AI and NLP Models for Q&A in Georgian

Sergo Tsiramua
University of Georgia
Tbilisi, Georgia
e-mail:s.tsiramua@ug.edu.ge

Hamlet Meladze
Muskhelishvili Institute of Computational Mathematics of Georgian Technical University,
Tbilisi, Georgia
e-mail: h_meladze@hotmail.com

Tinatin Davitashvili
Ivane Javakhishvili Tbilisi State University,
Tbilisi Georgia
e-mail: tinatin.davitashvili@tsu.ge

Davit Bitmalkishev
University of Georgia
Tbilisi, Georgia
e-mail: davit.bitmalkishev@ug.edu.ge

Tatia Elbakidze
University of Georgia
Tbilisi, Georgia
e-mail: tatia.elbakidze@ug.edu.ge

***Abstract*—The paper discusses the application of NLP technologies for the Georgian language in a question-answering format. In the developed web application, based on pre-uploaded PDF files, users can ask questions in Georgian related to document content via an NLP system and receive relevant answers in text and audio formats. The system analyzes uploaded documents, breaks down PDFs into vectors, processes them using the cosine similarity algorithm, and generates relevant answers. This required integrating several components: developing a Flask Web application, enabling PDF processing, using embeddings with a database, and creating a question-answering module. To ensure efficient information processing, the LangChain framework, OpenAI's language models, and ChromaDB were used.**

The purpose of developing this NLU (Natural Language Understanding) technology was to simplify information retrieval from large document volumes and generate fast, accurate, context-based responses. The use of NLU technology ensured the system's speed, simplicity, flexibility, efficiency, and adaptability to various needs and technological requirements. This system can be applied in fields where users deal with large textual data, including education, business, and research.

***Keywords*—Artificial intelligence, language processing technology, language models, cosine similarity algorithm.**

I. INTRODUCTION

In today's information-driven environment, with the growing volume of digital documents, the need for intelligent systems that process, analyze, and retrieve relevant information from these documents is becoming increasingly important. Traditional document search methods are often labor-intensive and time-consuming, creating a demand for more intelligent and interactive solutions.

The rapid digitization of information in the modern world generates massive amounts of data stored in various formats, including PDF files. The documents of this format are widely used across all areas of human activity where in-depth analysis of large volumes of text is required. However, finding the desired information within these documents is often labor-intensive and time-consuming. Traditional search functions in PDF programs are limited, as they cannot understand context, meaning, or content through simple keyword searches.

With advancements in artificial intelligence (AI) and natural language processing (NLP) technologies, there is a growing demand for applications capable of complex text analysis. Moreover, in the era of remote work and online collaboration, such applications become essential tools for improving productivity and access to information. The relevance of this work lies in its potential to transform how users interact with digital documents - making it faster and more efficient to retrieve and process the necessary information.

This paper discusses the development of a technology and application that allows uploading, processing, and answering user questions based on the content of Georgian-language documents in PDF format. The combination of a Retrieval-Augmented Generation (RAG) system and the GPT-4o model opens up new possibilities for more intelligent and faster interaction with documents. The application enables users to interact with documents dynamically: they can ask questions, and the system provides accurate answers in real time - greatly enhancing the efficiency of working with large-scale documents.

Based on the project objectives, the following tasks were carried out:

1. PDF File Upload and Vectorization Functionality Developed: Users can upload PDF documents into the application, which converts them into vectors using the Text-embedding-3-large model.
2. Integration of the RAG System Using the GPT-4o Model: The RAG system was integrated, utilizing the GPT-4o model to understand information in context and provide accurate answers to questions.
3. Use of the Text-embedding-3-large Model: Integration of the Text-embedding-3-large model enabled the system to store the corresponding vector data of the text, allowing the system to retrieve relevant context in response to user queries.

4. Development of a Flask-based Web Application: A web application built with Flask was created, ensuring system speed, flexibility, and scalability.
5. Information Retrieval and Question-Answering Mechanism Developed: An interface was developed within the application that allows users to ask questions based on the document content and receive accurate answers quickly.

Given the relevance of the project and the growing demand for similar services, the product we have developed can be applied across various fields, including business, the legal sector, healthcare, education, online learning, and more.

II. NLP Technologies for Q&A

Natural Language Processing (NLP) is a combination of artificial intelligence, computer science, and linguistics that processes human language in a way that is understandable to computers. As AI-powered devices and services become increasingly integrated into our daily lives, the influence of NLP continues to grow, enabling seamless interaction between humans and technology [1].

NLP involves various techniques for analyzing human language. Some of the most common techniques encountered in this field include:

1. Sentiment Analysis: An NLP technique that analyzes text to determine sentiment, such as “positive”, “negative”, or “neutral”. Sentiment analysis is widely used in business to better understand customer feedback.
2. Summarization: An NLP technique that summarizes longer texts to make them more accessible for time-constrained readers. These typically include reports and articles.
3. Keyword Extraction: An NLP technique that analyzes text to identify the most important keywords or phrases. Keyword extraction is used for search engine optimization (SEO), social media monitoring, and business research.
4. Tokenization: The process of breaking text into "tokens" - words, symbols, or subwords - that a program can analyze. Tokenization is widely used in NLP tasks such as word modeling, building lexicons, and identifying frequently occurring words [2, 3].

While NLP can be applied in a wide range of applications, the technology is still far from perfect. In fact, many NLP tools struggle to accurately interpret sarcasm, emotion, slang, context, errors, and other forms of ambiguous expression. This means NLP remains somewhat limited.

Retrieval-Augmented Generation (RAG) System, GPT 4o and Text-Embedding-3-Large Model. With the development of Large Language Models (LLMs), the NLP field has undergone a transformative shift, enabling significant improvements in language understanding, generation, and contextual applications. However, managing and searching high-dimensional vector embeddings remains a major challenge in building scalable systems based on LLMs. ChromaDB, a vector database, and LangChain, a framework for building complex LLM-based applications, represent key innovations in this domain. In the following sections, we will explore the integration of ChromaDB and LangChain, aimed at creating efficient and scalable architectures for real-world applications such as document retrieval, conversational agents, and personalized recommendation systems. This combination reduces complexity, improves response speed for question answering, and optimizes the scalable use of LLM applications [4].

Nonetheless, scaling these models and using them efficiently in real-world operations presents significant challenges. The embeddings generated by these models require efficient storage and search mechanisms, especially for tasks such as semantic search or personalized content delivery. Traditional databases, optimized for structured data, struggle with processing high-dimensional vector data, which calls for specialized solutions for embedding management, indexing, and retrieval.

The use of LLMs in modern NLP applications has led to the generation of high-dimensional embeddings that encode linguistic information. These embeddings are typically used in tasks such as document retrieval, text classification, and named entity recognition. They allow for a more nuanced understanding of semantic similarity than traditional keyword-based approaches.

However, as the size of these models and their associated datasets grows, managing and retrieving embeddings becomes increasingly complex. Techniques like exact nearest neighbor search become impractical for real-time use when dealing with high-dimensional data. To mitigate these challenges, indexing and approximate search methods have been proposed, including Locality-Sensitive Hashing (LSH) and k-d trees [5].

Vector databases are optimized for storing and retrieving high-dimensional embeddings. Unlike traditional databases, which are designed for structured data, vector databases are built to handle dense vectors generated by machine learning models. Tools like FAISS (Facebook AI Similarity Search), Pinecone, and Milvus have been developed to meet this need.

Chroma DB stands out by offering specialized support for real-time applications, focusing on scalability, real-time embedding storage, and fast similarity search. Its optimized indexing structures assist dynamic LLM applications, where embeddings are frequently generated and queried in real time.

LangChain simplifies the development of LLM-based applications by providing an abstraction layer for creating dynamic pipelines. It enables developers to build modular chains that connect LLMs with external tools such as databases, APIs, and memory components. LangChain facilitates dynamic querying, memory retention, and interaction with embeddings, offering flexibility for solving complex NLP tasks [6].

Unlike other frameworks, such as Hugging Face Transformers, which primarily focus on model inference, LangChain integrates LLMs into broader workflows that require continuous interaction with data sources, memory management, and query optimization. This makes it ideal for building applications where context retention and real-time interaction are critical, such as virtual assistants, chatbots, and recommendation engines.

Integrating vector databases into NLP pipelines presents challenges. Managing large volumes of high-dimensional embeddings is complex, and retrieving embeddings effectively in real time requires optimized indexing strategies. The use of Approximate Nearest Neighbor (ANN) search techniques improves retrieval speed, but often results in trade-offs with accuracy, requiring careful balancing between precision and performance.

Additionally, embedding management in LLM applications must account for dynamic updates, as embeddings may evolve over time based on user interactions or incoming data. This requires databases capable of handling frequent writes and updates while maintaining low-latency querying performance.

Euclidean distance measures absolute difference in coordinates and is sensitive to magnitude/scale [7].

Cosine similarity measures directional alignment (the angle); it is invariant to uniform rescaling of the vectors — useful when only orientation matters (e.g., text embeddings, high-dimensional sparse data) [8].

III. Realization of NLP Technology

A web application based on NLP technology is capable of learning from uploaded PDF files and answering user questions based on the content of the document [9]. The project consists of several components:

1. The main Flask application file, where the functionality for text processing and question answering is defined using the Flask framework. Additionally, the application's routing is configured here.
2. An HTML file, which defines the structure of the web application and, using JavaScript, implements the functionality for uploading PDF files and submitting questions.
3. A CSS file, which provides the visual styling of the web application.
4. The Upload folder, where PDF files uploaded by users are stored.
5. The Chroma folder, where information obtained from text processing is stored.

Discussion of the **Flask application file.** The core logic of the project handles text processing and answering user questions. For this purpose, we use the Text-Embedding-3-large and GPT-4o models for natural language processing, along with the Chroma DB vector database.

After uploading the PDF file, the text contained within it is processed. First, the text is divided into sections – chunks - using a text splitter. Although it's possible to split the text into individual sentences, this type of segmentation complicates the process of retrieving accurate answers to user queries. In this case, the text is divided into sections of 800 characters (see Fig. 1). However, such segmentation may cause a section to end in the middle of a sentence, potentially leading to a loss of context. To avoid this, overlaps are used - specifically, an 80-character overlap. This means that after one chunk ends, the next chunk starts 80 characters before the end of the previous one. Finally, the resulting chunks are stored in Chroma DB.

```
Text_splitter = RecursiveCharacterTextSplitter (
    chunk_size=800, chunk_overlap=80, length_function=len,
is_separator_regex=False
)
```

Fig. 1. Text Segmentation

After the text is divided into chunks, the processing phase begins. Using a natural language processing model, specifically Text-Embedding-3-Large, the text contained in the chunks is converted into vectors. These vectors are then stored in the vector database, Chroma DB, alongside their corresponding chunks.

The same process applies to the user's question. The question is processed using the Text-Embedding-3-Large model and converted into vector data.

After completing the above processes, the question answering phase begins.

The cosine similarity is calculated between the vector of the user's question and the vectors stored in Chroma DB. This process works as follows:

Cosine Similarity measures the angle between two vectors in a multidimensional space and evaluates their semantic similarity. It is defined as:

$$cosine_{similarity(A,B)} = \frac{A \cdot B}{\| A \| \cdot \| B \|}, \in [-1,1]$$

where:

- A·B is the dot product of the two vectors,
- ‖A‖ and ‖B‖ are the magnitudes (lengths) of vectors A and B.

The smaller the angle between the vectors, the closer the corresponding text vector is to the answer. Based on the ascending order of cosine similarity values, the system retrieves the five vectors with the smallest angles from the database. The texts corresponding to these vectors are then processed by the AI model, GPT 4o. GPT 4o summarizes the provided texts and returns a response according to the defined task (Prompt) (see Fig. 2).

```
PROMT_TEMPLATE = " " "
Answer the question based only on the following context:
{context}
---
Answer the question based on the above context: {question}
" " "
```

Fig.2. Prompt

Three routes are defined in the application:

1. Main route, which represents the main page of the application, where users can upload PDFs and ask questions. It uses the HTML file.
2. Upload route (/upload). This route accepts only POST requests and is used for uploading files. It checks whether a file has been uploaded. After processing the PDF, it returns a message confirming the file was successfully uploaded and divided into a specific number of chunks.
3. Query route (/query). This route accepts the user's question in JSON format, processes it, and returns an answer from GPT4o. The received answer is then sent back to the user.

Methods. For creating a web application, it is important to use the necessary software packages, algorithms, and technologies. The technologies we select influence the quality of the application, its speed, and stable operation.

Software packages and libraries used in the project:

1. **Flask**: Used to create the web server and handle HTTP requests and responses.
2. **Request**: Receives incoming HTTP requests, such as file uploads and query parameters.
3. **Jsonify**: Converts Python objects (e.g., dictionaries) into JSON format responses.

4. **Render_template**: Used to render HTML templates so that they can be displayed in the browser.
5. **OS**: Manages file paths and directories.
6. **Secure_filename**: Ensures safe saving of uploaded file names to avoid invalid characters or vulnerabilities.
7. **PyPDFLoader**: Used to extract text from PDF files.
8. **RecursiveCharacterTextSplitter**: Used to split large documents into manageable text chunks.
9. **Document:** Represents a text chunk with metadata.
10. **Chroma**: A vector database used for storing embeddings and searching.
11. **Text-embedding-3-large**: Converts text into numerical vectors used for similarity search.
12. **GPT4o**: A language model used to generate answers.
13. **ChatPromptTemplate**: Used to structure the prompt format for the language model.

The project was developed using the following technologies: Python 3.10, the primary programming language used for implementing all modules, and HTML, CSS and JavaScript, used to create the user interface.

Algorithms and approaches used:

1. **Natural Language Processing (NLP):** Used for processing the text within PDF files and the questions posed by users. With the help of LangChain and OpenAI models, it is possible to generate answers to questions.
2. **Text Embedding:** Text is converted into embeddings (vectors), which allows for comparison of texts and finding similar parts.
3. **Chunking Algorithm:** Documents are split into chunks using RecursiveCharacterTextSplitter (up to 800 characters).
4. **Similarity Search:** The most relevant texts are found in the Chroma database based on embeddings.
5. **Prompt Engineering:** Structured and context-oriented templates are used for efficient answer generation.

Supporting technologies:

1. **HTTP REST API:** Provided through Flask to enable data exchange.
2. **JavaScript Fetch API**: Enables communication between forms in the HTML file and the server.
3. **CSS:** Used for formatting visual elements.
4. **JSON**: Used as the data format for user questions and server responses.

In summary**,** a web application was created through which users can upload PDF files and receive answers based on the content of those files. The main page of the website consists of sections for file upload and question submission.

Application Improvement Opportunities. Although the existing web application functions smoothly, several improvements could further enhance its performance and user experience:

1. **Support for Multiple File Types**: Currently, the system only supports PDF files, but it could be extended to recognize other document types (such as Word or plain text files) by adding additional file readers.
2. **Model Refinement:** Given that language models evolve daily, it will be necessary to adopt newer language models for the application's improvement. This will enhance the quality of responses, especially for specialized documents.
3. **Explanation and Optimization:** As the volume of data grows, the application's processing speed might slow down, particularly during document processing and answer generation. Using optimization techniques like threading and parallelization would make the application more scalable.
4. **User Feedback**: To achieve better results, a feedback mechanism could be integrated where users rate the answers. This data can be used to further improve the model.
5. **Enhanced Search Functionality:** Additional search features could be added, such as filtering documents by sections or metadata (e.g., author, date), which would make it easier for users to find information within large documents.

IV. Conclusion

An application based on NLP technology has been developed that allows users to quickly receive answers to questions based on the content of an uploaded PDF file. The web application uses modern technologies such as Flask, LangChain, Chroma, and OpenAI models like Text-embedding-3-large and GPT4o to create a question-answering system based on PDF documents. It is designed to meet user demands quickly and accurately, making it versatile and useful across various fields.

The application is extendable and features high adaptability, enabling expansion with functionalities such as multilingual support, support for other file types, and faster real-time responses.

The resulting application is an example of how language models can be utilized. This presents an important opportunity for developers to integrate AI models into their own projects.

This project significantly simplifies the process of working with documents, allowing users to obtain highly relevant and context-based answers with minimal effort. The application helps save time and resources and substantially eases the process of managing digital documents for users.

Acknowledgment

The work was supported by the Shota Rustaveli National Science Foundation of Georgia. Grant #SC-24-753.

References

[1] A. Rayhan, R. Kinzler, R. Rayhan. "Natural Language Processing: Transforming How Machines Understand Human Language", *Conference: The development of Artificial General Intelligence*, 2023, DOI:10.13140/RG.2.2.34900.99200.

[2] D. Jurafsky & J. H. Martin. *Speech and Language Processing*, Pearson, 614 p., 2025.

[3] B. Min, H. Ross, E. Sulem, et all. "Recent Advances in Natural Language Processing via Large Pre-Trained Language Models: A Survey", 2021.

[4] Y. Goldberg, *Neural Network Methods for Natural Language Processing*, Springer Cham, 287 p., 2017.

[5] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions", *Commun, ACM*, vol.51, issue 1, pp. 117–122, 2008, DOI: 10.1145/1327452.1327494

[6] Wei-Meng Lee, "Exploring LangChain: A Practical Approach to Language Models and Retrieval-Augmented Generation (RAG)", *Published in: CODE Magazine*, 2025.

[7] H. Jégou, M. Douze, C. Schmid, "Product Quantization for Nearest Neighbor Search", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117-128, Jan. 2011.

[8] Barum Park. "What is Cosine Similarity?", *Department of Sociology, Cornell University*. 2023, CosineSimilarity.pdf.

[9] S. Tsiramua, H. Meladze, T. Davitashvili, D. Bitmalkishev, T. Elpakidze, "A Q&A system based on AI and NLP models", *XV International Conference of the Georgian Mathematical Union*, Batumi, Georgia, The Book of Abstracts, p.202, 2025.