complexity of these models. LangWatch not only tracks standard metrics such as CPU and memory usage but also delves deeper into model-specific diagnostics like inference times and response accuracy. This level of detail is essential for troubleshooting and optimizing LLM deployments.

A standout feature of LangWatch is its anomaly detection and predictive analytics, which utilize machine learning to identify and alert users to potential issues before they become critical. This proactive approach helps maintain the reliability and efficiency of AI systems.

LangWatch also plays a vital role in ensuring compliance with regulatory standards. Its logging and auditing capabilities provide transparency into LLM operations, a crucial feature for industries with stringent compliance requirements.

The platform is designed to scale with organizational needs, supporting both small enterprises and large-scale deployments. With its combination of advanced monitoring tools, machine learning analytics, and secure, compliant infrastructure, LangWatch is indispensable for organizations seeking to optimize the performance and reliability of their AI-driven applications.

2.2 Scientific Literature

In recent years, there has been a growing interest in enhancing the capabilities of large language models (LLMs) through various approaches that integrate external knowledge into the models' responses. Among these approaches, Retrieval-Augmented Generation (RAG) has emerged as a particularly effective method for improving the performance of LLMs in knowledge-intensive tasks. This section reviews several key research contributions that have advanced the state of the art in RAG and related techniques.

- Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks by Lewis et al. [16]: This paper introduces Retrieval-Augmented Generation (RAG) models that merge pre-trained sequence-to-sequence (seq2seq) architectures with a dense vector index of Wikipedia. It explores two distinct formulations of RAG: one that employs the same retrieved passages throughout the generated sequence and another that uses different passages for each token. The findings indicate that RAG models significantly outperform traditional seq2seq models on various open-domain question answering tasks.

  The solution proposed in this thesis aims to extend these advancements by incorporating more advanced retrieval techniques and optimizing the augmentation process. This approach seeks to further enhance the performance

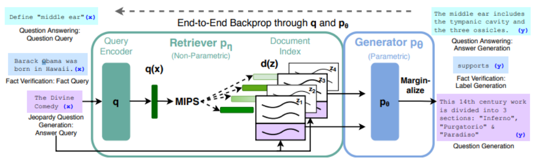

Define "middle ear" (x)
Question Answering: Question Query
Barack Obama was born in Hawaii. (x)
Fact Verification: Fact Query
The Divine Comedy (x)
Jeopardy Question Generation: Answer Query
End-to-End Backprop through q and p_θ
Query Encoder
Retriever p_η
(Non-Parametric)
Document Index
q
q(x)
MIPS
d(z)
z_4
z_3
z_2
z_1
Generator p_θ
(Parametric)
p_θ
Margin-alize
The middle ear includes the tympanic cavity and the three ossicles. (y)
Question Answering: Answer Generation
supports (y)
Fact Verification: Label Generation
This 14th century work is divided into 3 sections: "Inferno", "Purgatorio" & "Paradiso" (y)
Question Generation

Figure 2.5: Combine a pre-trained retriever (Query Encoder + Document Index) with a pre-trained seq2seq model (Generator) and fine-tune end-to-end. For query x, use Maximum Inner Product Search (MIPS) to find the top-K documents zi . For final prediction y, treat z as a latent variable and marginalize over seq2seq predictions given different documents.

and efficiency of RAG systems.

- Meta Knowledge for Retrieval Augmented Large Language Models by Laurent Mombaerts et al. [17]:

  This research presents a novel data-centric RAG workflow for large language models (LLMs), transforming the traditional retrieve-then-read system into a more sophisticated prepare-then-rewrite-then-retrieve-then-read framework. It focuses on generating metadata and synthetic question-answer pairs for each document to improve retrieval precision and recall. The proposed solu-

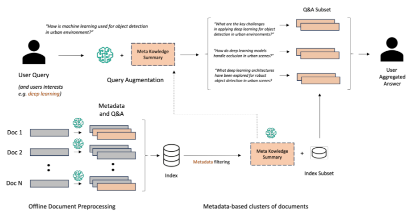

"How is machine learning used for object detection in urban environment?"
User Query
(and users interests e.g. deep learning)
Meta Kowledge Summary
Query Augmentation
Q&A Subset
"What are the key challenges in applying deep learning for object detection in urban environments?"
"How do deep learning models handle occlusion in urban scenes?"
"What deep learning architectures have been explored for robust object detection in urban scenes?
User Aggregated Answer
Metadata and Q&A
Doc 1
Doc 2
Doc N
Index
Metadata filtering
Meta Kowledge Summary
Index Subset
Offline Document Preprocessing
Metadata-based clusters of documents

Figure 2.6: Data-centric workflow for Retrieval Augmented Generation (RAG) systems. Prior to inference, documents are augmented with Claude 3, and clustered into metadata-based sets of synthetic questions and answers for personalized downstream retrieval. Meta Knowledge Summaries are used to guide the query augmentation step with clusters information.

tion in the thesis aims to advance the retrieval and augmentation processes further, with a focus on real-time application and scalability, building on the principles introduced in this research.

- A Hybrid RAG System with Comprehensive Enhancement on Complex Reasoning by Ye Yuan et al. [18]:

  This paper describes a hybrid RAG system enhanced through a series of optimizations that improve retrieval quality, augment reasoning capabilities, and refine numerical computations. The system significantly enhances complex reasoning abilities and reduces error rates.

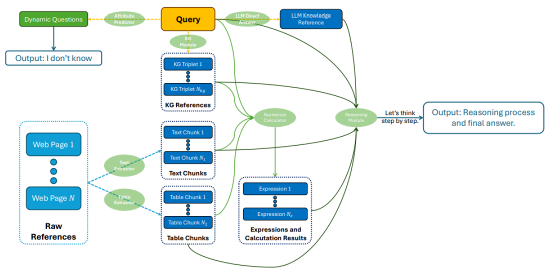


Figure 2.7: The complete design of system. There are two possible routes for the generation. If the query is classified by the in-context learning as "dynamic", we will output "I don't know" directly to reduce hallucination on these hard problems.

  The thesis will investigate how to incorporate similar enhancements while focusing on multimodal data integration and real-time processing, aiming to expand the functionality and applicability of the RAG system.

- Retrieval-Augmented Generation for Large Language Models: A Survey [14]:

This survey provides an extensive analysis of the RAG framework, detailing advancements in retrieval, generation, and augmentation techniques. It highlights the latest technologies embedded in each component of the RAG framework. The proposed solution in the thesis seeks to leverage insights

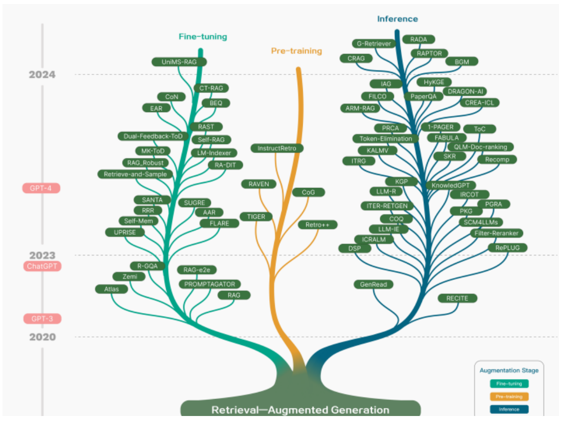


Figure 2.8: Enter Caption

from this survey to develop a more robust and efficient RAG framework, emphasizing continuous knowledge updates and integration with external databases.

Chapter 3

Technologies

3.1 LangFlow

LangFlow is a sophisticated open-source platform designed to simplify and enhance the process of building applications that utilize Large Language Models (LLMs), such as OpenAI's GPT series. The platform functions as a graphical user interface (GUI) that allows developers, researchers, and even non-technical users to design, prototype, and deploy AI-driven applications without needing to delve into the complexities of coding or API integrations.

Main Features of LangFlow

- Visual Interface: LangFlow provides a user-friendly graphical interface that enables users to create workflows by dragging and connecting blocks representing various components, such as language models, databases, and natural language processing tools. This intuitive environment simplifies pipeline creation, making it accessible even to those without advanced programming

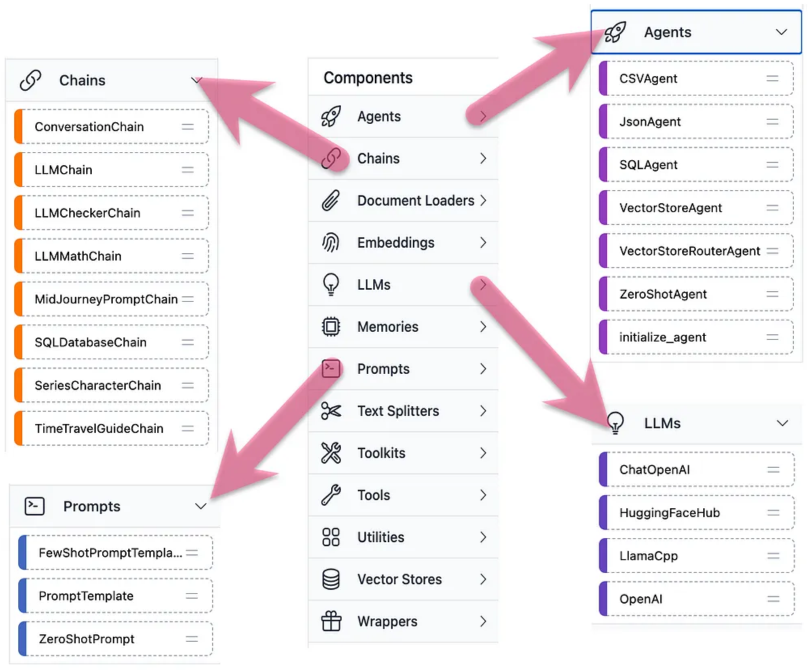


Figure 3.1: The image shows a list of the different component categories available. When expanded, the list highlights all development possibilities for Chains, Reminders, Agents and LLMs. This list is expected to expand as interest increases.

skills.

- Support for Language Models (LLMs): LangFlow integrates a wide range of language models, including advanced models like GPT-4, Gemini,LLama and other specialized models for various NLP tasks. Users can select and configure the model that best suits their specific needs.

- Integration with LangChain: A distinctive feature of LangFlow is its compatibility with LangChain, a framework that orchestrates multiple LLMs within a single pipeline. This allows the creation of complex inference chains, combining the capabilities of different models for more refined and specific outcomes.

- Compatibility with Various Databases: LangFlow supports integration with multiple databases, including PostgreSQL, MongoDB, and Redis. This facilitates efficient data management within workflows, offering both persistent storage and rapid data retrieval.
- Extended Customization: LangFlow is highly customizable, allowing users to develop custom components to meet the specific needs of their projects. This flexibility enables the integration of external tools and modules not natively supported by the platform.

Advantages of Using LangFlow

- Ease of Use: The visual interface makes AI pipeline creation accessible to non-programmers, accelerating development and experimentation processes.
- Flexibility and Scalability: With support for various language models and databases, LangFlow is versatile, suitable for applications ranging from simple text processing to complex large-scale linguistic analysis.
- Rapid Integration and Experimentation: LangFlow enables quick integration of new components and experimentation with different LLMs and pipeline configurations, making it ideal for research and iterative development.

Disadvantages of Using LangFlow

- Learning Curve: While the visual interface is user-friendly, mastering the full capabilities of LangFlow, particularly for complex workflows, may require time and effort.
- Resource Intensive: Running multiple LLMs and extensive pipelines can be resource-intensive, potentially requiring significant computational power and infrastructure.
- Customization Complexity: Although LangFlow is highly customizable, developing and integrating custom components may require advanced programming knowledge, which could be a barrier for some users.
- Limited Out-of-the-Box Functionality: For highly specialized or niche applications, users may need to develop additional modules or integrations, as the platform's native functionalities might not cover all use cases.

3.2 Chroma DB

ChromaDB [12] is an optimized database designed for information retrieval, specifically tailored to support advanced artificial intelligence applications. It is particularly well-suited for scenarios requiring fast and accurate access to relevant data for natural language generation (NLP) and other machine learning (ML) func-

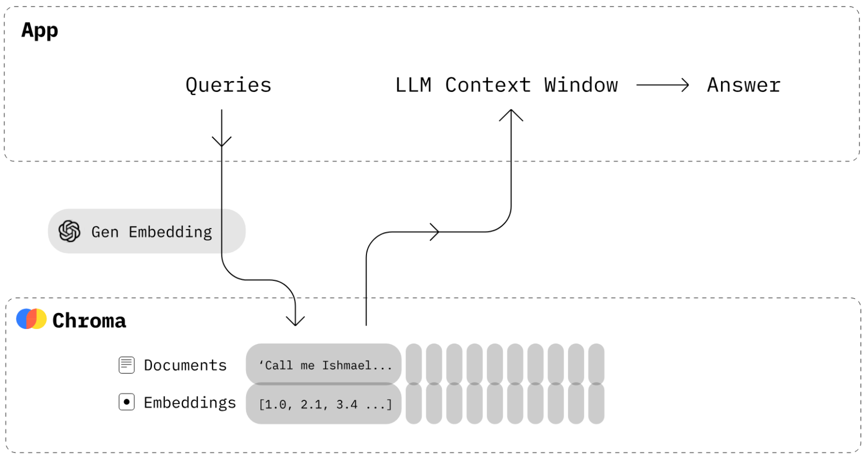


Figure 3.2: The system retrieves the relevant text data from ChromaDB. This involves finding and using pre-computed embeddings associated with the stored text.

tions. ChromaDB is a high-performance data storage and retrieval system capable of handling large volumes of information and executing complex queries efficiently.

Key Features of ChromaDB

1. Optimized Retrieval Architecture: ChromaDB is built with an architecture that prioritizes speed and efficiency in information retrieval. This is crucial for real-time applications, such as those utilizing Retrieval-Augmented Generation (RAG) models, where the speed of retrieving relevant documents directly impacts system performance and usability.

2. Advanced Indexing: At its core, ChromaDB features an advanced indexing system that categorizes and organizes large datasets to facilitate rapid and precise retrieval. Indexing can be based on various criteria, such as keywords,

metadata, semantic similarity, and other specific attributes of the stored information, ensuring that queries are answered with the most pertinent documents available.

3. Support for Structured and Unstructured Data: ChromaDB handles both structured data (e.g., tables and records) and unstructured data (e.g., free text, articles, PDF documents). This versatility allows the database to be used in a wide range of applications, from simple text searches to complex analyses based on large collections of documents.

4. Scalability: The system is designed to be highly scalable, capable of managing growing data collections over time without compromising performance. ChromaDB can be distributed across multiple servers or nodes, supporting expansion in data volume and retrieval requests without degrading speed or efficiency.

5. Semantic Search and Similarity: Beyond traditional keyword-based searches, ChromaDB offers advanced semantic search capabilities. This means it can find and retrieve documents not only based on specific words but also based on the meaning and context of sentences or paragraphs. Semantic search capabilities are essential for advanced NLP applications, where finding relevant information is critical even if it does not exactly match the original query terms.

6. Integration with Machine Learning Systems: ChromaDB is designed to integrate seamlessly with machine learning models, including large language models (LLMs). This makes it particularly useful for RAG configurations, where the retrieval process is directly linked to language generation. The database can provide structured and unstructured data in a format that ML models can use directly to enhance the quality of generated responses.

7. Filtering and Ranking Capabilities: ChromaDB includes sophisticated filtering and ranking features, allowing results to be sorted based on relevance or other user-defined criteria. This is essential for ensuring that generated responses are not only correct but also the most pertinent to the user's query.

3.2.1 ChromaDB in the Context of RAG and Multi-RAG Models

When using RAG or Multi-RAG models, ChromaDB plays a critical role as the primary data source for the retrieval module. Here's how it operates in practice:

1. Retrieval Process in RAG Models: When a user submits a question or input to the system, the retrieval module connected to ChromaDB performs a query to find the most relevant documents related to that request. These documents are then used by the LLM to generate a contextualized and informed response.

2. Multi-RAG and Parallel Retrieval: In the case of Multi-RAG, ChromaDB can be configured to execute parallel retrievals across different sections of the database or using multiple search criteria. This approach allows for the collection of a broader range of information, which is then combined by the language model to produce a richer and more comprehensive response.

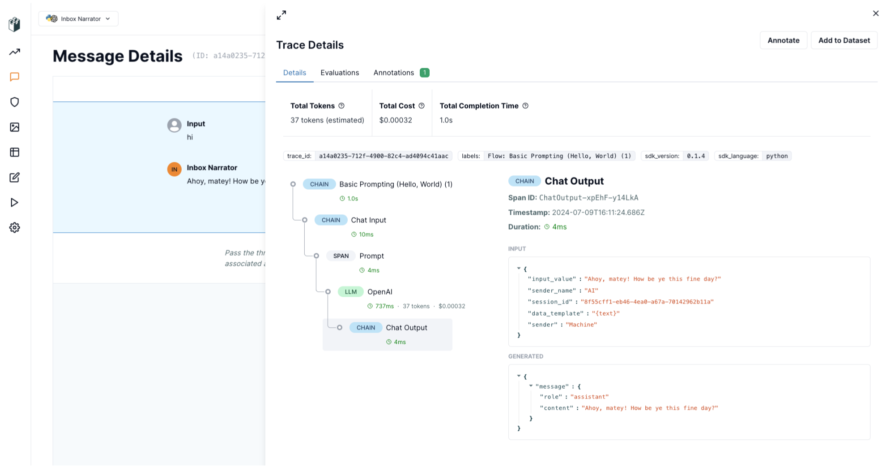
3. Relevance Optimization: Thanks to its advanced search and ranking capabilities, ChromaDB ensures that the retrieved documents are the most relevant and high-quality, thereby improving the accuracy and relevance of the generated responses. This is particularly important in contexts where information correctness and timeliness are critical, such as in medical or legal fields.

3.3 Langwatch

LangWatch is a cutting-edge platform designed for monitoring, analyzing, and optimizing the performance of applications utilizing Large Language Models (LLMs). It provides deep visibility into the operational behavior of LLM-based systems, offering essential tools for ensuring reliability, security, and efficiency in AI-driven applications.

3.3.1 Key Features of Langwatch

1. Real-Time Monitoring: Langwatch enables the real-time observation of language models in operation. This includes tracking data flow through the


Message Details (ID: a14a0235-712

Input
hi

Inbox Narrator
Ahoy, matey! How be y

Trace Details

Total Tokens: 37 tokens (estimated)
Total Cost: $0.00032
Total Completion Time: 1.0s

trace_id: a14a0235-712f-4900-82c4-ad4094c41aac
labels: Flow: Basic Prompting (Hello, World) (1)
sdk_version: 0.1.4
sdk_language: python

```
CHAIN Basic Prompting (Hello, World) (1)
      1.0s
  CHAIN Chat Input
        10ms
    SPAN Prompt
         4ms
      LLM OpenAI
          737ms · 37 tokens · $0.00032
        CHAIN Chat Output
              4ms
```

CHAIN Chat Output
Span ID: ChatOutput-xpEhF-y14LkA
Timestamp: 2024-07-09T16:11:24.686Z
Duration: 4ms

INPUT

```
{
  "input_value" : "Ahoy, matey! How be ye this fine day?"
  "sender_name" : "AI"
  "session_id" : "8f55cff1-eb46-4ea0-a67a-70142962b11a"
  "data_template" : "{text}"
  "sender" : "Machine"
}
```

GENERATED

```
{
  "message" : {
    "role" : "assistant"
    "content" : "Ahoy, matey! How be ye this fine day?"
  }
}
```

Figure 3.3: Langflow visualizes the workflow for each stage, including time, cost and metadata details, to monitor and optimize the process in detail

pipeline, identifying bottlenecks, and logging relevant events, such as errors or slowdowns. Real-time monitoring is crucial for quickly identifying operational issues that could affect the quality or efficiency of generated responses.

2. Error Message Analysis: The tool captures and thoroughly analyzes error messages generated by the models. These may include runtime errors, memory issues, or other anomalies that could impair proper functioning. Error message analysis allows for rapid diagnosis and resolution of problems, improving the robustness and reliability of the pipelines.

3. Performance Evaluation: Langwatch provides detailed metrics on model performance, such as response time, resource usage (CPU, GPU, memory), and overall pipeline latency. This information is essential for optimizing models, especially in contexts where speed and efficiency are critical.

4. Response Quality Metrics: In addition to technical performance, Langwatch allows for the evaluation of the quality of responses generated by the models. This may include metrics for accuracy, coherence, relevance, and completeness of the responses. These parameters are crucial for determining a model's effectiveness in real-world applications, where the quality of generated text is as important as its speed.

5. Intuitive Dashboard: Langwatch features an intuitive dashboard that displays all key information in an organized and accessible manner. Through customizable graphs, tables, and reports, users can easily monitor the trends of performance and quality metrics, facilitating decision-making to improve models.

6. Integration with Existing Systems: Langwatch is designed to seamlessly integrate with other development and monitoring tools already in use. This includes compatibility with NLP pipelines built with tools like LangFlow and databases like ChromaDB, allowing for an integrated and comprehensive view of system performance.

3.4 GPT-4-mini

GPT-4-mini [1] is a scaled-down version of the large language model (LLM) GPT-4, developed by OpenAI. Despite its smaller size compared to the full GPT-4 model, GPT-4-mini retains many of the advanced capabilities of the original, making it a versatile and powerful tool for a wide range of natural language processing (NLP) applications.

GPT-4-mini is designed to strike a balance between performance and efficiency. Due to its reduced architecture, the model requires fewer computational resources and less memory space than the full version, making it suitable for experimental settings or environments with limited resources. Despite its reduced size, GPT-4-mini maintains the ability to understand and generate coherent and contextually relevant text, thanks to its training on a vast corpus of linguistic data.

3.4.1 Key Features of GPT-4-mini

1. Advanced Linguistic Capabilities: GPT-4-mini retains a deep understanding of linguistic structures, semantic ambiguities, and context, similar to the full GPT-4 version. It is capable of generating fluent and cohesive text, responding pertinently and contextually to a wide range of queries and inputs.

2. Computational Efficiency: One of the main features of GPT-4-mini is its efficiency. With a reduced number of parameters compared to GPT-4, the model requires less computational power and can be run on more modest hardware, making it ideal for quick experiments and testing.

3. Versatility: Despite its smaller size, GPT-4-mini is versatile and can be used in various application domains, from creative text generation to answering

specific questions, customer support, and sentiment analysis.

4. Generalization Ability: GPT-4-mini retains a good generalization capability, meaning it can apply the knowledge learned during training to new contexts or questions that were not necessarily present in the training dataset.

5. Adaptability to Different Models: One of the primary reasons for using GPT-4-mini in experimental contexts is its adaptability. It can be easily integrated into various model architectures, including traditional language models, RAG (Retrieval-Augmented Generation) models, and Multi-RAG, making it a flexible option for comparative experimentation.

3.5 Text-Embedding-3-Small by OpenAI

Text-Embedding-3-Small [8], developed by OpenAI, is an advanced embedding model specifically designed to convert text into compact and high-quality vector representations, known as embeddings. This model is optimized for tasks that require efficient and accurate text processing, making it an ideal component for enhancing the performance of various natural language processing (NLP) applications, including those involving smaller, resource-constrained environments.

Key Features of Text-Embedding-3-Small

1. Compact Embeddings: Text-Embedding-3-Small generates embeddings that are smaller in size compared to larger models, without significantly com-

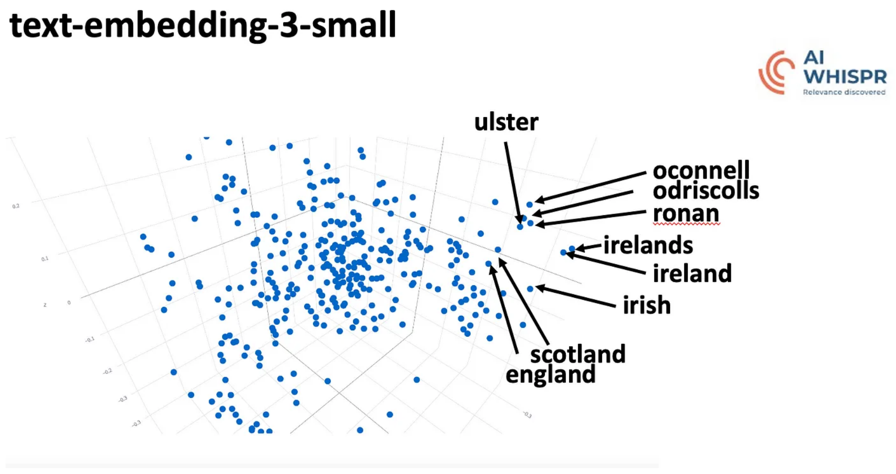


Figure 3.4: Each point in the vector space corresponds to a unique word.

promising on the quality of the representations. This compactness makes the model highly suitable for applications where computational resources are limited, such as edge devices or environments with restricted memory capacity.

2. Efficiency and Speed: The model is designed to be highly efficient, offering fast embedding generation. This speed is particularly beneficial in real-time applications, where quick processing of text is essential to maintain performance standards.

3. Versatile Application: Despite its smaller size, Text-Embedding-3-Small maintains a broad applicability across various domains. It can be effectively used in tasks such as semantic search, text classification, and clustering, providing accurate and meaningful insights even with its reduced footprint.

4. Contextual Understanding: Text-Embedding-3-Small retains the ability to produce context-aware embeddings, ensuring that the generated vectors accurately reflect the meaning and nuances of the input text. This capability is crucial for tasks that involve understanding the subtleties of language.

5. Integration with GPT-4-Mini: In the context of the GPT-4-Mini model, Text-Embedding-3-Small serves as the primary embedding generator. By leveraging these compact and efficient embeddings, GPT-4-Mini can achieve a balance between performance and resource utilization, making it a powerful tool for a wide range of NLP tasks, even in environments with limited computational power.

Text-Embedding-3-Small by OpenAI is a crucial technology for the implementation of GPT-4-Mini, enabling the model to efficiently handle text processing tasks while maintaining high levels of accuracy and contextual relevance. Its design emphasizes both performance and scalability, ensuring that it can meet the demands of various NLP applications.

3.6 Gemini-1.0-pro

Gemini-1.0-pro [2] is a cutting-edge language model designed to deliver high performance and resource efficiency. Unlike larger models that demand significant computational resources, Gemini-1.0-pro achieves a balance between model complexity and operational efficiency, making it suitable for scenarios where computational resources are limited or real-time processing is crucial.

Gemini-1.0-pro has been engineered to provide robust capabilities in language understanding and generation, enabling it to handle a variety of NLP tasks including text generation, summarization, translation, and sentiment analysis with high accuracy and coherence.

3.6.1 Key Features of Gemini-1.0-pro

1. Optimized Architecture: Gemini-1.0-pro features a streamlined architecture that minimizes the number of parameters while maintaining strong performance. This optimization allows Gemini-1.0-pro to function effectively on less powerful hardware without compromising the quality of text generation.

   vbnet Copia codice

2. High Efficiency: Gemini-1.0-pro is designed to maximize efficiency in terms of computational power and memory usage. This makes it ideal for applications where speed and responsiveness are critical, such as real-time data processing or deployment on devices with limited resources.

3. Advanced Text Processing: Despite its compact size, Gemini-1.0-pro excels in a range of NLP tasks, offering advanced capabilities in understanding and generating text. It can handle complex sentence structures, grasp subtle

nuances in language, and provide contextually relevant responses.

4. Scalability and Flexibility: Gemini-1.0-pro is highly adaptable, making it suitable for various applications and integration into different systems. Whether used in standalone NLP tasks or as part of a broader machine learning pipeline, Gemini-1.0-pro's scalability ensures it meets the demands of diverse projects.

5. Seamless Integration: Built with compatibility in mind, Gemini-1.0-pro allows for easy integration with existing systems and workflows. This ensures quick adoption and utilization without the need for significant changes to underlying infrastructure.

The use of Gemini-1.0-pro in this project highlights the focus on achieving a balance between performance and efficiency, ensuring that the system remains powerful and practical for real-world applications.

Chapter 4

Method

In this chapter, we describe the proposed solution for evaluating and benchmarking various natural language generation approaches. This includes a detailed discussion of the model architectures, training procedures, loss functions, and evaluation metrics used in our study. Our goal is to provide a comprehensive overview of the methods and technologies employed to compare traditional large language models (LLMs), Retrieval-Augmented Generation (RAG), and Multi-Retrieval-Augmented Generation (Multi-RAG) models.

4.1 No-RAG Approach

The No-RAG (No-Retrieval-Augmented Generation) approach evaluates the performance of traditional large language models (LLMs) without any augmentation from external retrieval mechanisms. It assesses how well LLMs generate responses based solely on the internal knowledge acquired during their pre-training, without accessing external knowledge sources. The LLMs considered in this approach include models like ChatGPT-4o-mini and Gemini-1.0-pro. These models are tasked with answering a set of reference questions purely based on their pre-existing knowledge, providing insights into their standalone capabilities.

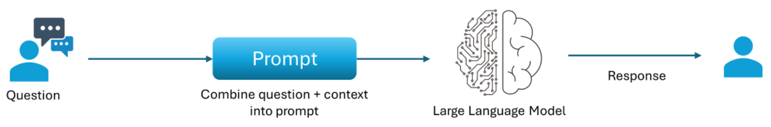


Figure 4.1: Architecture of the No-RAG approach

4.1.1 Architecture

The No-RAG architecture revolves around utilizing pre-trained language models to generate responses purely from their learned internal knowledge. Figure 4.8 illustrates the simple yet effective architecture used in the No-RAG approach. Key architectural components are as follows:

- Language Model (LLM): The central component of this architecture is the pre-trained LLM, such as GPT-4o-mini or Gemini-1.0. These models have been trained on vast amounts of text data and can generate responses based on patterns learned during training. In the No-RAG approach, no external database or retrieval system is involved.

- Input Interface: The system accepts user prompts or questions via an input interface, which is then passed to the LLM. The architecture is designed to process these inputs without requiring access to external knowledge stores or search engines.

- LLM Processing: Once the input is received, the LLM processes it internally using its stored knowledge. This is a purely generative process where the LLM leverages its training data to generate responses.

- Output Generation: After processing, the model outputs a response based on its internal reasoning capabilities. Since no external retrieval is involved, the quality of the output is dependent solely on how well the LLM was trained on related data.

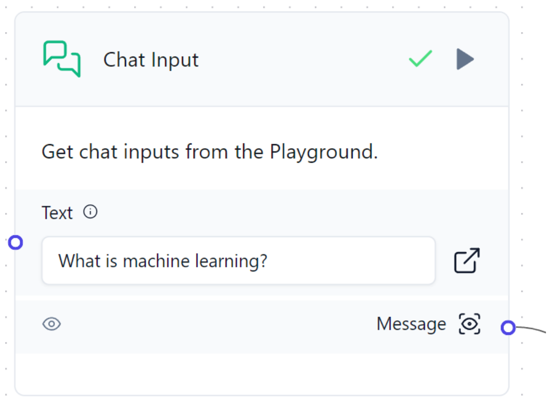
- Final Output: The model returns a generated response to the user's prompt. Since the model is not augmented with external knowledge or retrieval tools, the output solely relies on the LLM's ability to recall and synthesize relevant information from its training data.

This architecture enables LLMs to operate independently, which can be useful in environments where access to external databases is restricted or unavailable.

4.1.2 Workflow (Langflow-based)

In the No-RAG workflow, the Langflow tool is employed to facilitate each step of the process, from prompt design to response generation. The following are the main stages of the workflow, visualized through Langflow pipelines:

- User Input: The user provides a question that serves as the input to the model. The text should clearly express the intent of the user, such as asking for an explanation, clarification, or specific information.

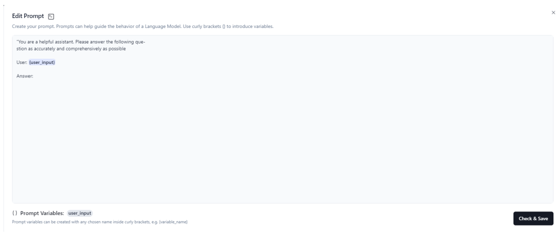


Get chat inputs from the Playground.

What is machine learning?

- Standard Prompt: A standardized prompt is applied to ensure the LLM provides a structured and standardized for consistent evaluation. The prompt used is:

- LLM Processing: After receiving the prompt, the LLM (e.g., GPT-4o-mini or Gemini-1.0) processes the input using its trained data. No external retrieval


Edit Prompt

Create your prompt. Prompts can help guide the behavior of a Language Model. Use curly brackets {} to introduce variables.

"You are a helpful assistant. Please answer the following que-
stion as accurately and comprehensively as possible

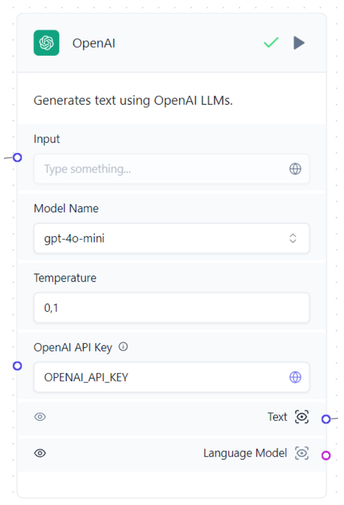
User: {user_input}

Answer:

{} Prompt Variables: user_input

Prompt variables can be created with any chosen name inside curly brackets, e.g. {variable_name}

systems or databases are queried during this step, making it a purely generative task. In Langflow, the configuration for the model can be adjusted. For instance, the *temperature* setting controls the degree of randomness in responses, with a lower value yielding more deterministic outputs, while a higher value introduces variability.


OpenAI

Generates text using OpenAI LLMs.

Input: Type something...

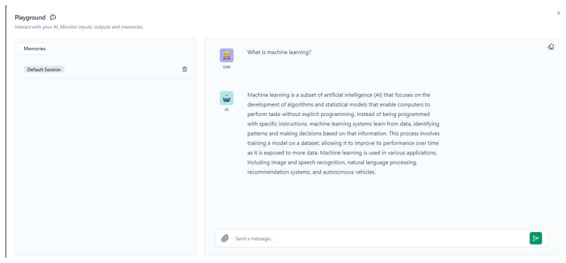
Model Name: gpt-4o-mini

Temperature: 0,1

OpenAI API Key: OPENAI_API_KEY

- Response Generation: Once the LLM processes the input, it generates a response. The output is a reflection of the model's internal knowledge and training quality. The Langflow tool provides a real-time display of this output, allowing users to interactively monitor the process.

Langflow's Playground feature displays the model's response and allows users to review past interactions for further analysis and refinement.


Playground
Interact with your AI. Monitor inputs, outputs and memories.
Memories
Default Session
User
AI
Send a message...

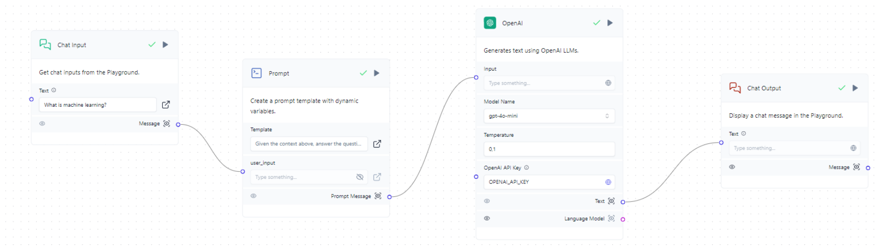
What is machine learning?

Machine learning is a subset of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform tasks without explicit programming. Instead of being programmed with specific instructions, machine learning systems learn from data, identifying patterns and making decisions based on that information. This process involves training a model on a dataset, allowing it to improve its performance over time as it is exposed to more data. Machine learning is used in various applications, including image and speech recognition, natural language processing, recommendation systems, and autonomous vehicles.

- Flow Pipeline:

  Figure above presents the visual representation of the pipeline used in Langflow for the No-RAG workflow. Each component of the pipeline is designed to ensure the LLM operates autonomously without relying on external resources.

By using this pipeline, the No-RAG approach allows for an efficient evaluation of LLM performance based solely on their internal capabilities. Langflow provides a structured environment to test multiple prompts and configurations consistently, helping evaluate the generative accuracy of the model without external retrieval.


Chat Input
Get chat inputs from the Playground.
Message
Prompt
Create a prompt template with dynamic variables.
Prompt Message
OpenAI
Generates text using OpenAI LLMs.
Input
Type something...
Text
Language Model
Chat Output
Display a chat message in the Playground.
Text
Type something...
Message

Text: What is machine learning?

Template: Given the context above, answer the questi...

user_input: Type something...

Model Name: gpt-4o-mini

Temperature: 0,1

OpenAI API Key: OPENAI_API_KEY

4.2 Simple RAG

The Simple-RAG (Retrieval-Augmented Generation) approach enhances a language model's ability to answer queries by integrating an external retrieval mechanism. In this setup, document embeddings are stored in a vector database (e.g., ChromaDB), and relevant context is retrieved based on user input to augment the model's internal knowledge. The main goal is to improve the accuracy and relevance of the model's responses by providing access to external information not present in its training data.

4.2.1 Architecture

The architecture of Simple-RAG combines a pre-trained language model (LLM) with an external retrieval mechanism through a vector database like ChromaDB. The architectural structure consists of the following key elements:

- LLM (Pre-Trained Language Model): Models like GPT-4o-mini are used to process user queries. Unlike the No-RAG approach, here the model is supported by a retrieval mechanism that provides additional context.

- Vector Database (ChromaDB): Documents are processed and converted into embeddings (vector representations) which are stored in the vector database. When a user submits a query, the system searches the database for relevant document chunks based on embedding similarity.

- External Knowledge Integration: Once relevant context is retrieved from the vector database, it is sent to the model along with the user's query. The model then generates a response enriched with external information, improving the overall quality of the answers.

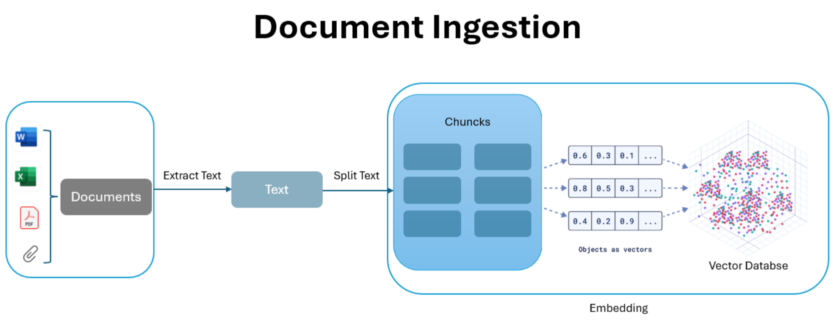


Figure 4.2: Illustrates the document ingestion process, which involves splitting texts into smaller chunks and generating embeddings. These embeddings are then stored in the database for later retrieval.

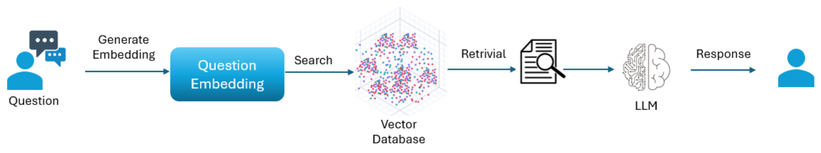
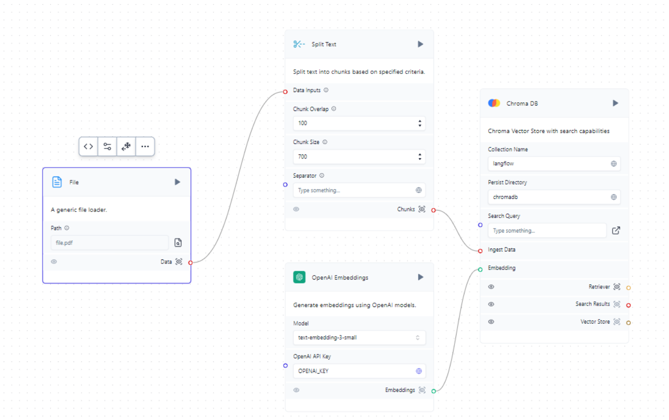


Figure 4.3: Querying: The input is embedded and compared to stored document embeddings in the database. The most similar contexts are retrieved and passed to the LLM, which generates the final response based on both the retrieved information and its internal knowledge.

4.2.2 Workflow (Langflow-based)

The Simple-RAG workflow uses Langflow to orchestrate each step, from document ingestion to response generation. The process involves the following steps:

1. Document Ingestion: First, documents are ingested into a vector database on ChromaDB. This involves splitting documents into chunks with a chunk size of 700 and a chunk overlap of 100. The default separators are standard sentence or paragraph breaks. An embedding model such as *embedding-3-small* is used to generate vector representations (embeddings) for each chunk. These embeddings are stored in the ChromaDB vector database for later retrieval. As shown in the image, ChromaDB is used locally, where a collection name is assigned as *Langflow* and the *persist directory* is set to *chromadb*, ensuring the data is stored for future querying.

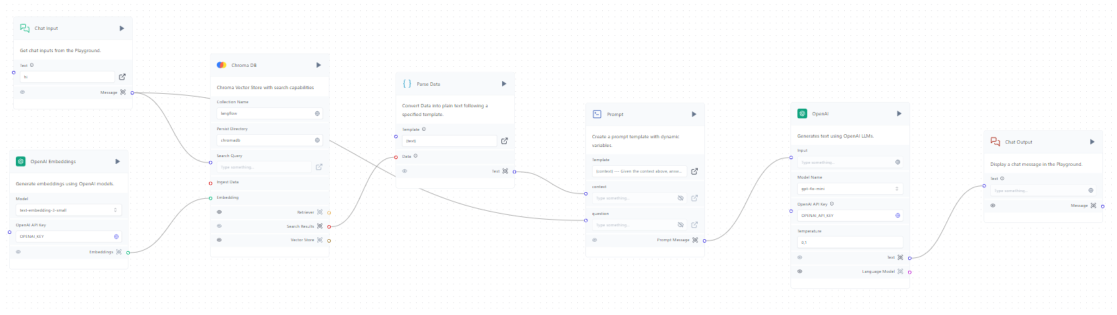


Path: file.pdf
Chunk Overlap: 100
Chunk Size: 700
Separator: Type something...
Model: text-embedding-3-small
OpenAI API Key: OPENAI_KEY
Collection Name: langflow
Persist Directory: chromadb
Search Query: Type something...

Figure 4.4: Document Ingestion: ChromaDB is used as a local vector database, with a collection name set to *Langflow* and a persistence directory specified as *chromadb*.

2. Context Retrieval: When the user provides input, the system computes the embedding of the input and searches the ChromaDB database for relevant document chunks. The retrieved chunks are formatted as plain text and

passed along with the user's original input to the language model.


Chat Input
Get chat inputs from the Playground.
Message
OpenAI Embeddings
Generate embeddings using OpenAI models.
Embeddings
Chroma DB
Chroma Vector Store with search capabilities
Ingest Data
Embedding
Retriever
Search Results
Vector Store
Parse Data
Convert Data into plain text following a specified template.
Text
Prompt
Create a prompt template with dynamic variables.
Prompt Message
OpenAI
Generates text using OpenAI LLMs.
Text
Language Model
Chat Output
Display a chat message in the Playground.
Message

Text: hi
Model: text-embedding-3-small
OpenAI API Key: OPENAI_KEY
Collection Name: langflow
Persist Directory: chromadb
Search Query: Type something...
Template: {text}
Data
Template: {context} --- Given the context above, answ...
context: Type something...
question: Type something...
Input: Type something...
Model Name: gpt-4o-mini
OpenAI API Key: OPENAI_API_KEY
Temperature: 0,1
Text: Type something...

Figure 4.5: Context Retrieval: The system searches ChromaDB for relevant document chunks based on the user's input embedding. The retrieved text is then passed to the LLM.

3. LLM Processing: The GPT-4o-mini model processes both the user's input and the retrieved context. Langflow manages the interaction between the input, retrieval process, and response generation. The temperature parameter of the model can also be adjusted to control the variability of the generated responses.

4. Response Generation: The model generates a response that combines its pre-trained knowledge with the external information retrieved from ChromaDB. This ensures that the answer is more accurate and contextually relevant than one generated solely from internal knowledge.

5. Final Output: The final response is displayed in the Langflow Playground interface. Users can view the retrieved context and monitor the full chat history to understand how the retrieved information influenced the final out-

put.

4.3 Multi-RAG

The Multi-RAG (Multi-Source Retrieval-Augmented Generation) approach enhances the Simple-RAG model by leveraging multiple sources or databases to retrieve relevant context. This allows the model to access a broader range of external knowledge, improving both the accuracy and relevance of its responses. Multi-RAG retrieves context from various vector databases or document sources, combining them to supplement the model's internal knowledge.

4.3.1 Architecture

The architecture of Multi-RAG is designed to handle multiple sources of information. It relies on a pre-trained language model (LLM) and multiple vector databases. The workflow starts with the original question from the user, which is sent to the LLM. The LLM generates multiple related questions based on the original query, increasing the diversity of possible relevant information. Each of these questions is then passed to the vector databases, where an embedding model is used to search for similar content. The retrieved text from each database is then combined and passed back to the LLM, which integrates the information to generate a final response.

The main components of the Multi-RAG architecture are:

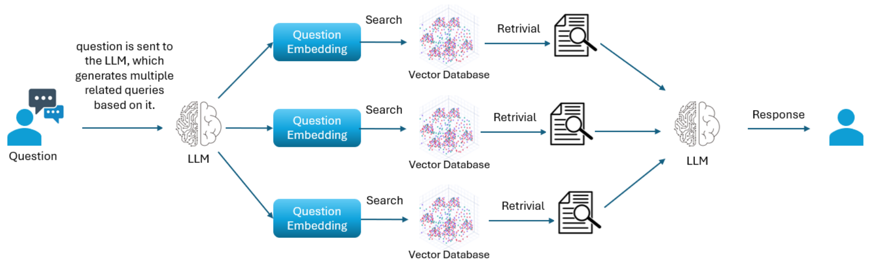


Figure 4.6: Multi-RAG architecture: The original question is expanded into multiple queries, which are sent to multiple vector databases. Retrieved context is passed to the LLM for response generation.

- **LLM (Pre-Trained Language Model):** A language model, such as GPT-4o-mini, is responsible for processing the user's input and retrieved external contexts.

- **Multiple Vector Databases (e.g., ChromaDB, FAISS):** Different vector databases represent various sources of knowledge. Documents are embedded into these databases, and each query is used to search for relevant information in these distinct sources.

- **Multi-Source Retrieval:** The LLM generates multiple queries based on the original question, and each query is used to search different vector databases. This allows for a comprehensive retrieval process, gathering information from various perspectives.

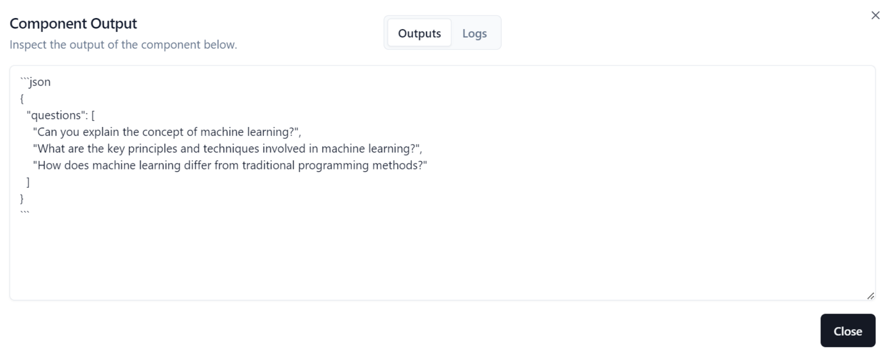
- **External Knowledge Integration:** Once the relevant document chunks are retrieved from the databases, they are combined into a unified context and passed to the LLM for response generation.

4.3.2 Flow (Based on Langflow)

In Multi-RAG, Langflow orchestrates the interaction between the LLM and multiple retrieval sources. The workflow proceeds as follows:

1. **Document Ingestion into Multiple Databases:** Documents are split into smaller chunks, embedded using a model like embedding-3-small, and stored in multiple vector databases. Each database may represent a different knowledge domain or data source (e.g., ChromaDB for general knowledge).

2. **Query Generation:** The user's original question is sent to the LLM, which processes it and generates a JSON output containing three related questions. This step is crucial as it allows the model to cover different angles or aspects of the original query, thus broadening the scope of the information retrieval process. An example of the generated JSON might look like this:



~~~
```json
{
  "questions": [
    "Can you explain the concept of machine learning?",
    "What are the key principles and techniques involved in machine learning?",
    "How does machine learning differ from traditional programming methods?"
  ]
}
```
~~~

Figure 4.7: Generated JSON file with three example questions

In this step, the LLM generates three distinct questions, stored as question[0], question[1], and question[2]. These questions are designed to capture various dimensions of the user's query, ensuring a more comprehensive retrieval of information.

3. **Multi-Source Context Retrieval:** Each of the three generated questions is embedded and sent individually to the vector databases. The retrieval model, such as ChromaDB, conducts a similarity search for each question, retrieving the most relevant document chunks based on their similarity to the query. The results from these queries are retrieved in order of similarity, ensuring that the most relevant information is prioritized.

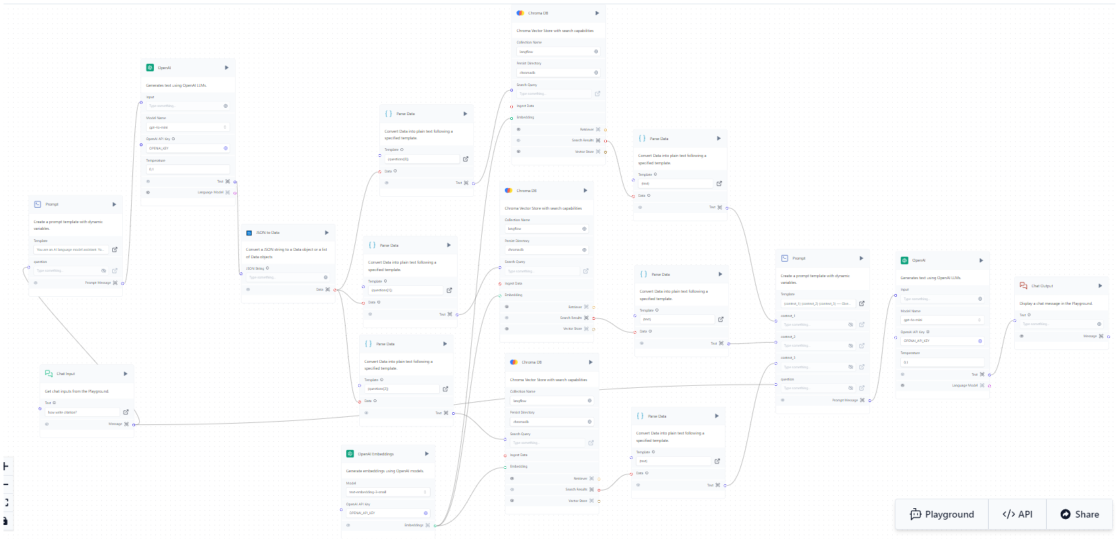

Prompt
Create a prompt template with dynamic variables.
Chat Input
Get chat inputs from the Playground.
how write citation?
OpenAI
Generates text using OpenAI LLMs.
gpt-4o-mini
JSON to Data
Convert a JSON string to a Data object or a list of Data objects
Parse Data
Convert Data into plain text following a specified template.
{questions[0]}
{questions[1]}
{questions[2]}
{text}
Chroma DB
Chroma Vector Store with search capabilities
langflow
chromadb
OpenAI Embeddings
Generate embeddings using OpenAI models.
text-embedding-3-small
Chat Output
Display a chat message in the Playground.
Playground
API
Share

Figure 4.8: Multi-RAG querying process: Generated questions are sent to ChromaDB, which retrieves relevant document chunks based on similarity.

4. **LLM Processing:** The LLM processes the original user input along with

the retrieved context from the vector databases. Langflow ensures a seamless interaction between the retrieval process and the response generation. Additionally, users can adjust settings like the model temperature to control the variability of the generated response.

5. **Response Generation:** Using both its pre-trained internal knowledge and the external information retrieved from the databases, the LLM generates a final response. This ensures that the answer is both accurate and enriched with detailed, contextually relevant content from multiple sources.

6. **Final Output:** The final response is displayed in the Langflow Playground interface, where users can view the retrieved contexts from each database and monitor how the information influenced the final response.

4.3.3 Key Characteristics

The Multi-RAG approach offers several key advantages:

- **Expanded Knowledge Base:** By retrieving information from multiple sources, the model can access a broader and more diverse range of information, leading to more comprehensive responses.

- **Multi-Domain Retrieval:** The ability to query different databases allows the model to handle complex or multi-domain questions more effectively, providing answers that span various fields.

- **Improved Answer Accuracy:** Integrating knowledge from multiple sources enhances the precision and depth of the model's responses, particularly for specialized or complex topics.

- **No-Code Setup:** Langflow provides a no-code environment, making it easy for users to configure the Multi-RAG pipeline, adjust parameters like context length and model temperature, and integrate multiple databases without needing to write code.

In conclusion, Multi-RAG expands on the Simple-RAG model by utilizing multiple sources of external information, improving the model's ability to generate accurate, diverse, and contextually rich responses.

Chapter 5

Experimental Results

This chapter presents the experimental results obtained using the SQuAD v2.0 dataset [11], with a focus on evaluating the performance of three retrieval-augmented generation (RAG) models: NO-RAG, Simple-RAG, and Multi-RAG. The experiments were conducted using two different large language models (LLMs): ChatGPT4o-mini and Gemini-1.0-pro.

The goal of these experiments is to assess the models' performance in answering a predefined set of questions based on the context from the Intergovernmental Panel on Climate Change (IPCC) Wikipedia entry. The evaluation includes several key metrics: RAGAS Answer Correctness, RAGAS Answer Relevancy, and response time. Additionally, Langwatch was used to visualize the execution pipeline for debugging and monitor various metrics, such as latency, token usage, and query cost. These evaluations offer insights into the models' accuracy, relevance, and efficiency in question-answering tasks.

5.1 Dataset Description

The dataset used for these experiments is the Stanford Question Answering Dataset (SQuAD) v2.0, a widely used benchmark for evaluating question-answering systems. SQuAD v2.0 differs from v1.1 in that it includes questions that do not have answers in the text, allowing for a more realistic evaluation of models' abilities to handle *no answer* cases.

For these experiments, we used the passage on the Intergovernmental Panel on Climate Change (IPCC) [3] and a set of ten questions related to that passage. This setup allowed us to evaluate models on both answerable questions and those where no clear answer could be derived from the passage, adding complexity to the task.

5.1.1 Preprocessing Steps

To standardize input across the different models, the following preprocessing steps were applied:

- **Tokenization:** Both the questions and the IPCC passage were tokenized using subword tokenization methods compatible with the selected LLMs (ChatGPT4o-mini and Gemini-1.0-pro).

- **Chunking:** The passage was split into manageable text chunks of 600 tokens, with an overlap of 100 tokens to ensure context retention between chunks.

- **Embedding:** The chunks were embedded using the embedding-3-small model to maintain uniformity across vector databases. These embeddings were stored in ChromaDB for efficient retrieval during the experiments.

- **Answer Matching:** Each question was paired with its ground truth answer (when applicable) for evaluation purposes.

5.2 Experimental Setup

The experiments were designed to evaluate the performance of the NO-RAG, Simple-RAG, and Multi-RAG models using two different base language models: Chat-GPT4o-mini and Gemini-1.0-pro. The evaluation focused on two key metrics: the RAGAS Answer Correctness, which compares the model-generated answers to the ground truth from the dataset, and the response times for each model configuration.

5.2.1 List of Questions and Expected Answers

Below are the ten questions and their expected answers (ground truth) from the dataset:

1. **Who is the vice-chair of the IPCC?**

   Ground Truth Answer: Ismail El Gizouli

2. **What has successfully dealt with ozone depletion?**

   Ground Truth Answer: the Montreal Protocol

3. **For how many years had temperatures been studied in the 2001 report?**

   Ground Truth Answer: ¡No Answer¿

4. **When did Barton and Whitfield demand climate research records?**

   Ground Truth Answer: 23 June 2005

5. **Who wrote the paper that the *Millennial Northern Hemisphere temperature reconstruction* graph was based on?**

   Ground Truth Answer: Michael E. Mann, Raymond S. Bradley and Malcolm K. Hughes

6. **How much did the statement predict global surface temperature would increase by 2100?**

   Ground Truth Answer: between 1.4 and 5.8°C above 1990 levels

7. **How many organizations issued the joint statement on climate change?**

   Ground Truth Answer: 16 national science academies

8. **When was the IPCC Trust Fund founded?**

   Ground Truth Answer: 1988

9. **Who is on the IPCC Panel?**

   Ground Truth Answer: representatives appointed by governments and organizations

10. **What organization is the IPCC a part of?**

    Ground Truth Answer: the United Nations

5.3 Evaluation and Metrics

5.3.1 RAGAS Answer Correctness

The RAGAS Answer Correctness metric evaluates how accurately each model answers the given questions. For each question, the model's generated answer is compared against the ground truth. The similarity score is computed using the cosine similarity between the embeddings of the ground truth and the generated response.

For questions with no answer, the model should ideally respond with *no answer.* Correctness is determined based on the similarity between the model's output and the expected result.

To measure the correctness of the answers, we use the F1 Score, which balances precision and recall. The F1 Score is given by the formula[9]:

$$\text{F1 Score} = \frac{|\text{TP}|}{|\text{TP}| + 0.5 \times (|\text{FP}| + |\text{FN}|)}$$

where:

- |TP| is the number of true positives, i.e., correctly identified answers.
- |FP| is the number of false positives, i.e., incorrect or redundant answers.
- |FN| is the number of false negatives, i.e., missed correct answers.

The F1 Score provides a harmonic mean of precision and recall, giving a single metric to evaluate the model's performance in terms of answer correctness.

Analysis of RAGAS Answer Correctness Scores:

Table 5.1: Average RAGAS Answer Correctness Scores

Model	**Chat-GPT4o-mini**	**Gemini-1.0-pro**
NO-RAG	0.25	0.54
Simple-RAG	0.43	0.76
Multi-RAG	0.46	0.78

- **NO-RAG:** This model has the lowest scores with 0.25 for Chat-GPT4o-mini and 0.54 for Gemini-1.0-pro. The results suggest that the NO-RAG model struggles with accuracy, although it performs significantly better when paired with Gemini-1.0-pro.
- **Simple-RAG:** Performance improves with this model, showing scores of 0.43 for Chat-GPT4o-mini and 0.76 for Gemini-1.0-pro. The results highlight the effectiveness of the retrieval mechanism and the ability of Gemini-1.0-pro to leverage retrieved information.

- **Multi-RAG:** Achieving the highest scores of 0.46 (Chat-GPT4o-mini) and 0.78 (Gemini-1.0-pro), Multi-RAG demonstrates superior performance, particularly when using the more advanced Gemini-1.0-pro model.

Language Model Comparison:

Across all models, Gemini-1.0-pro consistently outperforms Chat-GPT4o-mini, with higher RAGAS scores indicating stronger language understanding and more accurate answers.

Implications:

The results demonstrate that Multi-RAG combined with Gemini-1.0-pro is the most effective configuration for generating correct answers. For applications where accuracy is critical, this combination should be prioritized.

5.3.2 RAGAS Answer Relevancy

The RAGAS Answer Relevancy metric evaluates how relevant the generated answers are to the given questions. Higher scores indicate that the generated answers are more relevant to the original prompts, while lower scores suggest incomplete or redundant responses. This metric is calculated by comparing the original question with several artificially generated questions derived from the model's answer.

The *Answer Relevancy* score is defined as the *mean cosine similarity* between the embedding of the original question and the embeddings of artificially generated questions, which are reverse-engineered based on the generated answer[10]:

$$\text{answer relevancy} = \frac{1}{N} \sum_{i=1}^{N} \cos(E_{g_i}, E_o)$$

where:

- E_{g_i} is the embedding of the i-th generated question.
- E_o is the embedding of the original question.
- N is the number of generated questions.

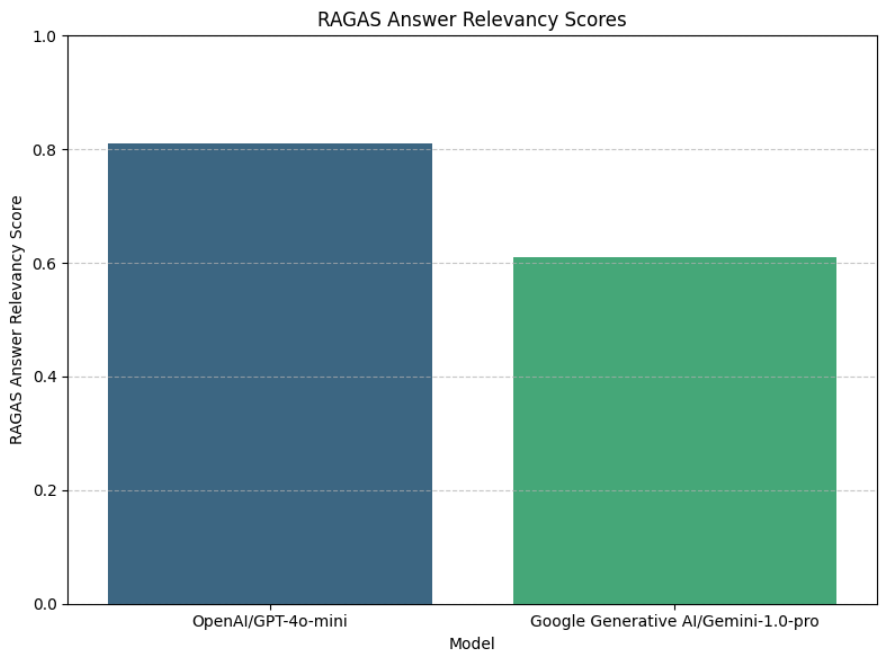


Figure 5.1: RAGAS Answer Relevancy Scores

Relevancy Scores and Interpretation: The plot above shows the RAGAS Answer Relevancy scores for the two models tested:

- **GPT-4o-mini** achieves a score of 0.81, indicating high relevancy in the generated answers. This model excels in providing responses that are closely aligned with the given prompts, reflecting its robust understanding and generation capabilities.
- **Gemini-1.0-pro** scores 0.61. Although this score is lower compared to OpenAI/GPT-4o-mini, it still represents a good level of relevancy. The lower score suggests that while Gemini-1.0-pro performs well, there may be some room for improvement in generating responses that are as pertinent to the prompts as those generated by OpenAI/GPT-4o-mini.

In summary, the results highlight that OpenAI/GPT-4o-mini demonstrates superior relevancy in its responses compared to Gemini-1.0-pro. This suggests that while both models are capable of generating relevant answers, OpenAI/GPT-4o-mini provides answers that are better aligned with the questions asked. This insight is valuable for applications requiring high relevancy in responses.

5.3.3 Response Time

The time taken to retrieve and generate answers was recorded for each model and LLM combination. This includes both the time to search for relevant document chunks in the vector database and the time to generate a response using the LLM.

The response time results reveal significant variations among the different models and language models. Here's a detailed analysis:

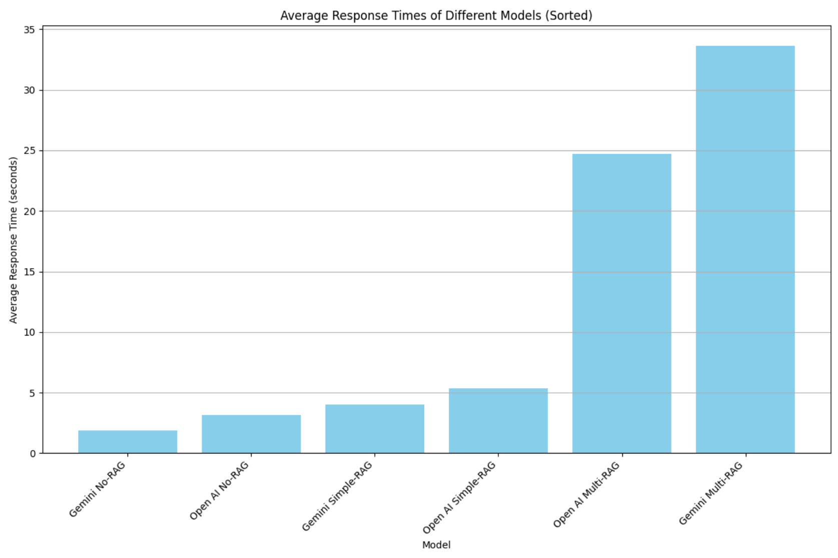


Figure 5.2: Average Response Time for Different Models and LLMs

- **Gemini Multi** has the highest average response time at 33.61 seconds. This longer time is likely due to the complexity of the Multi-RAG approach combined with the Gemini-1.0-pro model, which may involve more extensive retrieval and processing steps.

- **Gemini Single** shows a much faster average response time of 4.00 seconds. This indicates that the Single-RAG model, while less complex than Multi-RAG, still benefits from the advanced capabilities of Gemini-1.0-pro, achieving faster processing times.

- **Gemini No-RAG** has the lowest average response time at 1.86 seconds. The No-RAG model's minimal retrieval mechanism allows it to generate responses more quickly, taking full advantage of Gemini-1.0-pro's efficiency.

- **Open AI Multi** has an average response time of 24.72 seconds, which is considerably faster than Gemini Multi but still relatively high compared to other configurations. This suggests that while OpenAI's multi-RAG model is efficient, it is still more time-consuming than simpler models or those with less complex retrieval mechanisms.

- **Open AI No-RAG** shows an average response time of 3.17 seconds. Similar to Gemini No-RAG, the lack of a retrieval mechanism contributes to faster response times with OpenAI's model.

- **Open AI Simple-RAG** has an average response time of 5.38 seconds. This model offers a balance between complexity and efficiency, with response times faster than Multi-RAG but slower than No-RAG configurations.

5.3.4 Langwatch Integration

Langwatch was utilized to visualize the entire pipeline for each query. This tool provided insights into token usage, the cost of each query, the latency of model responses, and possible bottlenecks in the pipeline. Additionally, Langwatch helped identify errors or inconsistencies in the model flow, ensuring that the system was functioning as expected.

Chapter 6

Conclusions

6.1 Objective of the Thesis

The main objective of this thesis was to evaluate the effectiveness of Retrieval-Augmented Generation (RAG) models in enhancing the accuracy, relevancy, and efficiency of responses generated by advanced language models. The research focused on three RAG variants—NO-RAG, Simple-RAG, and Multi-RAG—paired with two distinct large language models: Chat-GPT4o-mini and Gemini-1.0-pro. The goal was to determine the most effective configurations for improving the performance of question-answering systems.

6.2 Proposed Solution

The proposed solution involved a thorough comparative analysis of the RAG models across multiple metrics, including answer correctness, answer relevancy, and response time. Through extensive experimentation, the aim was to identify which combinations of retrieval models and language models could generate the most accurate and contextually relevant answers while maintaining operational efficiency, particularly in real-time applications.

6.3 Results Obtained

The experimental results yielded the following key insights:

- **RAGAS Answer Correctness:** The Multi-RAG model, when paired with Gemini-1.0-pro, achieved the highest correctness score, demonstrating superior performance in generating accurate responses. This suggests that the Multi-RAG approach, combined with a highly capable language model, is highly effective for precise answer generation.

- **RAGAS Answer Relevancy:** Despite Multi-RAG's lead in accuracy, the Chat-GPT4o-mini model achieved the highest answer relevancy score, indicating that its responses were more contextually aligned with the questions. This points to Chat-GPT4o-mini's strength in producing contextually appropriate answers, even if it lags slightly behind in absolute correctness.

- **Response Time:** There was a noticeable trade-off between accuracy and speed. While simpler models like NO-RAG provided faster response times, they delivered lower accuracy. On the other hand, more complex models such as Multi-RAG were slower due to the retrieval process, but their accuracy was higher. This highlights the need to balance speed and precision based on specific application requirements.

6.4 Limitations and Future Work

The research revealed several limitations that suggest opportunities for future exploration:

- **Accuracy vs. Speed Trade-off:** Multi-RAG models, though accurate, had higher response times. Future work should aim to optimize the retrieval process to reduce latency while maintaining high accuracy, especially for real-time systems.
- **Domain-Specific Performance:** The experiments were conducted within a specific domain (the IPCC entry). Future research should explore how these RAG models perform across different domains and datasets, potentially requiring adjustments in the retrieval strategies.
- **Scalability:** As the size of datasets and the volume of queries increase, ensuring scalability of RAG models becomes crucial. Future studies should investigate techniques to improve the scalability of these models to maintain performance in large-scale environments.

To address these challenges, future research can explore hybrid methods that strike a balance between accuracy and speed, investigate domain-specific fine-tuning of RAG models, and focus on architectural improvements for better scalability.

In conclusion, this thesis has demonstrated that Retrieval-Augmented Generation models can significantly enhance the performance of language models in generating accurate and relevant answers, particularly for specialized tasks. While the results highlight the potential of RAG models, they also underscore the importance of optimizing these approaches to meet the demands of diverse applications and environments, where both accuracy and efficiency are critical factors.

Bibliography

[1] Chat GPT4o-mini by OPEN AI. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

[2] Gemini Pro Technologies. https://deepmind.google/technologies/gemini/pro/.

[3] Intergovernmental panel on climate change. https://rajpurkar.github.io/SQuAD-explorer/explore/v2.0/dev/Intergovernmental_Panel_on_Climate_Change.html?model=nlnet%20(single%20model)%20(Microsoft%20Research%20Asia)&version=v2.0.

[4] Lang Chain. https://www.langchain.com/.

[5] Lang Flow. https://www.langflow.org/.

[6] Lang smith. https://www.langchain.com/langsmith.

[7] Lang Watch. https://docs.langwatch.ai/introduction.

[8] New embedding models and API updates. https://openai.com/index/new-embedding-models-and-api-updates/.

[9] Ragas Answer Correctness. https://docs.ragas.io/en/latest/concepts/metrics/answer_correctness.html.

[10] Ragas Answer Relevancy. https://docs.ragas.io/en/latest/concepts/metrics/answer_relevance.html.

[11] The Stanford Question Answering Dataset (SQuAD). https://rajpurkar.github.io/SQuAD-explorer/.

[12] Vector Store ChromaDB. https://www.trychroma.com/.

[13] Idan A. Blank. What are large language models supposed to model. *Trends in Cognitive Sciences*, 2023.

[14] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. 2023.

[15] Ross Gruetzemacher. The power of natural language processing. *Harvard Business Review Home*, 2022.

[16] Patrick Lewis, Fabio Petroni Ethan Perez, Aleksandra Piktus, Vladimir Karpukhin, Naman Goyal, Wen-tau Yih Heinrich Küttler, Mike Lewis, Tim

Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. 2021.

[17] Laurent Mombaerts, Terry Ding, Florian Felice Adi Banerjee, Jonathan Taws, and Tarik Borogovac. Meta knowledge for retrieval augmented large language models. 2024.

[18] Ye Yuan, Chengwu Liu, Jingyang Yuan, Gongbo Sun, Siqi Li, and Ming Zhang. A hybrid rag system with comprehensive enhancement on complex reasoning. 2024.

Ringraziamenti

Desidero esprimere la mia profonda gratitudine ai miei genitori, che in tutti questi anni mi hanno sempre sostenuto. Anche i gesti più semplici, come un pasto preparato con cura o un aiuto offerto senza che io lo chiedessi, sono stati per me un prezioso sostegno, e di questo li ringrazio immensamente.

Un sentito ringraziamento va alla mia fidanzata, che mi ha accompagnato e supportato nei momenti più difficili, specialmente durante i periodi di burnout causati dalle scadenze pressanti.

Ringrazio anche la mia azienda, che mi ha concesso i permessi necessari per proseguire i miei studi, permettendomi di conciliare lavoro e studio. Un ringraziamento speciale va ad Alessio Cunsolo, che ha creduto in me e nelle mie capacità.

Il mio relatore merita una menzione particolare. La sua guida proattiva e il suo costante supporto fin dal primo giorno di università sono stati per me un punto di riferimento imprescindibile, permettendomi di crescere non solo dal punto di vista accademico, ma anche professionale.

Infine, voglio ringraziare Gabriele Tuccio: senza di lui probabilmente non mi sarei mai iscritto all'università. La sua presenza è stata per me uno stimolo costante a migliorare, una sorta di competizione che mi ha spronato a dare il massimo e a laurearmi nei tempi previsti. Non lo ringrazierò mai abbastanza!