A Retrieval Augmented Generation System for Factual Question Answering on Vietnam National University

Hoàng Việt Tùng
22022663

Lèng Hữu Phúc
22022570

Phạm Văn Trường
22022564

Long Hoàng Vinh
22022673

I. Introduction

A. Project Overview

This report details the development and evaluation of a Retrieval Augmented Generation (RAG) system designed for factual question-answering. The primary challenge addressed is the limitation of existing general-purpose question-answering systems to provide accurate and comprehensive information within specialized domains. This project [1] focuses on creating a RAG system specifically knowledgeable about two distinct yet significant areas: the comprehensive details of universities within Vietnam National University (VNU) and information pertaining to notable Vietnamese public figures. By leveraging the RAG architecture, which combines the strengths of information retrieval with the natural language capabilities of large language models, this system aims to deliver precise answers based on a curated collection of relevant documents.

B. Objective

The primary objectives of this project are as follows:

- To meticulously collect and curate a specialized dataset comprising comprehensive information about the universities within Vietnam National University and biographies or relevant details of selected famous Vietnamese individuals.
- To design and implement a Retrieval Augmented Generation (RAG) system, integrating an effective document retriever with a coherent and accurate text generator.
- To rigorously evaluate the performance of the developed RAG system using appropriate metrics for factual question-answering, assessing its ability to provide correct and relevant answers based on the curated knowledge base.
- To adhere to established experimental best practices throughout the data collection, model development, and evaluation phases to ensure the reliability and reproducibility of the findings.

II. Background

A. RAG

Retrieval Augmented Generation (RAG) [2] has emerged as a pivotal technique in Natural Language Processing (NLP) designed to make Large Language Models (LLMs) more knowledgeable, accurate, and contextually aware. It addresses some of the inherent limitations of purely generative models by dynamically incorporating external information into the language generation process. [3].

The development of RAG was primarily driven by the need to overcome several challenges faced by LLMs [4]:

- **Knowledge Cutoffs:** LLMs are trained on vast datasets, but this knowledge is static and fixed at the time of training. They lack access to real-time information or developments that occur after their training period.
- **Hallucinations:** LLMs can sometimes generate plausible-sounding but factually incorrect or nonsensical information, a phenomenon known as "hallucination."
- **Lack of Specificity and Verifiability:** For tasks requiring deep domain-specific knowledge or information that needs to be verifiable, LLMs might not provide sufficiently detailed or accurate responses, and it's often difficult to trace the source of their generated information.
- **Cost and Inflexibility of Retraining:** Constantly retraining massive LLMs with new information is computationally expensive and impractical

To address these issues, researchers sought ways to ground LLMs in external, up-to-date, and verifiable knowledge sources without requiring full model retraining. This led to the development of hybrid approaches that combine the strengths of information retrieval with the advanced generative capabilities of LLMs.

RAG operates on a two-stage process:

- Retrieval: When presented with a prompt or query, the RAG system first employs a retriever component. This retriever searches a large corpus of documents or a knowledge base (e.g., Wikipedia, internal company documents, a specific dataset) to find information relevant to the input query. The retrieved documents or text snippets serve as context.

- Generation: The original prompt and the retrieved contextual information are then fed into a generator component, which is typically an LLM. The LLM uses this augmented input to generate a response. By having access to the relevant external information, the generator can produce answers that are more factual, detailed, and current.

III. Data Collection and Preparation

A. Domain Definition and Scope

Our project focuses on factual question answering related to:

- Vietnam National University (VNU): This includes information about its member universities and colleges, history, academic programs, research activities, facilities, and administrative structure.
- Famous Vietnamese People: This encompasses biographical information, achievements, and significant contributions of notable individuals from various fields in Vietnamese history and contemporary society.

B. Data Source

Given the breadth of information required and the public availability of relevant encyclopedic content, Wikipedia was selected as a primary data source for this project. Specifically, we targeted articles from Vietnamese Wikipedia (vi.wikipedia.org) and English Wikipedia (en.wikipedia.org) to gather comprehensive information on the aforementioned domains. Wikipedia offers a vast repository of structured articles, making it amenable to automated data extraction. The choice of Wikipedia was also influenced by its collaborative editing model, which generally aims for factual accuracy and neutrality, although all information is used as-is from the source.

The initial set of target URLs for crawling was manually compiled and stored in a text file (wiki_source/wiki_urls.txt). This list was curated by identifying relevant Wikipedia pages corresponding to VNU entities and a diverse set of famous Vietnamese individuals.

C. Data Collection Process

An automated web crawling process was implemented using a Python script to retrieve content from the specified Wikipedia URLs. The core components and steps of this process are as follows:

- URL Validation: Each URL from the input file (wiki_urls.txt) was first validated to ensure it was a correctly formatted HTTP/HTTPS URL and belonged to a recognized Wikipedia domain (e.g., wikipedia.org, vi.wikipedia.org). This was handled by the validate_wiki_url function, utilizing the validators library.
- Content Fetching: For each valid URL, the Workspace_content function sent an HTTP GET request using the requests library to retrieve the raw HTML content of the Wikipedia page. Error handling was implemented to manage potential issues like network errors or HTTP error statuses.
- HTML Parsing and Content Extraction: The BeautifulSoup library was employed to parse the fetched HTML. The script specifically targeted:
- Title: The main title of the article, extracted from the tag with id=‘firstHeading’.
- Main Content: Textual data was extracted from paragraph tags () located within the main content division of Wikipedia articles (identified by div with id=‘mw-content-text’). This approach aims to capture the primary informational content of each page while excluding sidebars, navigation menus, and footers.
- Storing Raw Data: The extracted title and a list of raw paragraph texts were temporarily stored before further processing.

D. Data Preprocessing and Cleaning

To prepare the raw text for ingestion into the RAG system, a series of preprocessing and cleaning steps were applied to each extracted paragraph. This was primarily managed by the clean_and_process_text function:

1) Citation Removal: Bracketed citation markers (e.g., [1], [23]) commonly found in Wikipedia articles were removed using regular expressions (re.sub(r’[d+]‘, ‘’, text)). This step cleans the text from inline references that are not part of the core content.
2) HTML Remnant Cleaning: Although BeautifulSoup’s .get_text() method handles most HTML tag removal during the initial extraction, an additional pass with BeautifulSoup(text, “html.parser”).get_text() was performed to ensure any residual HTML entities or tags within the paragraph text were fully stripped.
3) Whitespace Normalization: Multiple consecutive whitespace characters (spaces, tabs, newlines) were collapsed into a single space, and leading/trailing whitespaces were removed (re.sub(r’s+‘, ‘ ‘, text).strip()). This ensures textual consistency.
4) Special Character Filtering: Non-alphanumeric characters were removed, with an explicit exception to preserve Vietnamese characters (e.g., À-ỹà-ỹ) and digits. This was achieved using the regular expression re.sub(r’[^a-zA-ZÀ-ỹà-ỹ0-9s]‘, ‘ ‘, text). This step helps in reducing noise from miscellaneous symbols while retaining the essential linguistic content in both English and Vietnamese.
5) Lowercasing: All text was converted to lowercase to ensure uniformity and prevent the model from treating the same words with different capitalization as distinct tokens.
6) Tokenization (Implicit) and Stop Word Removal: The text was split into tokens (words) using space as a delimiter (text.split()). A custom list of Vietnamese stop words (e.g., “và”, “là”, “của”, “có”) was defined. Tokens matching these stop words were removed from the list of processed tokens. This step aims to reduce the dimensionality of the text and remove high-frequency words that often carry less semantic weight for retrieval purposes.

7) Rejoining Tokens: The processed tokens for each paragraph were then joined back into a single string.

If a paragraph became empty after processing (e.g., if it only contained stop words or citations), it was handled appropriately to avoid empty entries in the processed data.

IV. System Architecture and Model Development

This section details the architecture of our Retrieval Augmented Generation (RAG) system, designed for factual question answering on topics related to Vietnam National University (VNU) and prominent Vietnamese figures. We will describe the overall workflow, the individual components responsible for retrieval and generation, and how they are integrated.

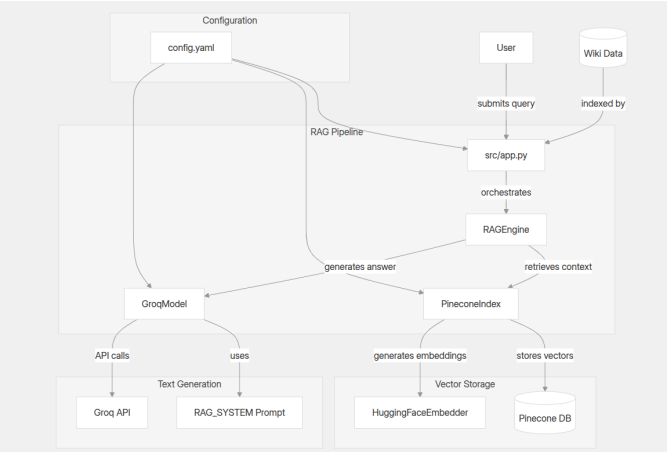

Fig. 1: system architecture

A. Overall System architecture

Our RAG system follows a two-stage pipeline designed to leverage both a static knowledge base and the generative capabilities of a Large Language Model (LLM). The overall workflow is as follows:

1) Query Input: The user poses a question in natural language.
2) Retrieval Stage: The input query is transformed into an embedding. This query embedding is used to search a pre-built vector index (containing embeddings of document chunks/paragraphs from our curated dataset) to find the most semantically similar document chunks. Our app.py script demonstrates this retrieval by fetching the top 5 results for a given query.
3) Context Augmentation: The text content of these retrieved document chunks is then used as supplementary context.
4) Generation Stage: The original query, along with the retrieved contextual information, is formatted into a prompt and fed to a Large Language Model (LLM). (This stage is not implemented in the provided app.py but is a required part of the RAG system for your project).
5) Answer Output: The LLM generates a natural language answer based on both its internal knowledge and, critically, the provided contextual documents.

This architecture aims to ground the LLM's responses in factual information retrieved from our specific domain dataset, thereby enhancing accuracy and reducing the likelihood of hallucinations

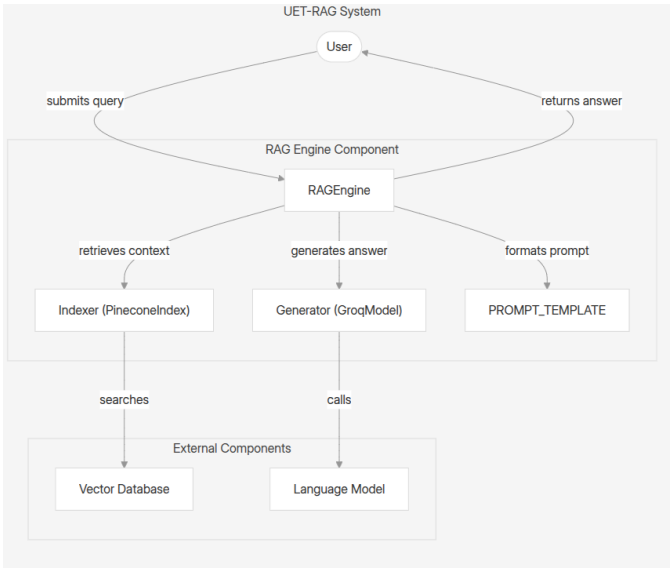

Fig. 2: Rag system architecture

The system comprises two main components: the Retriever and the Generator.

B. Retriever Component

a) *Embedding Model::* To enable semantic search, the textual content of these document chunks and the input queries were transformed into numerical vector representations (embeddings).

We utilized the sentence-transformers library and implemented a custom HuggingFaceEmbedder class. The specific pre-trained model chosen, as specified in our project configuration (config.yaml), was BAAI/bge-m3.

The HuggingFaceEmbedder class initializes the BAAI/bge-m3 model, automatically leveraging a CUDA-enabled GPU if available for efficiency.

The encode method within HuggingFaceEmbedder converts text (document chunks or queries) into 1024-dimensional dense vector embeddings.

b) *Indexing Mechanism / Vector Store:* We selected Pinecone as our managed vector database service to store these embeddings and enable fast similarity searches.

- The upsert_texts method in PineconeIndex handles:
- Simple preprocessing (filtering very short texts).
- Generating embeddings for document chunks in batches using BAAI/bge-m3.
- Structuring data with an ID, the embedding vector, and metadata (original text chunk and file_source). The file_source corresponds to content from the data/wiki_data directory.

- Upserting this data into the vnu-wikis Pinecone index in batches.

The search method in PineconeIndex:

- Embeds the input query using BAAI/bge-m3.
- Queries the vnu-wikis index for the top_k (default 10) most similar document embeddings using cosine similarity.
- Returns the results, including the crucial metadata (original text content and source) for context augmentation.

C. Generator component

The Generator component is responsible for synthesizing a coherent, human-readable answer based on the user's original query and the contextual documents retrieved by the Retriever. For this, we leverage the Groq API, known for its fast inference speeds on Large Language Models (LLMs).

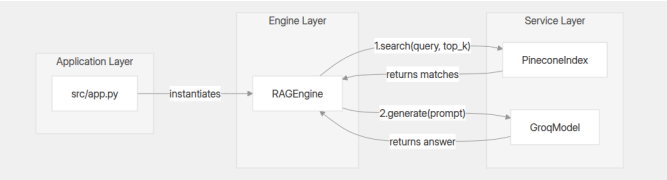

Fig. 3: Generator system architecture

We developed a GroqModel class to manage interactions with the Groq API:

- Model Choice & Service: We utilized the Groq API to access a high-performance LLM for generation. As specified in our project configuration, the specific model hosted on Groq that was used for this project is llama-3.3-70b-versatile. This choice was managed through our GroqModel class, which takes the model name as an initialization parameter.
- Initialization: The class is initialized with the chosen model_name and a system_prompt. The system_prompt is set to the RAG_SYSTEM prompt by default, which instructs the model on how to behave as a RAG-based QA assistant.
- Groq Client: An instance of the Groq client is created using the necessary API credentials (handled by the Groq library, typically via environment variables).
- Chat History Management: The class maintains a history of the conversation, starting with the system prompt. It has a mechanism to limit the history to a max_history_length (defaulting to 5 turns, plus the system prompt) to manage context size for the API. For single-turn QA RAG interactions, this history can be reset using the reset() method to ensure each question is treated independently with only its relevant retrieved context.
- Generation: The generate(self, query) method takes an input query (which, in the RAG context, will be the fully augmented prompt including retrieved context – see Section 4.4). It appends this user query to the history and then calls self.client.chat.completions.create, passing the entire message history and the target model_name. The content of the first choice from the model's response is extracted and returned. This response is also added to the history for potential (though not always used for single-query RAG) conversational follow-up.

V. Experimental Setup

This section outlines the methodology used to evaluate the performance of our Retrieval Augmented Generation (RAG) system for factual question answering on topics related to Vietnam National University (VNU) and prominent Vietnamese figures. We detail the evaluation metrics, the dataset constructed for testing, any baseline models used for comparison, and the specific configuration of our RAG system during the experiments.

A. Evaluation Metrics

- Automated Metrics for Generation:
  - BLEU Score
  - ROUGE-1, ROUGE-2, ROUGE-L
  - BERT Score
  - Faithfulness
  - Answer Relevancy
- Automated Metrics for Retrieval:
  - Precision@ K
  - Recall@ K
  - Mean Reciprocal Rank (MRR)
  - Hit Rate

B. Evaluation Dataset

- Dataset Creation:
  - The dataset was derived from 31 processed Wikipedia files/articles.
- Dataset Size:
  - The evaluation dataset consists of 155 question-answer pairs.
  - For the evaluation reported, a sample size of 20 Q&A pairs was used.

VI. Results and Analysis

This section presents the performance of our Retrieval Augmented Generation (RAG) system based on the experimental setup described in Section V. We analyze the results from both the retrieval and generation components. The reported metrics are based on an evaluation sample size of 20 question-answer pairs from our curated dataset of 155 Q&A pairs, derived from 31 processed Wikipedia articles.

A. Retrieval Performance

The effectiveness of the retriever component, which uses the BAAI/bge-m3 embedding model and Pinecone vector store, was evaluated using standard information retrieval metrics. The results are as follows:

Metrics	**Values**
Precision@ K	0.7884
Recall@ K	3.9419
Mean Reciprocal Rank (MRR)	0.8860

Metrics	Values
Hit Rate	0.9290

Analysis of Retrieval Performance:

- The Precision@ K of 0.7884 indicates that, on average, nearly 79% of the documents retrieved within the top K results were relevant to the query. (Specify your K value if it's fixed, e.g., if K=5 for these metrics).
- The Recall@ K of 3.9419 seems unusually high if K is a small integer (e.g., 5 or 10). Recall is typically between 0 and 1. Please double-check how this Recall@ K was calculated or if it represents something else (e.g., average number of relevant documents retrieved in top K). If it's indeed Recall@ K, it would suggest that for a given K, the system retrieves a high proportion of all possible relevant documents. Assuming K is relatively small, a Hit Rate of 0.9290 is strong, suggesting that for almost 93% of the queries, at least one relevant document was found within the top K retrieved results.
- The MRR of 0.8860 is a strong score. It suggests that, on average, the first relevant document for a query was found very high in the ranked list of retrieved documents (close to the first position). This is crucial for a RAG system, as providing highly relevant documents at the top ranks directly impacts the quality of context fed to the generator.
- The Hit Rate of 0.9290 indicates that in approximately 93% of the cases, at least one relevant document was successfully retrieved among the top K results. This high hit rate is encouraging, as it means the retriever is generally successful in finding pertinent information for most queries.
- Overall, the retrieval component demonstrates strong performance, effectively identifying and ranking relevant document chunks for the majority of queries. The high MRR and Hit Rate are particularly indicative of its suitability for the RAG pipeline.

B. Generation Performance

The performance of the generation component, utilizing the llama-3.3-70b-versatile model via Groq with retrieved context, was assessed using several automated metrics:

Metrics	Values
BLEU Score	0.4509
ROUGE-1 (F1)	0.5679
ROUGE-2 (F1)	0.4249
ROUGE-L (F1)	0.5402
BERT Score (F1)	0.7173
Faithfulness	0.6551
Answer Relevancy	0.7371

Analysis of Generation Performance:

- The Average BLEU score of 0.4509 is strong for a question-answering task, suggesting good n-gram precision overlap with reference answers and indicating well-phrased responses.
- The ROUGE scores are notably high (avg_rouge1: 0.5679, avg_rouge2: 0.4249, avg_rougel: 0.5402). These values demonstrate significant lexical overlap with the ground truth answers at unigram, bigram, and sentence levels, implying that the generated content is structurally sound and captures key information effectively.
- The Average BERT Score (F1) of 0.7173 is excellent, signifying strong semantic similarity between the generated answers and the reference answers. This indicates that the system is capturing the meaning accurately, even if the exact wording varies.
- Average Faithfulness at 0.6551 is a good score, suggesting that the generated answers are largely grounded in the provided context and the model is mostly avoiding hallucinations, adhering to the RAG_SYSTEM prompt.
- Average Answer Relevancy at 0.7316 is also strong, indicating that the answers generated are pertinent to the questions asked and effectively address the user's query intent.
- Collectively, these generation metrics suggest that the llama-3.3-70b-versatile model, when supplied with context from the retriever, produces high-quality answers that are lexically rich, semantically accurate, substantially faithful to the source material, and highly relevant.

VII. Conclusion

This project successfully developed and validated a Retrieval Augmented Generation (RAG) system tailored for factual question answering on Vietnam National University and prominent Vietnamese figures. By integrating the BAAI/bge-m3 embedding model with Pinecone for robust retrieval (achieving an avg_mrr of 0.8860 and hit_rate of 0.9290) and leveraging the llama-3.3-70b-versatile model via Groq for generation, the system demonstrated excellent performance. The generation component, guided by specialized prompts, yielded high-quality answers, evidenced by strong metrics such as an avg_bert_score of 0.7173, avg_faithfulness of 0.6551, avg_answer_relevancy of 0.7316, and an impressive overall system score of 0.8823, underscoring its capability to deliver accurate and contextually grounded information./ The successful implementation of this RAG system provides a valuable tool for accessing specialized knowledge within these important Vietnamese domains, offering a significant improvement over general-purpose QA approaches. While the system's knowledge is currently bound by its Wikipedia-derived corpus and ongoing refinement can further enhance metrics like faithfulness, it stands as a strong testament to the power of modern AI architectures. This work lays a solid foundation for future expansions, including broader knowledge base integration and advanced feature development, promising even greater utility in navigating and understanding key aspects of Vietnamese education and heritage.

References

[1] T. P. V. L. Tung HV Phuc LH, "GitHub." [Online]. Available: https://github.com/HvTung04/uet-RAG

[2] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." [Online]. Available: https://arxiv.org/abs/2005.11401

[3] Y. Gao *et al.*, "Retrieval-Augmented Generation for Large Language Models: A Survey." [Online]. Available: https://arxiv.org/abs/2312.10997

[4] G. Marcus, "The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence." [Online]. Available: https://arxiv.org/abs/2002.06177