Retrieval-Augmented Generation implementation for Wikipedia API

Creating specific knowledge assistant, Winter 2024
Warsaw University of Technology

Stanisław Kurzatkowski
01150709@pw.edu.pl

Filip Mieszkowski
01171165@pw.edu.pl

Jakub Szypuła
01142147@pw.edu.pl

1 Project approach and current state

The goal of the project is developing a model which can serve as a specific knowledge assistant. The idea to achieve it is to implement the Retrieval - Augmented Generation (RAG) to already existing Large Language Model (LLM). Since Wikipedia API serves as the main corpus of documents, a subset of articles related to a particular topic (for example: Star Wars) is extracted. This part is executed in $create\_database.ipynb$ file. Next, the external vector database is created.

database ready (database with Star Wars articles is named $wiki\_db.7z$ and is available in GitHub project repository, so creating new database is not necessary, it takes some time), one can proceed to start using the model. Example usage is shown in $WikiRAG.ipynb$. Notice the imported files, one of the is $wikirag.py$. This file contains WikiRAG class, which is responsible for methods like querying the model or searching the database for context corresponding to user query (retriever module). Querying the model is per-
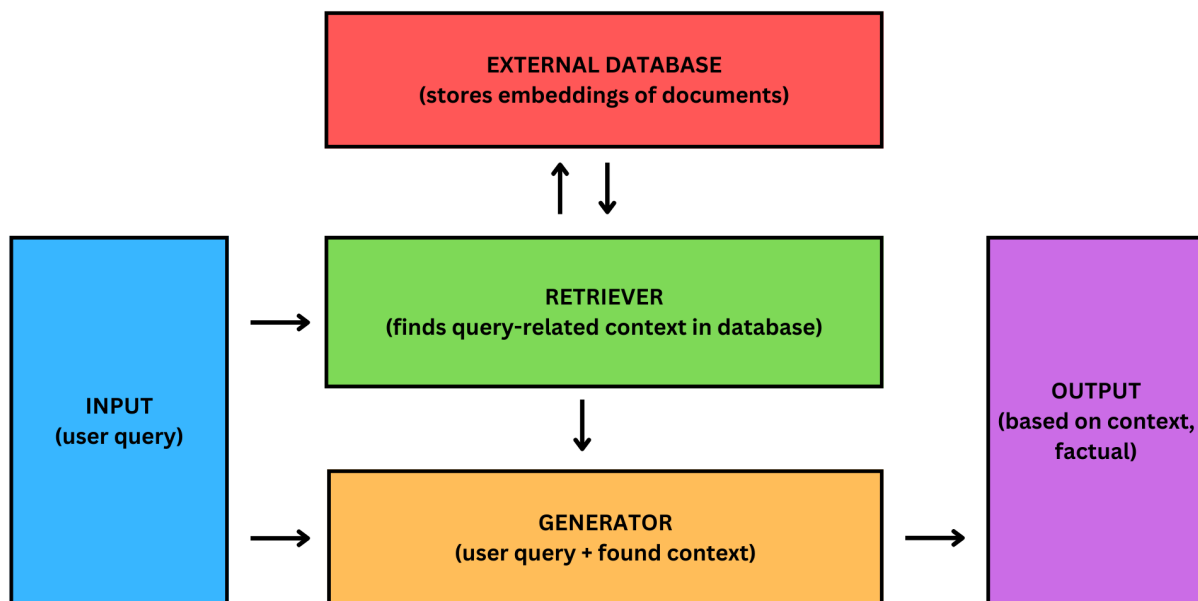
This part consists of 2 stages: chunking the text and dense embedding of obtained chunks. After, the embedding procedure is over, the vector database is created, which stores the embeddings of chunks of texts and keys, used to identify the actual piece of text behind the embedding. Since the real text is not stored in the external database, it is fairly light. This part is also executed in $create\_database.ipynb$. With external

formed with $query()$ method of WikiRAG class. Inside of it, query-based fusion takes place, which comes down to querying the model with a particular query-template filled with the user question and found context (retrievals). To run the $WikiRAG.ipynb$ file, one needs to have at least the 'llama3.2:1b' model installed locally, because this pretrained model was chosen for the generator module.

2 Retriever part

The retriever module consists of several parts; each will be briefly described below:

2.1 Chunking

Chunking is the process of splitting text (in our case, it is the text of Wikipedia articles) into smaller pieces. In the main database $wiki_db.7z$, subsections are taken as chunks, but different approaches were tested, in particular by splitting the text into sentences. On the one hand, taking long text as a chunk will lead to noninformative embeddings, on the other hand, taking long text as a chunk will lead to noninformative embeddings. However, small chunks may miss important context.

2.2 Encoding

In the encoding process, we create an embedding vector for each chunk of text. Among many different approaches, we decided to use dense, LLM-based encoding. $ada - 002$ encoder by $OpenAI$ is used, providing embedding of the 1536 dimensions. In case of this model, cosine similarity is usually used to measure the distance between data points. This is implemented by API calls to the model in the file $embedding.py$.

2.3 Creating Vector Database

To store the embeddings, we decided to use $faiss.IndexFlatL2$. A special database wrapper $VectorDatabaseWraper$ is created in the $vector_database.py$ file. When the database is saved to the file, a folder is created. Inside the folder there are two files: $database.index$ and $metadata.json$, since $faiss.IndexFlatL2$ does not provide storing metadata on its own. The name of the folder is considered to be the database name. Importantly, only metadata are stored. If needed, the original chunk of text is again delivered by Wikipedia API, using the stored metadata.

2.4 Quering the database

Quering the database is the most important part of the RAG architecture. In our case it is realized by the $faiss.IndexFlatL2.search$ function. This function returns k nearest neighbours of the query vector. Importantly, this function uses $L2$ (euclidean) metrics, differently from the encoder. This inconsistency may be the reason of some drawbacks described later in ths paper. However,

it should not have a major influence on performance, as points close to each other in the cosine similarity are close to each other in the $L2$ distance and vice versa. The approximate nearest neighbor algorithm that lies under the hood of the $search$ method is also the most time-consuming process. The test on the $wiki_db.7z$ shows that in case of $k = 100$ (number of records to be returned) the average time of response of the database is $3 - 5$ miliseconds, whereas in case of $k = 1000$ - around $6 - 9$ milliseconds. The database contains 7806 records. Of course, the reasonable values of k are much smaller, as there is no point in retrieving that many results.

3 Results part

3.1 Setup

We performed qualitative tests, full test runs and results can be seen in the notebook `WikiRAG.ipynb` and the corresponding `.csv` file with results saved.

Full table with questions and results present in appendix A.

For our results we used query-based fusion, where we extracted 20 subsections from Wikipedia for context. We had 20 questions prepared, split into four categories:

1. Characters.

2. Worldbuilding.

3. Real-life details (i.e. production of actual movies, fan reactions, reception to different star wars media, actors).

4. Trivia.

We prepared three levels of difficulty for questions:

- Easy.

- Medium.

- Hard.

For evaluation, we used three approaches:

- Ollama 3.2 model with 1 billion parameters without RAG. This was our "baseline" model to see how well RAG improves the data.

- Our RAG that uses Ollama 3.2 model with 1 billion parameters as generator. This model was used due to its size, as larger models hosted locally took a lot of time to process queries.

- Ollama 3.1 model with 8 billion parameters without RAG. This was used as a benchmark what a more complex model can achieve.

All our models were setup locally (which also means that all our experiments were run locally too).
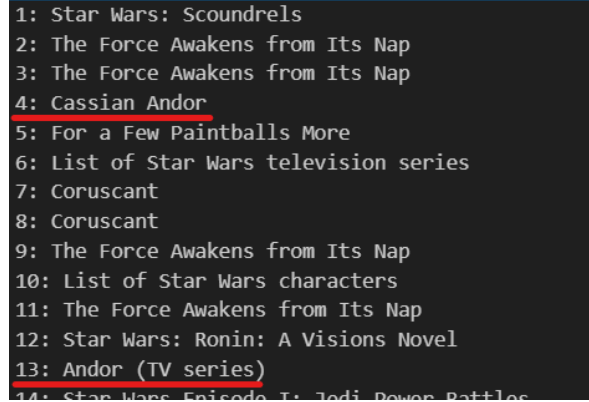
3.2 Discussion of results

Each question was ranked manually on scale -2 to +2, where -2 is "wholly incorrect" while 2 is "totally correct." In general, for our Proof of Concept, WikiRAG performed on average worse than the base model. This was usually due to situations where the smaller model failed to retrieve relevant information from the provided context.

However, it is important to note that our RAG is not always strictly worse than the base Ollama 3.2 model. Consider the question: *What is the name of the planet where the Jedi Order built their temple, later used as the Galactic Empire's headquarters?*. The correct answer to this question is "Coruscant", and our WikiRAG provides it, while the base Ollama model responds with "Dantooine."

One big advantage of basing our database on Wikipedia is access to recent information, which large pre-trained models do not have. For example, consider the question *What is the name of the prison complex Cassian Andor is sent to in Andor?*. Base Ollama models struggle with answering this question, because *Andor* is a relatively new TV show. Unfortunately, WikiRAG replied with a "The prison complex where Cassian Andor is sent, according to the text, is called Jabba's Hut Jedi Preschool." This is obviously not the case.

We decided to explore the reasoning behind this further, and we found that among top 20 retrieved documents (see 2) only two subsections were relevant directly to the character of Cassian Andor. We believe that improving performance of the retriever will lead us to better results.

Also, interestingly, additional data can apparently confuse the model. For example, the simple question *Who directed The Empire Strikes Back?* was answered correctly by basic Ollama models (Irvin Kershner) but WikiRAG answered George Lucas.


```
1: Star Wars: Scoundrels
2: The Force Awakens from Its Nap
3: The Force Awakens from Its Nap
4: Cassian Andor
5: For a Few Paintballs More
6: List of Star Wars television series
7: Coruscant
8: Coruscant
9: The Force Awakens from Its Nap
10: List of Star Wars characters
11: The Force Awakens from Its Nap
12: Star Wars: Ronin: A Visions Novel
13: Andor (TV series)
14: Star Wars Episode I: Jedi Power Battles
```

Figure 1: List of the most relevant articles produced for question regarding Cassian Andor's prison. Names are repeated, because each entry is a separate section.

In one case, our WikiRAG replied in a confusing manner (see **??**). This occured only once (the question was *What species is Chewbacca, and where is he from?*; the answer is Wookie and Kashyyk respectively). We suspect that larger amounts of text in query may have confused the smaller model into generating incoherent output.

4 Challenges to overcome

The results obtained at this stage of the project should be improved. Main investigation should focus on the retrievals quality. In some cases, provided context is inaccurate or insufficient.

Further experiments with the size of the chunks of text should take place to find an optimal chunking strategy. At this point, most results come from RAG enforced model with database created based on chunks of text corresponding to sections of Wikipedia articles. This kind of chunking might result in chunks being too large to embed them and maintain their meaning, especially that the embedded query is usually much shorter text.

Experimenting with different generator modules, choosing different pretrained, smaller models to expose RAG influence on the model outcome, model extended life-time and restricting model halucinations thanks to this solution.

Considering different databases to work with data that are more unlikely to be seen by the model during its training, since Wikipedia in popular source of articles.

Figure 2: Example of a not entirely gramatically or coherent answer.