T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

ADAPTIVE FRAMEWORK SUPPORT THROUGH SELF-REFLECTIVE RETRIEVAL-AUGMENTED GENERATION SYSTEM

EREN TORLAK

SUPERVISOR
DR. SALIH SARP

GEBZE
2025

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

ADAPTIVE FRAMEWORK SUPPORT THROUGH SELF-REFLECTIVE RETRIEVAL-AUGMENTED GENERATION SYSTEM

EREN TORLAK

SUPERVISOR
DR. SALIH SARP

2025
GEBZE

GEBZE TECHNICAL UNIVERSITY	GRADUATION PROJECT JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 14/01/2025 by the following jury.

JURY

Member
(Supervisor) : Dr. Salih Sarp

Member : Dr. Salih Sarp

Member : Assoc. Prof. Dr.Mehmet Göktürk

ABSTRACT

The project ”Adaptive Framework Support through Self-Reflective Retrieval-Augmented Generation System” addresses critical challenges in the application of large language models (LLMs) for complex tasks requiring precise retrieval and generation of information. Central to the project is the integration of Self-RAG (Self-Reflective Retrieval-Augmented Generation), a novel framework designed to improve the performance of LLMs in accessing internal knowledge bases, performing dynamic web searches, and generating contextually accurate outputs.

A significant focus of this work is solving the chunking problem inherent in managing datasets comprising structured code and unstructured documentation. Arbitrary splitting of content often leads to the loss of logical coherence or inefficiency in retrieval processes. To address this, the project implements advanced summarization techniques that preserve context, enhance metadata indexing, and ensure retrieval of accurate information.

The project incorporates dynamic routing mechanisms, enabling decision-driven workflows that adapt to user queries. Self-evaluation processes, including the grading of document relevance and assessment of output grounding, allow the system to refine its operations iteratively, ensuring high-quality results. Practical evaluations demonstrate the system's utility in diverse applications such as software development.

Keywords: Self-RAG, Chunking Problem

ÖZET

Proje, ”Adaptive Framework Support through Self-Reflective Retrieval-Augmented Generation System”, büyük dil modellerinin (LLM’ler) bilgi getirme ve üretim gerektiren karmaşık görevlerdeki kullanımında karşılaşılan zorlukları ele almaktadır. Projenin temelinde, Self-RAG (Kendini Yansıtan Bilgi Getirme Destekli Üretim) adı verilen yenilikçi bir çerçevenin entegrasyonu yer almaktadır. Bu çerçeve, LLM’lerin iç bilgi tabanlarına erişimini, dinamik web aramaları yapmasını ve bağlamsal olarak doğru çıktılar üretmesini geliştirmeyi amaçlamaktadır.

Çalışmanın önemli bir odağı, yapılandırılmış kod ve yapılandırılmamış dokümantasyon gibi veri kümelerinde karşılaşılan parçalama sorununu (chunking problem) çözmektir. İçeriğin rastgele bölünmesi, mantıksal bütünlüğü bozabilir veya bilgi getirme süreçlerini verimsiz hale getirebilir. Bu sorunu çözmek için proje, bağlamı koruyan, meta veri indekslemeyi geliştiren ve doğru bilgilerin getirilmesini sağlayan gelişmiş özetleme teknikleri kullanmaktadır.

Proje ayrıca, kullanıcı sorgularına uyum sağlayan dinamik yönlendirme mekanizmaları (dynamic routing mechanisms) ve karar odaklı iş akışlarını içermektedir. Öz değerlendirme süreçleri (self-evaluation processes), belgelerin uygunluğunun derecelendirilmesi ve çıktıların dayanağının değerlendirilmesini kapsayarak, sistemin işlemlerini yinelemeli bir şekilde iyileştirmesini sağlar. Pratik değerlendirmeler, özellikle yazılım geliştirme gibi alanlarda sistemin faydasını göstermektedir.

Anahtar Kelimeler: Self-RAG, parçalama sorunu

ACKNOWLEDGEMENT

I sincerely thank my supervisor, Dr. Salih Sarp, for his guidance and support.

Eren Torlak

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	Explanation
LLM	Large Language Model, an AI-based model trained on vast textual datasets for natural language processing tasks.
RAG	Retrieval-Augmented Generation, a framework that combines information retrieval with text generation.
Self-RAG	Self-Reflective Retrieval-Augmented Generation, the framework developed in this project for adaptive and accurate outputs.
API	Application Programming Interface, a set of definitions and protocols for building and integrating application software.
Chunking Problem	The challenge of dividing structured code and unstructured documentation into manageable yet coherent pieces.
LangChain	A framework used for orchestrating workflows in LLM-based applications.
LangGraph	A tool for managing stateful processes and dynamic workflows with branching and looping capabilities.
Vector Database	A database that stores documents as vector embeddings for efficient retrieval based on semantic similarity.
Metadata Indexing	The process of organizing and tagging data with relevant metadata for improved retrieval efficiency.
Dynamic Routing	A mechanism to adapt workflows based on the specific requirements of user queries.

CONTENTS

Bibliography 19

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The project ”Adaptive Framework Support through Self-Reflective Retrieval-Augmented Generation System” aims to address the critical challenges of utilizing large language models (LLMs) for complex tasks requiring precise information retrieval and contextual generation. This chapter introduces the motivation behind the project, the key problems it aims to solve, and the objectives and significance of the proposed framework.

1.1. Background and Motivation

In recent years, the field of artificial intelligence has witnessed remarkable advancements, especially in natural language processing (NLP) and large language models (LLMs). Models such as OpenAI's GPT-4 have showcased extraordinary capabilities in generating human-like text, performing complex reasoning, and assisting in coding and problem-solving tasks. Despite these impressive feats, LLMs often fall short when required to deliver precise, context-specific, and domain-relevant information. This limitation is particularly evident in scenarios involving extensive technical documentation, codebases, or APIs.

Standalone LLMs primarily rely on their pretraining data, creating a bottleneck when users demand highly specific information or require integration with external knowledge sources. These challenges become particularly significant in software development workflows, where developers spend considerable time navigating disparate documentation repositories or interpreting incomplete materials.

To address these limitations, retrieval-augmented generation (RAG) systems have emerged as a promising solution. By enabling LLMs to dynamically access external knowledge bases or perform web searches, RAG systems enhance the capabilities of LLMs in solving complex information retrieval tasks. This project, motivated by the demand for robust, adaptive, and self-improving systems, introduces the Self-Reflective Retrieval-Augmented Generation (Self-RAG) framework. By incorporating a self-corrective feedback loop into the RAG process, the framework ensures iterative refinement of retrieval strategies and generated responses, making it particularly suited for domains like software engineering and academic research.

1.2. Problem Statement

The application of LLMs in real-world workflows is hindered by a fundamental issue: the ”chunking problem.” Managing datasets that combine structured code and unstructured documentation requires precise segmentation for efficient indexing and retrieval. Structured code must retain logical integrity, while unstructured documentation demands careful preservation of context. Arbitrary chunking often leads to disrupted coherence, inefficiencies, and reduced retrieval accuracy.

Existing solutions often apply fixed heuristics for chunking, which fail to address the nuanced requirements of specific tasks. For example, splitting a function into multiple segments can render it meaningless, while overly small chunks of documentation may lack sufficient context for effective retrieval. These issues compound the inefficiencies of indexing such data, resulting in irrelevant or generalized responses.

Additionally, current RAG systems lack robust mechanisms for evaluating the relevance and grounding of retrieved documents. This limitation frequently leads to hallucinated content—syntactically correct but factually unsupported outputs. Such unreliability poses significant risks, especially in critical domains like software development and healthcare. Without a self-reflective mechanism to adapt retrieval strategies and improve output quality, existing systems are insufficient for addressing these challenges.

1.3. Project Aims and Objectives

The overarching aim of this project is to develop an adaptive framework that integrates Self-RAG to enhance LLM performance in retrieving and generating contextually relevant information. The following objectives guide the project:

- **Addressing the Chunking Problem:** Develop advanced summarization techniques to segment structured and unstructured data into manageable yet coherent units. This ensures logical and contextual integrity while improving retrieval efficiency. - **Dynamic Routing Mechanisms:** Design workflows capable of dynamically adapting to user queries, intelligently routing tasks to appropriate data sources or modules for generation. - **Self-Evaluation Processes:** Integrate self-reflective mechanisms to assess the relevance of retrieved documents and the grounding of generated outputs. This iterative refinement process ensures high-quality and accurate results. - **Practical Validation:** Demonstrate the framework's utility through real-world testing in domains such as software development and academic research, establishing benchmarks for its effectiveness.

1.4. Significance and Potential Impact

The successful implementation of the Self-RAG framework holds significant potential to redefine how LLMs are utilized across various domains. By addressing key challenges like the chunking problem and incorporating adaptive, self-reflective mechanisms, this project contributes to the development of more efficient and reliable AI systems.

In software development, the framework can streamline workflows by providing developers with precise, contextually relevant information from extensive codebases and API documentation. This can reduce development time, improve code quality, and minimize errors. Beyond software engineering, the adaptive capabilities of Self-RAG extend to other domains, including healthcare, legal research, and education, where reliable and context-aware information retrieval is crucial.

Ultimately, this project advances the state of the art in retrieval-augmented generation, paving the way for innovative, AI-driven systems that align closely with human needs and expectations. It establishes a foundation for future advancements in information retrieval and generation, ensuring the practical applicability of LLMs in diverse professional environments.

2. SYSTEM DESIGN AND DEVELOPMENT

This chapter elaborates on the design and development process of the ”Adaptive Framework Support through Self-Reflective Retrieval-Augmented Generation System” (Self-RAG). The system was carefully designed to overcome challenges inherent in large language model (LLM) applications, particularly those related to combining structured codebases with unstructured documentation, routing user queries dynamically, and integrating iterative self-reflective mechanisms. These design decisions aim to create a flexible and efficient system capable of producing accurate, contextually grounded, and reliable outputs.

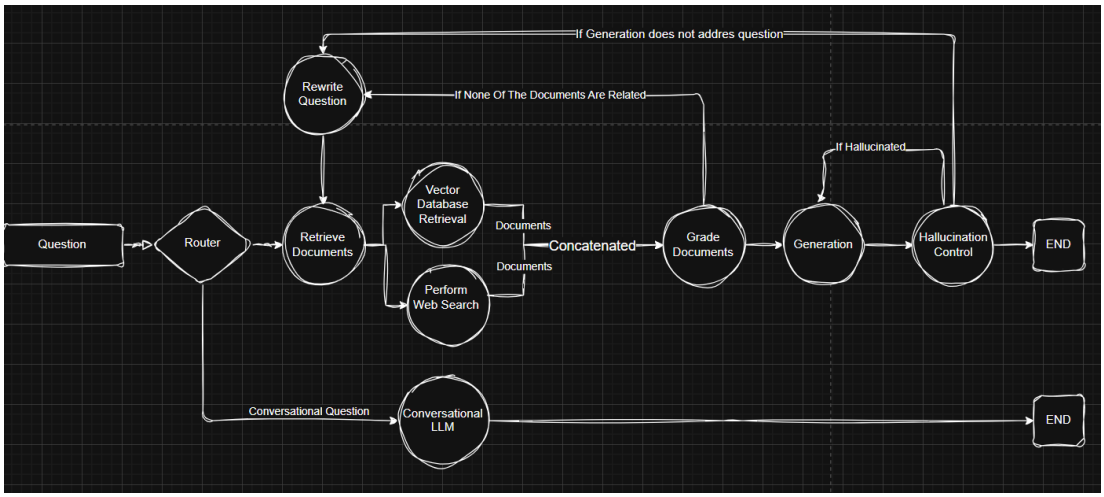

Figure 2.1: Workflow

2.1. Overview of the Methodology

The methodology for this project was structured around the integration of RAG principles with self-reflective mechanisms to create a framework capable of addressing the limitations of traditional LLM implementations.

The project began by analyzing these limitations, including inefficiencies in retrieving specific information, the inability to handle large datasets effectively, and the generation of hallucinated or irrelevant responses. To overcome these challenges, the framework—Self-Reflective Retrieval-Augmented Generation (Self-RAG)—was designed to refine its retrieval strategies iteratively and adapt outputs dynamically based on user queries.

The methodology was implemented in phases. Initially, data acquisition and preprocessing were conducted, focusing on technical documentation, APIs, and code-bases. This phase ensured the creation of a reliable vector database. The subsequent phase emphasized the development of dynamic workflows capable of managing diverse user queries.

The workflow relies on a graph-based design where each query passes through predefined stages, or nodes, that process data and make decisions based on predefined conditions. These nodes are interconnected by edges that represent transitions, allowing the query to flow dynamically based on its context and the results at each stage. This structured approach ensures that the system remains adaptable while maintaining logical coherence.

2.2. Data Processing and Vector Database Creation

A key component of Self-RAG is its ability to efficiently manage and retrieve information from a vector database. Data acquisition targeted relevant sources, including online documentation repositories and API references. A ”RecursiveUrlLoader” was employed to systematically extract content from predefined URLs. This tool ensured crawling nested pages while preserving their logical structure.

The preprocessing phase included preserving structured data, such as code snippets, in its original format and summarizing unstructured documentation and code for improved retrievability.

Using the GPT-4o model, each document was analyzed to generate concise summaries (5–7 sentences) and a list of potential user queries. These enhancements are required to ensure that the system retrieves relevant and contextually accurate information when queried. The generated summaries and query examples encapsulate the essence of each document, reducing the computational load during retrieval and enhancing the semantic representation of documentation pages. By summarizing entire documents rather than arbitrarily splitting them, the system preserves the logical integrity of data.

Metadata, such as source URLs, titles, and descriptions, was combined with the generated summaries and queries to create enriched entries. These were stored in a Chroma vector database, where embeddings were generated using the OpenAI "text-embedding-3-large" model. The database supported semantic searches, enabling precise retrieval of contextually relevant results. Persistent storage ensured data consistency and scalability, accommodating updates without reprocessing.

```
# Define LLM schema for structured outputs
You, 9 hours ago | 1 author (You)
class CreateSummary(BaseModel):
    """Summary and possible queries for a document."""
    summary: str = Field(description="Summary of the document content in 5-7 sentences.")

    possible_queries: List[str] = Field(
        description="List of possible queries that users might ask about this topic."
    )

# Initialize ChatOpenAI with structured output
llm = ChatOpenAI(model="gpt-4o", temperature=0)
structured_llm_router = llm.with_structured_output(CreateSummary)

Run Cell | Run Above | Debug Cell
#%%
# Create Prompt Template
system = """You are an expert at technical documentation.
Analyze the given documentation and provide:
1. A concise summary (5-7 sentences)
2. List of 5-7 possible queries that users might ask about this topic.
Use the document to generate the summary and possible queries.
"""
summary_prompt = ChatPromptTemplate.from_messages(
    [("system", system), ("human", "Document: \n\n {document} \n\n ")]
)
question_router = summary_prompt | structured_llm_router
```

Figure 2.2: Representation of documentation pages

2.3. User Query Routing

The dynamic workflow in the system is designed to handle two main query cases: retrieval workflows and conversational language model workflows. The routing mechanism decides which path to follow based on the nature of the query.

If the query requires retrieval, the system engages both the vector database and the web search tool to gather relevant information.

The system retrieves two documents from the database that most closely match the query's semantic content. Simultaneously, the web search tool is activated to perform a real-time search, retrieving two additional documents from external sources. These four documents—two from the vector database and two from the web search—are then concatenated into a single collection. This combined set of documents ensures that the response generation process benefits from both internal knowledge and external information.

If the query is casual, conversational, or a follow-up question, it bypasses the retrieval mechanism entirely and is routed directly to the conversational language model. Examples of such queries include informal statements like "Can you explain that?" or "What do you mean by this?" In these cases, the conversational model generates responses using the context of the ongoing interaction, without involving the vector database or web search. This ensures smooth and efficient handling of informal queries while keeping the system's resources optimized.

2.4. Self-Reflective Evaluation

Self-reflective evaluation is a fundamental component of the Self-RAG framework, designed to enhance the accuracy and reliability of the system's outputs. By iteratively assessing retrieved documents and generated responses, this mechanism ensures that the system adapts dynamically to user queries while minimizing errors, hallucinations, and irrelevance. This evaluation process is seamlessly integrated into the graph-based workflow, allowing feedback to refine outputs and guide the system through multiple stages of improvement.

Once a user query is processed and routed to the retrieval workflow, the system begins by gathering documents from both the vector database and the web search tool. The vector database contains structured and summarized internal knowledge, while the web search tool provides external sources to supplement this information.

After retrieval, the system evaluates the relevance of each document using a grading mechanism. This mechanism employs a binary scoring model, which assesses whether a document is relevant (yes) or irrelevant (no) to the query. The relevance scoring is based on a prompt-guided process, ensuring that both semantic alignment and keyword relevance are considered. Irrelevant documents are filtered out, preventing them from contributing to the response generation process. This initial grading step is critical to maintaining the quality of the system's outputs, as it ensures that only meaningful and contextually accurate data is used.

Following the document grading process, the remaining relevant documents are passed to the generation module. Here, the system synthesizes a response that addresses the user's query. However, the generation process is not the final stage; the response itself undergoes a rigorous evaluation to verify its quality. The first step in this evaluation is a grounding check, which determines whether the generated response is supported by the retrieved documents. Using a binary scoring model, the system evaluates whether the response is factually accurate and directly aligned with the evidence provided in the documents. If the response fails this grounding check, it is flagged as unsupported, and the system initiates a corrective process.

When a response is flagged as unsupported, the system uses the feedback to refine its operations. This may involve rewriting the query to improve its semantic alignment with the stored data or revisiting the retrieval phase to include additional or alternative documents. These iterative adjustments allow the system to explore different avenues to resolve the query, reducing the likelihood of producing hallucinated or inaccurate outputs. This iterative loop ensures that the system continues to refine its understanding of the query and the information needed to address it.

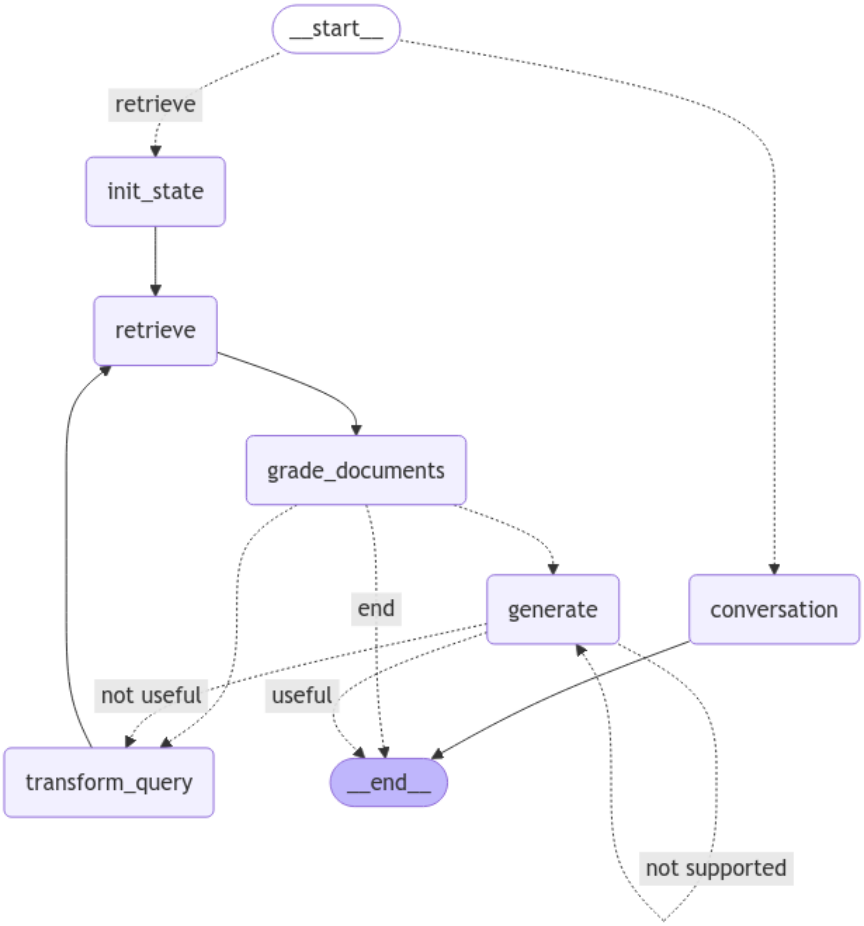

Figure 2.3: Graph representation of system

In addition to grounding, the system also evaluates whether the generated response sufficiently addresses the user's question. This evaluation is performed using another binary scoring model, which determines whether the response resolves the query (yes) or fails to provide a satisfactory answer (no). If the response is deemed insufficient, the workflow redirects to the retrieval or query refinement phase, allowing the system to improve its results. This iterative process continues until the system produces a response that meets both grounding and relevance criteria or reaches a predefined limit on refinement attempts.

The self-reflective evaluation mechanism provides several key benefits. It significantly reduces the likelihood of errors by iteratively refining each stage of the process. It minimizes the risk of hallucinated or irrelevant outputs by ensuring that all responses are grounded in factual and contextually relevant information. By incorporating multiple layers of feedback and refinement, the system demonstrates adaptability and resilience, even when handling complex or ambiguous queries. This iterative approach enhances the reliability of the framework, making it suitable for applications requiring high levels of precision, such as software development

```
---ROUTE QUESTION---
---ROUTE QUESTION TO RETRIEVER---

---INIT STATE---
---VECTOR DATABASE RETRIEVE---
---WEB SEARCH---
---CHECK DOCUMENT RELEVANCE TO QUESTION---
---GRADE: DOCUMENT RELEVANT---
---GRADE: DOCUMENT NOT RELEVANT---
---GRADE: DOCUMENT RELEVANT---
---GRADE: DOCUMENT RELEVANT---
---ASSESS GRADED DOCUMENTS---
---DECISION: GENERATE---
---GENERATE---
---CHECK HALLUCINATIONS---
---DECISION: GENERATION IS GROUNDED IN DOCUMENTS---
---GRADE GENERATION vs QUESTION---
---DECISION: GENERATION ADDRESSES QUESTION---
```

Figure 2.4: Inference example

2.5. System Architecture

The system architecture was designed to be modular and scalable, consisting of the following layers:

- Data Layer: Hosted the vector database containing enriched documents and embeddings. This layer supported efficient retrieval using semantic queries.

- Processing Layer: Managed data preprocessing, embedding generation, and query routing. LangChain and LangGraph enabled dynamic workflow management.

- Evaluation Layer: Implemented the self-reflective evaluation process, iteratively refining results through document grading and output grounding assessment.

- Interface Layer: Provided a user-friendly interaction platform via Gradio, supporting query submission, response streaming, and iterative refinements.

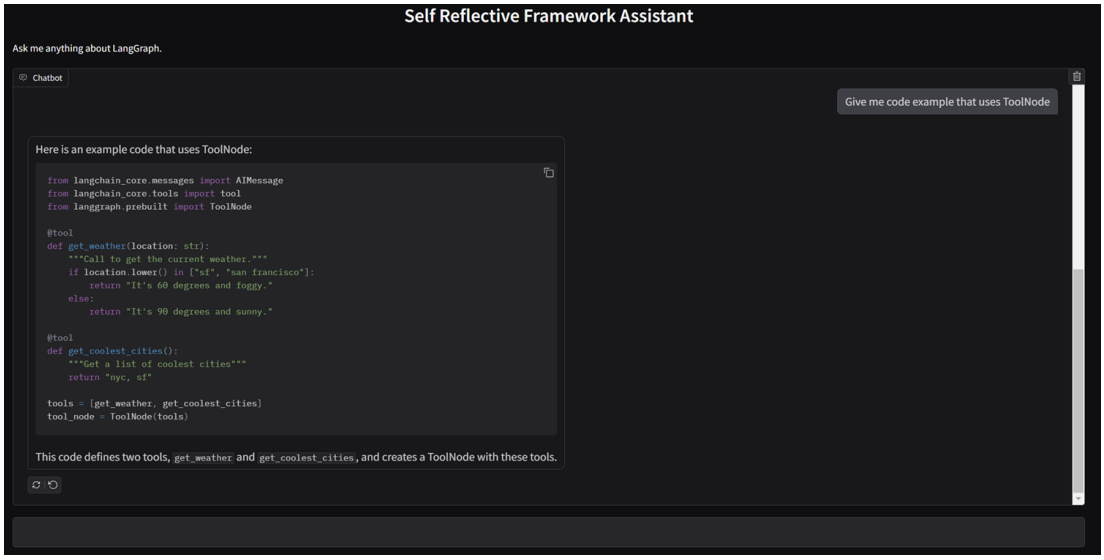

Figure 2.5: Interface

3. CONCLUSIONS

The conclusion provides an in-depth overview of the evaluation results, contributions, limitations, and potential future directions for the Self-Reflective Retrieval-Augmented Generation (Self-RAG) framework. The framework's innovative design demonstrates its ability to address complex information retrieval challenges and highlights its potential for further advancements.

3.1. Summary of Findings

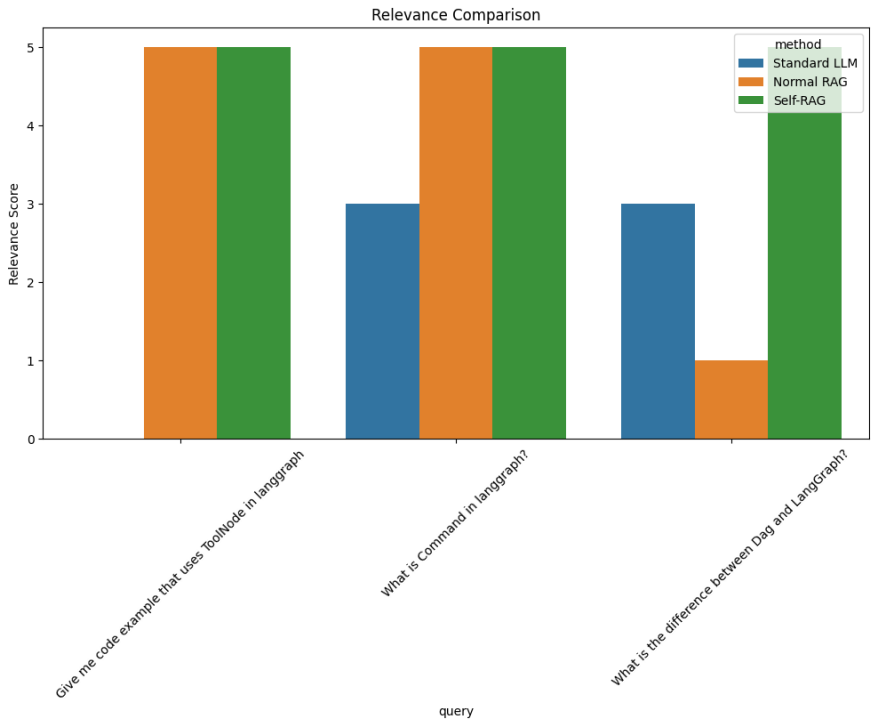

Figure 3.1: Relevance

The evaluation of Self-RAG against standard LLMs and traditional RAG systems revealed its superior performance in relevance and accuracy. This evaluation done by using a LLM with provided context.Self-RAG achieved the highest average relevance score of 5.0, significantly outperforming standard LLMs (2.0) and normal RAG systems (3.67). This demonstrates its consistent ability to provide responses that are directly aligned with user queries and supported by relevant evidence.

Detailed query-level analysis reinforced these findings. For example, when asked for code examples involving ToolNode in LangGraph, Self-RAG retrieved and synthesized accurate, actionable responses supported by detailed code snippets. Similarly, its explanations of conceptual topics like the role of Commands in LangGraph and the differences between LangGraph and directed acyclic graphs (DAGs) were both comprehensive and well-grounded. The system effectively leveraged its dual-retrieval mechanism, combining documents from the vector database and web search, to deliver nuanced and contextually appropriate answers.

While Self-RAG demonstrated higher latency (14.38 seconds on average), this was offset by its ability to produce high-quality responses that far exceeded the capabilities of the other systems. This trade-off underscores the framework's emphasis on precision and reliability, which are critical for tasks requiring domain-specific knowledge and accuracy.

Another transformative aspect of the project was the integration of a self-reflective evaluation module. This mechanism enabled the framework to iteratively assess the relevance of its retrieved documents and the grounding of its generated outputs, ensuring more reliable and contextually appropriate results.

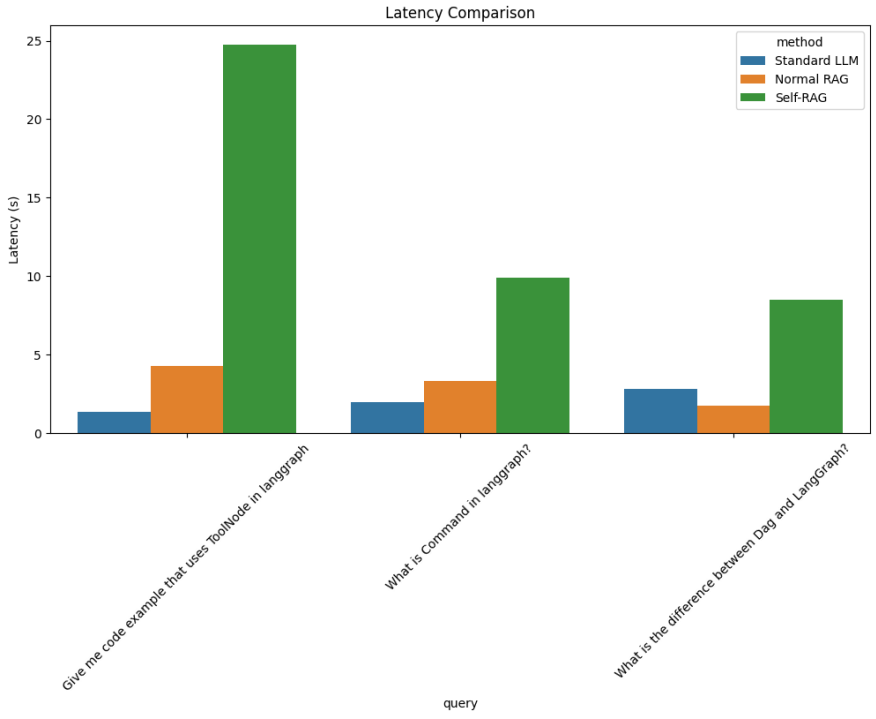

Figure 3.2: Latency Comparision

3.2. Contributions and Significance

The project demonstrated the application of advanced summarization techniques to solve the chunking problem. This innovative approach improved data indexing and retrieval efficiency when dealing with hybrid datasets.

Its integration of self-reflective evaluation mechanisms addresses a common challenge in LLM applications: the generation of hallucinated or irrelevant responses. By iteratively assessing retrieved documents and generated outputs for relevance and grounding, the framework ensures that its responses are both accurate and contextually aligned.

Second, the dual-retrieval mechanism, combining vector database queries and web searches, enhances the system's ability to address diverse queries. This design ensures comprehensive coverage by drawing on both internal knowledge repositories and external information sources. The adaptive workflow, which dynamically refines queries and iterates through retrieval and generation processes, further strengthens the framework's versatility.

3.3. Limitations

The most notable is its relatively high latency, which averages 14.38 seconds per query. This latency results from the iterative nature of the self-reflective evaluation process, which includes multiple grading, grounding, and refinement steps. While this trade-off is acceptable in domains where accuracy is paramount, it may limit the framework's applicability in scenarios where speed is a critical factor.

Another limitation lies in the dependency on high-quality input data for the vector database. The framework's performance relies heavily on the accuracy and completeness of its internal knowledge base. Any deficiencies in the preprocessing phase, such as inadequate summarization of documentation could impact the relevance and quality of retrieved results.

3.4. Future Work

Future enhancements to the Self-RAG framework will focus on reducing latency through parallel query processing and optimizing grading mechanisms. The preprocessing phase can be improved with advanced summarization techniques and more effective query generation, ensuring higher retrieval precision.

Improving the user interface is another priority. Features like displaying references with links or page numbers for retrieved documents will increase transparency and usability. These improvements aim to make the system faster, more functional, and user-friendly while broadening its applicability.

BIBLIOGRAPHY

[1] Self-RAG Documentation. Available at: https://selfrag.github.io/, Accessed on: January 14, 2025.

[2] Advanced RAG Techniques. Available at: https://arxiv.org/pdf/2310.11511, Accessed on: January 14, 2025.

[3] Blog: Advanced RAG with Self-Correction and LangGraph. Available at: https://blog.gopenai.com/advanced-rag-with-self-correction-langgraph-no-hallucination-agents-groq Accessed on: January 14, 2025.

[4] Blog: Building an Effective RAG Pipeline. Available at: https://blog.gopenai.com/building-an-effective-rag-pipeline-a-guide-to-integrating-self-rag-corre Accessed on: January 14, 2025.

[5] LangGraph RAG Example Notebook. Available at: https://github.com/langchain-ai/langgraph/blob/main/examples/rag/langgraph_self_rag.ipynb, Accessed on: January 14, 2025.

[6] LangSmith Tools. Available at: https://smith.langchain.com/, Accessed on: January 14, 2025.

[7] Gradio Chat Interface Documentation. Available at: https://www.gradio.app/docs/gradio/chatinterface, Accessed on: January 14, 2025.

APPENDICES

GitHub Repository

https://github.com/erentorlak/Self-Reflective_Framework_Assistant_RAG