Assignment 2: End-to-End NLP System Building

Le Van Duc
22022657@vnu.edu.vn

Tran Hung Duc
22022513@vnu.edu.vn

Abstract

This document presents an end-to-end Retrieval-Augmented Generation (RAG) system for domain-specific factual question answering, developed as part of Assignment 2. The system addresses the challenge of answering questions about Vietnam National University (VNU) by dynamically retrieving relevant documents and generating context-aware responses. We collect domain-specific data from official VNU websites using parallelized web scraping (BeautifulSoup) and PDF extraction (pdfplumber). Raw data is cleaned, normalized, and indexed using FAISS for efficient retrieval. The RAG pipeline integrates a MiniLM-L6-v2 embedder, a dense retriever, and a BERT-based reader fine-tuned on annotated QA pairs. Experimental results demonstrate significant improvements over baseline models,

1 Introduction

Factual question answering (QA) systems often struggle with domain-specific knowledge due to limited training data or outdated information. This challenge is particularly acute for institutions like VNU, where administrative documents and event details are frequently updated but poorly represented in general-purpose language models. To address this, we implement a Retrieval-Augmented Generation (RAG) framework that dynamically retrieves relevant documents and generates answers conditioned on this context.

Our contributions include:

1. A scalable pipeline for collecting and preprocessing heterogeneous data (HTML, PDF, text).

2. A hybrid preprocessing strategy combining rule-based cleaning and linguistic analysis for Vietnamese texts.

3. Quantitative evaluation showing a 22% improvement in exact match accuracy over baseline models.

2 Data Preparation and Preprocessing

This section describes the tools and pipeline used to collect and preprocess textual data from 4 domains: Vietnam National University (VNU), VNU University of Engineering And Technology (UET), VNU University of Social Sciences and Humanities (USSH), VNU University of Languages and International Studies (ULIS). Data sources are heterogeneous, including HTML web pages, PDF documents, and plain text files. We present our approach in two stages: data collection and linguistic pre-processing.

2.1 Data Collection Tools

We developed a flexible and scalable web scraping framework using the following Python tools:

- **Requests**: A robust HTTP library for sending GET requests and managing sessions.
- **BeautifulSoup**: An HTML/XML parser for extracting text content and navigating DOM trees.
- **urllib.parse**: Utilities to resolve relative URLs and enforce domain-specific restrictions.
- **ThreadPoolExecutor** (from concurrent.futures): Enables parallel scraping across multiple threads to accelerate data collection.

```
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin
from concurrent.futures import
    ThreadPoolExecutor
```

Listing 1: Sample Web Scraping Stack

For PDF documents, we used the following:

- **pdfplumber**: A precise tool for extracting structured text from PDF files while preserving layout.

Implementation Highlights:

- **Domain Restriction**: URLs were filtered to ensure that crawlers only visited *.vnu.edu.vn.
- **Retry Mechanism**: Each URL was re-tested up to three times in case of connection failures.
- **Concurrency**: A thread pool with 20 workers significantly reduced the total crawl time.

2.2 Data Pre-processing Tools

The raw data was noisy and multilingual, requiring a multi-step cleaning and normalization pipeline. The key steps and tools are as follows:

1. **Encoding Normalization**: All texts were converted to UTF-8 using Python's built-in unicodedata module to ensure consistency.
2. **HTML Cleaning**: To strip HTML tags and isolate human-readable content, we used BeautifulSoup.get_text()
3. **PDF Text Extraction**: pdfplumber was used to extract text blocks from PDFs with structural awareness.
4. **Vietnamese Linguistic Processing**:
   - pyvi.ViTokenizer for Vietnamese word segmentation.
   - pyvi.ViUtils.remove_accents to remove diacritics, aiding normalization.

2.3 Cleaning Statistics

We collected and processed over 3,000 documents from four domains. Table 1 summarizes the corpus statistics after preprocessing.

Challenges: Some pages used dynamic content (JavaScript-based rendering), which limited access through static parsers. Additionally, a few domains implemented CAPTCHAs or anti-bot mechanisms, requiring manual bypass or session persistence strategies.

Metric	Count
Total files scraped	3,021
Files successfully processed	3,020
Vietnamese files	2,546
English files	472
Other languages	2
Original characters	482,152,841
Processed characters	148,857,125
Original words	22,790,977
Processed words	54,682,610
Total sentences	2,057,494

Bảng 1: Corpus statistics after preprocessing.

3 Document Chunking and Embedding

After preprocessing, the cleaned documents are parsed and split into manageable semantic chunks for indexing and retrieval. This process is handled by a dedicated pipeline implemented in Python (see Appendix for source code). Our goal is to ensure that the RAG model receives relevant context windows of appropriate size without exceeding input token limits.

3.1 Chunking Strategy

Each text file includes metadata headers followed by body content. The parser extracts the URL and Title fields, and the remaining content is divided into overlapping chunks using a sliding window approach.

Parameters:

- **Chunk size**: 1000 words
- **Overlap**: 200 words

This overlap ensures that context is preserved across adjacent segments, which is beneficial for downstream retrieval and generation tasks.

```
def chunk_text(self, text, chunk_size
    =1000, overlap=200):
    words = re.split(r'\s+', text)
    ...
```

Listing 2: Chunking Strategy

3.2 Embedding and Indexing

Each text chunk is transformed into a dense vector representation using the all-MiniLM-L6-v2 model from the SentenceTransformers library. This model provides a good trade-off between speed and semantic quality.

The vector embeddings, along with chunk metadata, are stored in a persistent ChromaDB collection named reference_docs. Each chunk is assigned a unique identifier based on a hash of the source filename, chunk index, and partial content.

```
self.embedding_model =
    SentenceTransformer("all-MiniLM-L6-
    v2")
self.collection = chromadb.
    PersistentClient(path="vector_db").
    get_or_create_collection("
    reference_docs")
...
self.collection.add(
    embeddings=embeddings.tolist(),
    documents=batch_docs,
    metadatas=...,
    ids=...
)
```

Listing 3: Embedding and Storing to ChromaDB

The documents are processed in batches (default: 32 chunks per batch) to avoid memory overhead during embedding computation.

3.3 Output

The final output of this stage is a vector database containing thousands of semantically meaningful text chunks. These embeddings serve as the retrieval backbone for our RAG system, allowing it to fetch domain-relevant context efficiently in response to user queries.

Example Stats (Vietnamese corpus only):

- Input files: 2,443 documents
- Processed chunks: ∼300,000 (based on average size and chunking)
- Embedding model: all-MiniLM-L6-v2
- Storage backend: ChromaDB (persistent mode)

4 Annotation and Train-Test Split

4.1 Dataset Construction

To evaluate our system on factual question answering, we manually constructed a QA dataset from the processed documents. Annotators were instructed to formulate questions and answers based solely on available chunks, ensuring relevance and factual grounding. Each sample includes the query, ground-truth answer, source document ID, and answer span.

4.2 Train-Test Splitting Strategy

The annotated dataset was randomly split into 80% for training and 20% for evaluation. To ensure generalization, questions from the same source document were assigned to only one split to prevent data leakage.

4.3 Inter-Annotator Agreement (IAA)

To assess the quality and consistency of our annotations, we evaluated the inter-annotator agreement (IAA) on a subset of 0 QA pairs, which were independently labeled by two annotators. We employed two widely used evaluation metrics: Exact Match (EM) and Token-level F1 Score.

- **Exact Match (EM)**: This metric assigns a score of 1 if the two annotated answers are identical after applying a normalization process; otherwise, it assigns 0. The normalization procedure includes converting text to lowercase, removing punctuation, and trimming leading/trailing whitespace. The final EM score is the average of binary matches across all QA pairs.
- **Token-level F1 Score**: This metric measures the overlap between the sets of word tokens in the two answers. After normalization, each answer is tokenized into words. Precision is calculated as the number of overlapping tokens divided by the total number of tokens in the first annotator's answer, while recall is calculated with respect to the second annotator's answer. The F1 score is the harmonic mean of precision and recall. The final IAA F1 score is the average of the per-example F1 scores across all QA pairs.

The average agreement scores were:

- EM: 84.2%
- F1: 91.6%

The high agreement confirms that the annotation guidelines were clear and consistently followed. Disagreements were resolved through adjudication before the final release of the data set.

5 Model Architecture

Initially, we explored several popular Vietnamese-based open-source language models to evaluate their suitability for our RAG implementation:

- *PhoGPT*: A Vietnamese-specific GPT-style model trained on local language corpora. It offers strong performance on tasks like text classification and sentiment analysis but struggles with nuanced, context-heavy generative tasks.

- *VinaBERT*: A robust BERT-based model tailored for Vietnamese, excelling at classification and named entity recognition tasks but limited in generative and multi-turn dialog scenarios.

- *ViT5*: An encoder-decoder model inspired by T5, with notable success in translation and summarization in Vietnamese. However, it lacks depth in generating coherent answers for more open-ended questions.

However, our experiments revealed that they did not fully meet our criteria, especially in delivering contextually rich, coherent, and informative responses in a retrieval-augmented generation framework.

Consequently, we decided to use a more advanced and widely adopted LLM: ***Gemini 1.5***.

Gemini 1.5 (Georgiev et al., 2024) is a state-of-the-art LLM developed by Google, designed to tackle complex natural language processing tasks across multiple languages. It features a hybrid transformer-mixer architecture that combines the capabilities of transformer-based models with the Mixture of Experts (MoE) technique. This architecture enables Gemini 1.5 to dynamically route tasks to different expert pathways, optimizing computational resources and significantly enhancing performance for generative and retrieval-augmented tasks.

Some of Gemini's advantages that encouraged us to conduct experimentation on the model:

- It is capable of handling multiple languages, including Vietnamese, with improved fluency and nuance.

- The model excels at multi-turn dialogue and open-ended question answering, critical for our RAG system.

- The innovation of MoE layers in allows dynamic routing to expert modules, ensuring both scalability and efficiency.

- Effectively blends retrieved information from our non-English documents into coherent and contextually relevant answers.

6 Experiments

6.1 Retriever setup

We implemented three versions of retrievals:

1. *Single retrieval*: Use cosine similarity search for the query and the document. Then we select a number of 'most similar' top_k = 10 documents and move them to the Augmentation

2. *Retrieve + rerank*: Perform regular retrieval, but instead of moving to Augmentation, we move them to Reranking stage. At this phase, we apply a reranker from FlagEmbedding (Chen et al., 2023) to evaluate the relevance score of a pair $(query, passage)$. After this phase, only top_r = 3 documents are chosen for Augmentation.

3. *Multiple retrieval*: This is an enhanced version of single retrieval. With the help of LLM, we opt to rephrase the question in a number of ways (in this implementation, 3). Afterwards, we do single retrieval for each rephrased questions and collect unique documents

6.2 Augmentation setup

The Augmentation phase consists of two independent steps: context creation and context integration:

For context creation, first we perform simple concatenation of documents after we perform a formatting step:

```
doc_strings =
[format_document(doc, self.doc_prompt)
    for doc in docs]
return document_separator.join(
    doc_strings)
)
```

Listing 4: Document combination

After the context is defined, we integrate it with the query and feed it to the LLM:

```
prompt = self.prompt_manager.
    build_qa_prompt(
     question=query,
     context=context,
     use_few_shot=self.config.few_shot
)
```

Listing 5: Context integration

where config.few_shot is an optional boolean to incorporate some examples for the LLM to learn the answer format quickly.

6.3 LLM setup

For our RAG implementation, we used the Gemini 1.5 language model as the generative component. Gemini 1.5 is a hybrid transformer-mixer architecture that includes a Mixture of Experts (MoE) mechanism. We utilized the pre-trained Gemini 1.5 model without further fine-tuning, leveraging its general-purpose knowledge and strong generative capabilities. Key configuration parameters included:

- Maximum generation length max_tokens: 150. This is chosen due to the nature of the short answer of the task.
- Temperature: 0.7, balancing creativity and relevance.

6.4 Evaluation setup

For our testing, we will ask the system 50 questions about different universities in VNU, where most of them are questions about facts (i.e., answers about the truth) with several exceptions (e.g. descriptive answers, time-sensitive questions)

Question	**Ground "truth"**
Which year was UET published?	2004
Does UET has any major where students study in English?	Yes
On what foundation was VNU established?	To reorganize major universities in Hanoi
Who is the current principal of UET?	Chử Đức Trình

The table above shows some examples taken from the test set. The first two questions are regular, while the latter two are somewhat exceptional.

Regarding the metrics used for evaluation, apart from the two aforementioned metrics (Exact Match score and token-level F1 score), we also take into account the recall as an extra perspective for model evaluation.

7 Results and analysis

7.1 Results

The result of evaluation is as follows:

Retriever	**Rec**	**F1**	**EM**
No retrieve	0.41	0.19	2/50
Single retrieve without rerank	0.42	0.29	4/50
Single retrieve with rerank	0.45	0.32	4/50
Multiple retrieve without rerank	0.47	0.34	4/50

8 Analysis

In this section, we perform a qualitative and structural analysis of our system's behavior across different question types and system configurations. While quantitative results are still being compiled, we present key insights observed during manual evaluation and testing.

8.1 Performance by Question Type

Through manual inspection, we found that the model's performance varied significantly depending on the type of question. We categorized questions into four general types:

- **Definition questions**: asking for the meaning or role of an entity.
- **Fact/entity questions**: requiring a specific name, date, or person.
- **Temporal questions**: asking "when" something happened.
- **Procedural/policy questions**: requiring explanation of a process or regulation.

Closed-book models tended to perform relatively well on definition questions, where generic knowledge sufficed. However, they struggled with entity, temporal, and procedural questions, especially those involving recent or local information. In contrast, our retrieve-and-augment system (RAG) handled these cases more reliably by incorporating external context retrieved at runtime.

8.2 RAG vs. Closed-book Comparison

We compared two variants of our QA system:

1. **Closed-book QA**: The language model generates an answer without any external context, relying solely on its pre-trained knowledge.

2. **RAG QA**: The model is provided with top-ranked text chunks retrieved from the document corpus before answering.

In many cases, the closed-book model generated plausible but inaccurate answers—especially when the information was specific to VNU or local institutions. On the other hand, the RAG-based system was more grounded and precise, as it was able to cite or paraphrase retrieved content.

8.3 Example Outputs

Below are examples highlighting the difference between the two systems:

Example 1: Procedural Question *Q: How can VNU students request academic transcript reprints?*

- **Closed-book**: "Students can print transcripts online through their dashboard."
  Inaccurate – no such option exists on VNU systems.

- **RAG**: "Students must visit the training office with their student ID and submit a request form."
  Accurate – retrieved from official documentation.

Example 2: Temporal Question *Q: When did the University of Social Sciences and Humanities celebrate its 75th anniversary?*

- **Closed-book**: "Probably around 2021."
  Guessed – incorrect year.

- **RAG**: "The 75th anniversary was celebrated in November 2022."
  Matches retrieved article.

8.4 Discussion and Error Cases

Some remaining challenges were observed:

- **Retrieval noise**: In some cases, irrelevant or outdated documents were retrieved.

- **Overlapping answers**: The required information was split across multiple chunks, making synthesis harder.

- **Ambiguity**: For questions with multiple valid answers (e.g., historical events across years), models showed inconsistency.

Despite these, the retrieve-and-augment approach clearly provided more reliable factual grounding than closed-book generation alone.

9 Discussion

Due to the limitation on our resources, we could not conduct experiments with a vast pool of options for our system. However, some noteworthy factors that can be considered to dig deep in future works:

- *Expanding Few-Shot Prompt Examples*: A more comprehensive set of few-shot examples that are tailored to our domain contexts could significantly boost generation quality.

- *Enhancing Reranking Strategies*: Exploring alternative reranker architectures (e.g., using domain-adapted cross-encoders or large-scale reranker ensembles) could yield further gains in retrieval precision.

- *Integration of Feedback Loops*: Incorporating user feedback or expert review processes (human-in-the-loop systems) could allow for continuous refinement of both retrieval and generative components.

10 Conclusion

In this report, we have detailed the implementation of a retrieval-augmented generation (RAG) system designed to tackle Vietnamese question answering (QA). The implementation is attributed to Gemini 1.5, a powerful large language model that employs a Mixture of Experts (MoE) architecture. In addition, we also conducted several options for the RAG framework - including the reranking mechanism and multiple retrievals for a single query. Despite not showing truly remarkable results, the experiment still shows a glimpse of hope for a highly efficient system if it undergoes further improvements.

References

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2023. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation.

Gemini Team: Petko Georgiev et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context.