RAG for searching for answers in school literature
Midterm Project Report for NLP Course, Winter 2024

Maja Andrzejczuk
Warsaw University of Technology
01161548@pw.edu.pl

Piotr Bielecki
Warsaw University of Technology
01161399@pw.edu.pl

Jakub Kasprzak
Warsaw University of Technology
01161454@pw.edu.pl

Maciej Orsłowski
Warsaw University of Technology
01161526@pw.edu.pl

Paweł Gelar
Warsaw University of Technology
01161422@pw.edu.pl

supervisor: Anna Wróblewska
Warsaw University of Technology
anna.wroblewska1@pw.edu.pl

Abstract

Retrieval-Augmented Generation (RAG) is becoming an increasingly popular technique of improving Large Language Models (LLMs) accuracy and reliability. This project focuses on developing a RAG model designed to query the content of a school reading effectively (e.g. an assigned book). The model will retrieve relevant information from the text, enabling students to interact with the book's content in a question-answer format. It will be joined with an LLM for easier interaction and possibly even verification of student responses.

1 Introduction

Retrieval-Augmented Generation is a crucial branch of Natural Language Processing. Its base use is to locate document passages that best answer queries made by the user. Such models can be combined with Language Models (LMs) to provide a user-friendly interface for document interaction. Combining RAG models with LMs is a prevalent approach with other use cases, such as improving the reliability of language model responses (Li et al., 2024). This is because providing a base for a model's response makes it easier to avoid hallucinations, but it also allows verifying the language model's output. This also allows us to provide up-to-date data to a language model, which on its own relies on training data limited to a certain moment in the past.

Retrieval-Augmented Generation is split into a few stages (Gao et al., 2024). The first step is to collect the documents and split them into chunks, often referred to as passages. These passages are then indexed and stored in a vector database (Pan et al., 2023). Such databases index the passages to allow for quick searching for them based on some query. This capability is used to then search for relevant documents when a user submits a query, for example, a question regarding some book. The indexing is possible thanks to the documents contents being converted to embeddings, that is, a numerical representation of text. Such embeddings are normally performed by machine learning models. Quick search time allows for retrieval of some documents, which then proceed to the next stage, which is referred to as post-retrieval stage. This step is not a part of a traditional RAG pipeline, however nowadays is often added to further boost the performance of RAG systems. Documents here are ranked to allow for more efficient injection into a query for the large language model. This way, less relevant documents can be omitted, allowing LLM to focus on the most important data, instead of overloading it with tons of documents. The reason why reranking is done after retrieval is that it is less efficient when performed on numerous documents. This way, numerous relevant documents can first be searched for using a vector database, and then a more accurate relevance measurement can be performed on the top k documents retrieved from the database by a reranker model. Finally, prompt engineering techniques are used to combine the original user query with the contents of the documents into a final query for the LLM, which then provides an answer to the user.

Our project will involve implementing a RAG system for information retrieval from school literature readings. We believe this domain is a perfect use case for such models, and to the best of our knowledge such systems have not been implemented before. We aim at utilising the power of language models to help the students search through their school readings, thus making the studying process significantly more efficient in a world where time is a valuable but limited currency. We also intend to combine our retrieval model with a language model. This will make it easier to use for users, as they will be able to receive the answers to their questions in natural language.

Our goal is to create a system, where students will be able to ask questions related to any given school literature. The system will then point the student to a relevant fragment of the book, which the student can use to answer their question. Additionally, if we integrate this solution with a language model, the model can also use the extracted fragment to provide an answer to the question, which will improve the user experience.

Our solution will be developed in the Python programming language, most likely in the *Google Colab* environment, because it provides access to GPU, which significantly speeds up computations. A detailed concept for our solution is described in section 3.

2 SOTA Review

2.1 Related works

To deepen our knowledge in the selected area of research, we explored several state-of-the-art works related to Retrieval-Augmented Generation. While we might not eventually incorporate most of them into our project for various reasons, they give us valuable insights into various aspects of the topic. Therefore, we decided to explore these works in hope of finding inspiration for our work, and perhaps even ideas for future work on our system after the course project is concluded. Let us then explore some of the modern approaches in this field.

Combining Large Language Models with RAG is an effective way to make the models more trustworthy, as their answers can be verified using the parts of the document which were selected to generate the response. This paradigm is called Verifiable Generation and one of the proposed approaches is LLatrieval, or *Large Language Model Verified Retrieval* (Li et al., 2024). This method addresses the issue of RAG models hindering the performance of the whole pipeline due to them being significantly less complex than LLMs, thus rendering the pipeline less effective than standalone LLMs. It introduces a verify-update iteration. In the *verify* step, the LMM provides feedback to the RAG model on the document it selected as a base for generating the answer. This feedback then can be used to avoid relying on low-quality retrievals. If this is the case, the next step, *update*, aims to update and improve the retrieval. Through experiments, the authors of this framework were able to achieve new state-of-the-art results.

Document retrieval is key to obtain an accurate response to queries in RAG systems. One of the works proposes *Unsupervised Passage Re-ranker*, or UPR in short (Sachan et al., 2022), which aims to improve passage (document) retrieval process. Its main feature is the zero-shot generation of questions when re-ranking. It then utilises a pre-trained LLM, without any fine-tuning, to estimate the probability of obtaining a question given a passage text. The zero-shot approach makes the evaluation process more robust and forces a model to better explain its generated questions. The authors prove the effectiveness of this approach compared to other state-of-the-art solutions and claim that the gains from using this method will grow with the improvements of available language models. While our work plan does not involve utilising this method in the reranking stage, we might consider experimenting with it should the time allow us to do so, or, more likely, as a part of future work on our solution after the project.

A common problem when developing systems based on RAG is their evaluation. This is because it heavily relies on manual human work of domain experts, which is time-consuming and expensive.

To minimise the reliance on this aspect, a RAGAS framework was proposed (Es et al., 2023). The authors stress that the biggest strength of this solution is its ability to evaluate RAG systems even without access to ground-truth answers to evaluation questions, which might be useful in our setting. RAGAS measures context relevance (is the retrieved passage relevant to the query), answer faithfulness (is the answer generated relevant to the retrieved passage, or is it hallucinated), and answer relevance (is the answer relevant to both the query and the passage). RAGAS utilises an LLM under the hood to perform evaluations. While the authors use powerful models in their implementation, the package allows for using more lightweight models.

Another framework for automatic RAG evaluation, which is even newer, is ARES, which stands for *Automated RAG Evaluation System* (Saad-Falcon et al., 2024). It works by leveraging LLM judges, trained on queries, passages, and answers. Queries and answers are generated synthetically by LLMs using provided passages. These judges are then applied to assess query-document-answer triples obtained from any RAG system. ARES evaluates the same three metrics as RAGAS, and three separate LLM judges are used for this. This framework is proven to be more accurate than other modern evaluation approaches including RAGAS, while maintaining a low cost of evaluation, with reduced need for human annotations. However, one of its core limitations are high computational requirements, which might eventually make it hard for us to utilise.

There are also attempts at incorporating the ideas of Active Learning into Retrieval-Augmented Generation, most notably a *Forward-Looking Active REtrieval augmented generation* (FLARE) method has been proposed (Jiang et al., 2023). Its goal is to address the issue of hallucinations in language models, by actively retrieving external documents when generating answers to provide more accurate response. It is different to most models, which perform such retrieval only once upon receiving an input, and then use it to generate the whole response. Some more advanced models might perform this action multiple times in fixed intervals, but often such retrievals are performed when they are not necessary. In contrast, FLARE retrieves only when it generates low-confidence tokens, i.e. part of the response that likely could be incorrect. It then retrieves the documents which then are used to regenerate the response, leading to it being more accurate. The use of active learning principles also prevents unnecessary retrievals.

2.2 Datasets

The data we intend to be using comes from a free library of eBooks, named *Project Gutenberg* (`https://www.gutenberg.org/`). It claims to contain over 70,000 different eBooks, which includes school literature that we are interested in. We will select some examples of school readings, and use them in our project. They are available in the plain text format, but can also be converted to PDF – our solution will support loading data from both formats. There is no API available for this data, which means collecting it will require substantial manual work. However, it is definitely feasible considering the volume of the data that we can realistically work on given the computational resources available to us. We will collect them in a *GitHub* repository, which we can clone in *Colab* to have easy access to them.

Regarding the model evaluation datasets, unfortunately we were not able to locate a free dataset with school literature retrieval based exam questions which could be used for evaluation of our system. We could keep searching, but as an alternative, we could manually design question-answer pairs for our system. With five people in the team, it should be feasible to develop an exam dataset of at least 150 entries.

2.3 Models

We intend to use open-source models, and we will focus on pre-trained ones, since we will not have high computational power at our disposal during the project. There are three types of models we were looking for:

- embedding models,
- re-ranker cross-encoder models,
- large language models.

We found several models on HuggingFace platform, which could be useful for our system. Let us now discuss them in more detail. Starting with embedding models, the first model we consider using is called *all-MiniLM-L6-v2*, which has 22.7M parameters. It is a part of *sentence-transformers* library (Reimers and Gurevych, 2019). This library includes numerous models aiming to generate embeddings for text input data. These models are based on Sentence-BERT, which itself is a modification of BERT (Devlin et al., 2019). The authors of Sentence-BERT claim that cross encoder embedded into BERT is not efficient, and their proposed method is based on siamese networks. This allows it to generate high-quality fixed-size embeddings, and the *all-MiniLM-L6-v2* model, which we intend to use, is a lightweight example of such a model, and will allow us to generate embeddings for passages for efficient storage in a vector database.

A larger alternative for this model would be *bge-large-en-v1.5* (Xiao et al., 2023), which has 335M parameters. This model comes from a large family of models named BGE (BAAI General Embeddings), which originally was introduced in a paper that proposed a method for embeddings for Chinese language. However, this family contains not just models for Chinese language, but also Multilingual, and models which work just for the English language, and the model we chose is exactly one of such.

When it comes to re-ranking models, the ones that we are considering using are *bge-reranker-large* with 560M parameters, and *bge-reranker-base* with 278M parameters (Xiao et al., 2023). They also come from the BGE model family. These are cross-encoder models, which utilise attention mechanism to rank the retrieved documents. Contrary to embedding models, these models take user query and documents and output similarity scores. These scores are not constrained to any range.

Finally, there is an LLM which will generate the final outputs. We are looking for a lightweight model due to limited resources. One that we could potentially use is Cerebras GPT (Dey et al., 2023), which is available for free on HuggingFace. It comes in seven variants, ranging from 111M to 13B parameters. We will most likely use the variant with 2.7B parameters, as in our opinion it provides a good balance between quality and computational requirements. These models were released by Cerebras Systems and are pre-trained on the Pile dataset (Gao et al., 2020). The authors show in their experiments that they are compute-optimal, at least compared to other state-of-the-art models. They claim that models from this family offer the most quality out of other models trained on the same budget, due to them using the latest advancements in LLMs training and scaling techniques.

Alternatively, we could also try *Llama3.2* (Touvron et al., 2023), which is a lightweight release from a group of *Llama 3* models, and comes in two sizes: one with 1B parameters, and the other with 3B parameters. This gives us flexibility when it comes to choosing the most optimal one. Llama models are released by Meta AI and are trained solely on publicly available data, unlike some of the other LLMs. What is very useful from our perspective is that the authors state Gutenberg as one of the data sources on which their model was trained – this is the same data source that we intend to source the school readings from. The only problem with using this model is that while it is completely free, it requires approval from its owners. While it is usually granted for students, one needs to provide their personal details in order to obtain it. Once the access is granted, an access key from HuggingFace is needed to use it.

A third alternative that we found is a model called *Qwen2.5-7B-Instruct* (Team, 2024). This model is the newest release from the Qwen2 family of models (Yang et al., 2024), which is open-source and available without any restrictions. It is a larger architecture than all the other proposed, however it is quantised to *Int4* data type, which enables its usage in environments with less computational power, whilst maintaining solid performance. Its parameter number varies from 0.5 up to 72 billion, depending on the variant.

All the models that we listed above can be loaded through either *transformers* (Wolf et al., 2020), or *sentence-transformers* (Reimers and Gurevych, 2019) packages. Instructions on how to do that are included on these models' homepages on HuggingFace.

2.4 Vector database

As mentioned earlier, vector database is a key component of any RAG systems as it allows for quick similarity searches used when retrieving documents relevant to a query. The vector database that we intend to use is Meta's FAISS (Douze et al., 2024). It is an open-source library. The authors point out the flexibility of their library in that it can use multiple indexing methods, instead of just some preselected one. FAISS does not generate embeddings on its own, and it requires an external embedding model to generate vectors for the data.

3 Work plan

Let us now discuss our plan for this project. Our work will start by performing exploratory data analysis on our data. We found a *LangChain* (Chase, 2022) library, which will not only help us load the PDF documents, but also contains other useful functionalities for RAG tasks, which we may utilise further in the project. Once we have investigated our data, we will start pre-processing it. There are many state-of-the-art Python libraries we could utilise for this task, and beside *LangChain*, the one we find the most promising is *Unstructured* (`https://unstructured.io/`). Both can be easily integrated with each other and provide many functions for pre-processing of PDF documents. We will definitely need to split selected books into chunks, which we could do, for example, by utilising their structure and dividing them by paragraphs, or by just splitting them into fixed-sized chunks. We also need to clean them, for example, by removing redundant whitespace. However, the exact preprocessing steps will need to be decided only once we have explored our data.

Simultaneously, we will split our efforts to also work on assessing the models. We will need to verify that:

- they load correctly in our environment,
- they are feasible to run and use (in terms of computational resources).

We plan to perform these checks early in the project, so that in case anything goes wrong, we have time to find emergency alternatives for these models.

Once the documents are pre-processed, and the models are loaded, we can feed the documents to the embedding model. It will generate vector embeddings, which we will need to store in a vector database, which allows for efficient vector similarity search, and also is integrated into *LangChain* library. At this point, we will be able to take inputs from a user, and retrieve relevant documents from our database. Then, these documents will be passed to a re-ranker cross-encoder model, which will rank the retrieved documents to help select the most relevant one. We will also add a language model to act as an interface for interacting with our system.

At this point, we will have the prototype versions of all the key components in our system (pre-processing and RAG models). We will test them on an example book from our dataset. This will serve as the proof-of-concept for our solution, and therefore we plan to arrive at this stage for the midterm checkpoint in December.

Afterwards, we will work on improving individual components and coupling them all together. Our goal will be to create a reusable, customisable pipeline. Internally, it will consist of two parts. The first part will be responsible for loading the documents, generating embeddings, and storing them in a vector database. Since this is a lengthy process, we plan on performing it once for our whole dataset and preserving the vector database for later use. The second part will perform a search for a given query, using the database from earlier. However, the users will have a choice to either utilise our ready-made database, or to create theirs by uploading their own books. Additionally, the whole pipeline will be contained within a single class, which will make it easier to use. Listing 1 shows an early prototype vision of the design of this pipeline (the names of the class and methods are going to be more specific in the final solution).

Listing 1: Prototype design of the pipeline

```
# Default database
pipeline = Pipeline.default()

# Custom books
pipeline = Pipeline(
    books=['./my_book.pdf']
)

# Using the pipeline
pipeline.query(
    'year of the Easter Rising'
)
```

At this stage, with our system working, we will conduct multiple experiments to test the quality of our solution. Let us discuss our plan for the evaluation of our RAG system. Once we have the evaluation dataset as discussed in one of the earlier sections, we could proceed with one of the following:

- Utilise a framework such as ARES (Saad-Falcon et al., 2024). The strength of ARES is that it requires very little human annotations, however our initial tests proved that it is computationally heavy and thus might not be feasible to run. However, it will be a good idea to at least try this solution in the experiments phase;

- Utilise a lighter alternative to ARES. An example of such is RAGAS (Es et al., 2023), which, while less accurate than ARES, is still a state-of-the-art method which is a bit more lightweight in terms of computational requirements;

- If both frameworks proposed above end up not being feasible to use, and we are unable to find another alternative, we can always fall back to using an LLM judge. This requires prompt engineering, which aim is to make a query for a judge that will output concrete metrics for each sample generated by our system. At this stage, we could either utilise some lightweight LLM model such as Llama 3.2 (Touvron et al., 2023), or a larger model like GPT-3.5 through its web interface (OpenAI, 2024) by manually pasting the prompts and gathering the responses.

Once we make sure that everything works, we will export the whole pipeline from our notebook into a repository on GitHub (the same one where the data will be stored), so that we can easily access it by cloning the repository in our *Colab* environment. This will also declutter the notebook, and make it easier to use by the users. This will mark the end of the project, which we will present on the deadline day in January. A timeline for our work is presented in Table 1 (dates in bold are checkpoints).

Date	Planned progress
27/11	Project proposal
04/12	EDA, verifying models feasibility
11/12	Proof of concept
18/12	Reviews
08/01	Reusable pipeline
15/01	Experiments
22/01	Final presentation and report

Table 1: Detailed work schedule

4 Proof of concept

In this section, we describe the work we have done on the proof of concept. We focus here on our methodology, explaining the environment, data transformations, which models we have eventually used, their parameters, etc., in order to provide grounds for reproducibility. We also mention our first findings related to this system.

4.1 Environment

Let us begin by discussing the environment in which we developed our proof of concept. We used Google Colab, and the code required 6GB of Google Drive storage space to store all the data and models downloaded by the code. We run our code on the *T4 GPU* runtime. This is critical, as running this code on CPU would take significantly more time, especially the steps involving models inference. The code cells that can take longer than 15 seconds to run are marked in the notebook with a comment, which informs of the average execution time (in the *T4 GPU* runtime). Note that we did not use any paid plan for the runtime. This made the development difficult due to occasional disconnects, meaning we would have only up to around two hours of development each day, and after that we would be forced to wait around 24 hours before we could connect to any GPU runtime again. This forced us to work on the project in short but regular bursts.

Additionally, right at the beginning of the notebook, we set all relevant random number generators seeds. This ensures that the code in the notebook produces the same results every time it is run.

4.2 Data loading

For our analysis, we collected a collection of books available through Project Gutenberg, which is a large online library of free e-books. These books are stored in our repository in the `data` folder (link: `https://github.com/andrzejczukm/NLP-RAG`). In total, the dataset includes 35 books, mostly in `.txt` and `.pdf` formats. These texts were carefully selected to include only books known worldwide and recorded for use in our analysis. We conducted a detailed exploratory data analysis (EDA) to better understand the structure and characteristics of the texts to help guide the next stages of our NLP project.

4.3 Exploratory Data Analysis

The EDA process involved analyzing the data structure and results of splitting input texts into chunks for further NLP processing. Initially, we encountered problems with chunk splitting, especially for documents in `.txt` format. These problems stemmed from inconsistencies in the way we handled line splits and separators. The following section outlines our approach and subsequent improvements.

4.3.1 Initial Analysis

In our initial analysis, we examined the distribution of passages across documents and the variability in passage lengths. These visualizations revealed significant inconsistencies in chunk sizes, particularly for `.txt` files.

The first plot (Figure 1) shows the number of passages generated per document during the initial chunking process. This revealed significant variation, indicating inconsistent chunking logic. Some documents were split into far too many passages due to the aforementioned newline issues. The next plot (Figure 6) shows the distribution of passage lengths for each document. While we aimed for a consistent passage length of approximately 2000 characters, the results deviated significantly. On the logarithmic scale, we can see that five documents that exhibited especially unusual distributions, named:

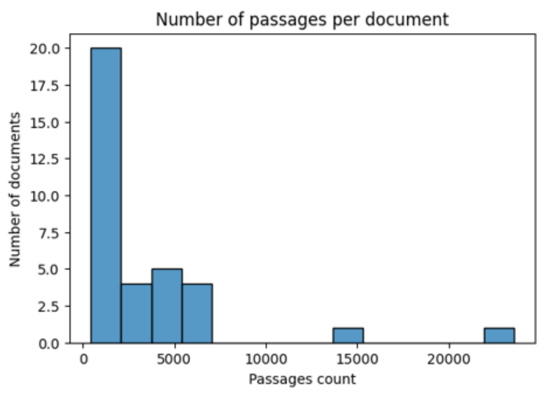


Figure 1: Number of passages per document in the initial chunking process. Notice the uneven distribution across documents.

- `kafka_metamorphosis.txt`
- `milton_paradise_lost.txt`
- `homer_iliad.txt`
- `kafka_the_trial.txt`
- `conrad_heart_of_darkness.txt`

The above-mentioned files are identified by the nearly invisible or highly compressed boxes, suggesting irregular or extreme passage lengths compared to other documents.

The histogram in Figure 2 provides a global view of passage lengths across all documents. While the target passage length was 2000 characters, many passages were excessively short.

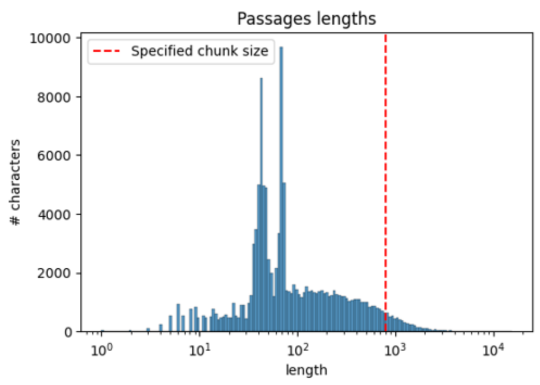


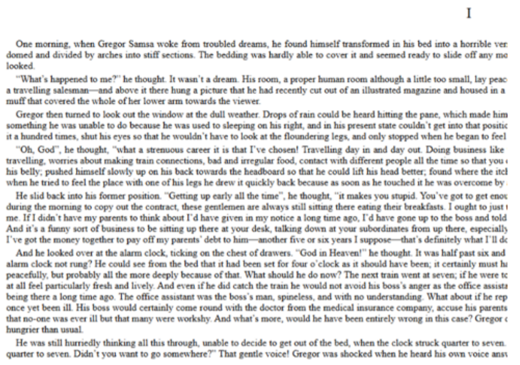
Figure 2: Histogram of passage lengths before improvements. Note the cluster of very short passages, especially in `.txt` files.

4.3.2 Problem Finding and Resolution

During the initial analysis, we encountered significant discrepancies in the way chunks were generated from `.pdf` and `.txt` files sourced from the Gutenberg library. The problem was caused by differences in the formatting of the two file types. While `.pdf` files retain their original structure, `.txt` files from Gutenberg often contain artificial line divisions that disrupt the natural flow of text, making it difficult to properly divide content into meaningful chunks.

To better understand the issue, we compared sample of `.pdf` and `.txt` files sourced from Gutenberg. As seen in Figures 3 and 4, the formatting of the two file types is quite different. Figure 3 shows a `.pdf` file with well-structured content, where paragraphs are properly aligned and separated, with no artificial line breaks.

In contrast, Figure 4 illustrates a `.txt` file from Gutenberg, where artificial line breaks are inserted between lines of text, causing fragmentation and making it difficult for the chunking process to divide the text appropriately. Upon



I

One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible ver
domed and divided by arches into stiff sections. The bedding was hardly able to cover it and seemed ready to slide off any mo
looked.

"What's happened to me?" he thought. It wasn't a dream. His room, a proper human room although a little too small, lay peac
a travelling salesman—and above it there hung a picture that he had recently cut out of an illustrated magazine and housed in a
muff that covered the whole of her lower arm towards the viewer.

Gregor then turned to look out the window at the dull weather. Drops of rain could be heard hitting the pane, which made him
something he was unable to do because he was used to sleeping on his right, and in his present state couldn't get into that positic
it a hundred times, shut his eyes so that he wouldn't have to look at the floundering legs, and only stopped when he began to feel

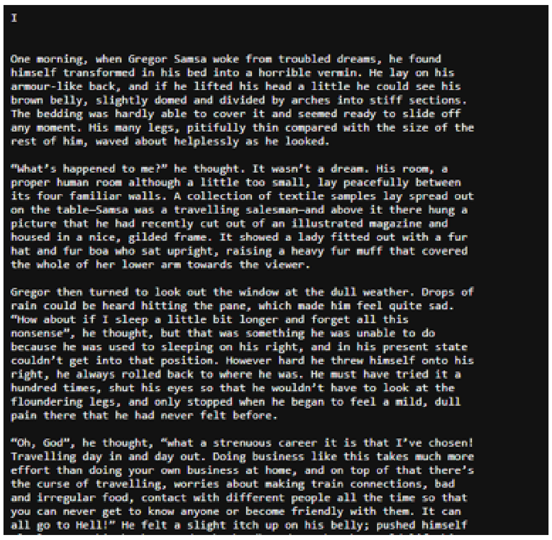
"Oh, God", he thought, "what a strenuous career it is that I've chosen! Travelling day in and day out. Doing business like
travelling, worries about making train connections, bad and irregular food, contact with different people all the time so that you
his belly; pushed himself slowly up on his back towards the headboard so that he could lift his head better; found where the itc
when he tried to feel the place with one of his legs he drew it quickly back because as soon as he touched it he was overcome by

He slid back into his former position. "Getting up early all the time", he thought, "it makes you stupid. You've got to get enou
during the morning to copy out the contract, these gentlemen are always still sitting there eating their breakfasts. I ought to just
me. If I didn't have my parents to think about I'd have given in my notice a long time ago, I'd have gone up to the boss and told
And it's a funny sort of business to be sitting up there at your desk, talking down at your subordinates from up there, especially
I've got the money together to pay off my parents' debt to him—another five or six years I suppose—that's definitely what I'll dc

And he looked over at the alarm clock, ticking on the chest of drawers. "God in Heaven!" he thought. It was half past six and
alarm clock not rung? He could see from the bed that it had been set for four o'clock as it should have been; it certainly must ha
peacefully, but probably all the more deeply because of that. What should he do now? The next train went at seven; if he were tc
at all feel particularly fresh and lively. And even if he did catch the train he would not avoid his boss's anger as the office assista
being there a long time ago. The office assistant was the boss's man, spineless, and with no understanding. What about if he rep
once yet been ill. His boss would certainly come round with the doctor from the medical insurance company, accuse his parents
that no-one was ever ill but that many were workshy. And what's more, would he have been entirely wrong in this case? Gregor
hungrier than usual.

He was still hurriedly thinking all this through, unable to decide to get out of the bed, when the clock struck quarter to seven.
quarter to seven. Didn't you want to go somewhere?" That gentle voice! Gregor was shocked when he heard his own voice ansv

Figure 3: Sample chunk from a .pdf file. The passage structure is consistent and reflects coherent textual units.



```
I

One morning, when Gregor Samsa woke from troubled dreams, he found
himself transformed in his bed into a horrible vermin. He lay on his
armour-like back, and if he lifted his head a little he could see his
brown belly, slightly domed and divided by arches into stiff sections.
The bedding was hardly able to cover it and seemed ready to slide off
any moment. His many legs, pitifully thin compared with the size of the
rest of him, waved about helplessly as he looked.

"What's happened to me?" he thought. It wasn't a dream. His room, a
proper human room although a little too small, lay peacefully between
its four familiar walls. A collection of textile samples lay spread out
on the table—Samsa was a travelling salesman—and above it there hung a
picture that he had recently cut out of an illustrated magazine and
housed in a nice, gilded frame. It showed a lady fitted out with a fur
hat and fur boa who sat upright, raising a heavy fur muff that covered
the whole of her lower arm towards the viewer.

Gregor then turned to look out the window at the dull weather. Drops of
rain could be heard hitting the pane, which made him feel quite sad.
"How about if I sleep a little bit longer and forget all this
nonsense", he thought, but that was something he was unable to do
because he was used to sleeping on his right, and in his present state
couldn't get into that position. However hard he threw himself onto his
right, he always rolled back to where he was. He must have tried it a
hundred times, shut his eyes so that he wouldn't have to look at the
floundering legs, and only stopped when he began to feel a mild, dull
pain there that he had never felt before.

"Oh, God", he thought, "what a strenuous career it is that I've chosen!
Travelling day in and day out. Doing business like this takes much more
effort than doing your own business at home, and on top of that there's
the curse of travelling, worries about making train connections, bad
and irregular food, contact with different people all the time so that
you can never get to know anyone or become friendly with them. It can
all go to Hell!" He felt a slight itch up on his belly; pushed himself
```

Figure 4: Sample chunk from a '.txt' file. Artificial line breaks caused fragmentation of text into incoherent passages.

identifying this issue, we applied the paragraph-grouper=group-broken-paragraphs setting during the document loading process. This setting addressed the artificial line breaks in the Gutenberg `.txt` files, grouping fragmented text into coherent paragraphs. As a result, the chunking process generated consistent and meaningful passages, allowing for better handling of the text for further analysis.

4.3.3 Improved Chunking Approach

After addressing the issue of artificial line breaks in the Gutenberg `.txt` files, we improved the chunking process by ensuring that text was divided into coherent passages with a consistent length. The result of this improvement is reflected in the following visualizations.

In Figure 7, we present boxplots of passage lengths per document after the improvements. The variability observed in the previous boxplots has been significantly reduced. The plots now demonstrate a more consistent distribution of passage lengths across all documents, with fewer extreme outliers. This indicates that the chunking approach is now more stable and effective at dividing the text into roughly equal-sized passages, as intended.

Similarly, Figure 5 shows the histogram of passage lengths after the improvements. The distribution now closely aligns with the intended chunk size of at most 2000 characters. This histogram illustrates that the chunks are more uniform in length, with fewer instances of unusually short or excessively long passages. However, it is evident that the majority of the chunks are slightly smaller than the intended 2000 characters, likely due to the nature of the text and how the splitting process handles natural language.

We ended up splitting the documents using `RecursiveCharacterTextSplitter` with separators set to double new line, full stop, question mark, and single newline characters, in that order of preference. This means that we try to split documents into chunks shorter than maximum size using double new line, if we fail then we split on full stops etc. With this approach, the only problematic book was 'Ulysses', because there were ten paragraphs without any punctuation. It is a rare quirk of this book, and we assume that it is abnormal. Even those should not be a significant problem, as they are an order of magnitude shorter than the maximum context length for the chosen LLM (5.5k letters vs 30k tokens). Around 60 (of 50k) passages are just punctuation. We can just ignore them, as the database and reranker should not return them, or we may not add them to the database.

These improvements in chunking are vital for ensuring that the data is properly structured for further NLP analysis. With consistent and coherent passages, we are now able to proceed with more accurate text processing and analysis steps.

4.4 RAG Model Performance Testing

For the proof of concept, we limited the data to just one book – "Ulysses" by James Joyce (`joyce_ulysses.pdf` in our dataset).

We begin by loading the file and splitting the

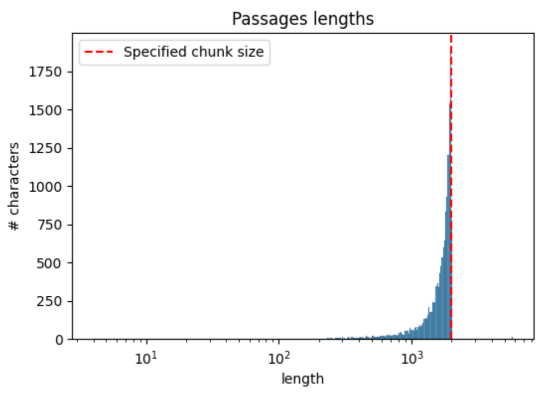


Figure 5: Histogram of passage lengths after improvements. The distribution aligns closely with the intended chunk size.

book into passages. We use a chunk size of 2000 which we believe should be enough for a question-answer system. We also set a chunk overlap of 200 to allow context to transfer between chunks. We believe that this will help if the information that needs to be retrieved is located near the boundary of some chunk. We split the document by a double newline character. We also utilise two preprocessors from the *Unstructured* library – one for removing excess whitespace (for example, when two or more white spaces occur after one another) and the other for grouping broken paragraphs (paragraphs that are split by newline characters solely for visual purposes). This resulted in 5724 passages. A great advantage of using *LangChain* is that we can load pdf and txt documents exactly the same way.

As discussed in the work plan section (Section 3), we attempted to use all the models that we listed in Section 2.3 in order to test their feasibility in our environment. The first was the embedding model, and the one we eventually selected for our proof of concept is *all-MiniLM-L6-v2*. We did try to use the bigger model that we proposed, *bge-large-en-v1.5*, however, due to its higher complexity, it was taking much longer in terms of runtime. However, in this case, we do not exclude the possibility of using it in our final solution. This is because we need the embedding model mostly in the process of creating a vector store, which, for the final solution, will need to be performed only once. Thus, the long computation time might not matter as much, provided that there are no hardware limitations. Once the vector database is created, we can then save it and load it whenever we start the system, and only use the embedding model on the query, which we assume would not be computationally expensive. However, this will require further testing at later stages of our work. Furthermore, at this stage, where the solution is highly unstable, it is still easier to work with the more lightweight model, thus the current approach uses *all-MiniLM-L6-v2*.

When it comes to the reranker model, again we tried both proposed architectures: *bge-reranker-base* and *bge-reranker-large*. Fortunately, for this model, the computation times were very quick for both variants, so we were able to use the larger model in our solution.

Finally, there is the large language model, which interacts with the user by producing the final output. Initially, we tested various models from the *CerebrasGPT* family. However, we quickly realised that their performance was underwhelming, no matter the model size. Their answers were usually highly inaccurate, and, in addition, their outputs would often repeat parts of sentences in an infinite loop. Thus, we decided to abandon these models completely and instead try the *Llama3.2* models. We attempted to utilise both 1B and 3B parameter variants, but unfortunately, the environment we used did not have enough virtual memory on the GPU to load the larger model. Thus, we decided to utilise the smaller variant with 1B parameters. However, this model's performance did not satisfy us. While it would occasionally produce correct responses, it was very inconsistent and would often hallucinate or draw incorrect conslusions from the passages. Therefore, we tried the final proposed LLM which was *Qwen2.5-7B-Instruct* quantised to *Int4* data type to enhance performance, which we use in the final proof-of-concept as described further. Choosing instruct-type model greatly increased quality of generated text, as the model always tried to answer the given questions instead of generating new ones or trying to write more of the book.

Having loaded the data and chosen the models, we constructed a prototype version of the system. Let us now discuss it. In overview, it works as outlined in the work plan section (see section 3). Thus, let us only describe the specific parameter choices for different components. Firstly, we set the number of best-matching documents retrieved from the vector database to 30. We found the value of 30 to be a good balance between the amount of information retrieved, and the computation time for the reranker model. We then use the five best passages according to the reranking model to construct the prompt for the LLM. We build it in the following way: It starts with *"Based on these passages:"*, and then the five best passages are concatenated. Then we put *"Answer this question:"*, and we paste the original query. Note that all these components of the query, as well as the passages, are concatenated with a double new line character as a separator. For now, we limit the length of the answer generated to 50 tokens.

We run a simple test on this prototype system, to verify that it works and has potential. The query that we passed to the system was: *"Who is Leopold Bloom's father?"*. While the response was abruptly cut at the end due to the maximum tokens limit, the model correctly stated that the father of Leopold Bloom is Rudolph Bloom (also referred to as Rudolf Virag). It also provided some additional minor facts about him. Upon the inspection of the passages that were used to generate the query, we saw that the answer was in both the first and the second passage, as ranked by the reranking model. The other listed facts were also clearly based on the best-ranked passages. Not only does it prove the reranker's quality, but also the quality of the retrieval from the vector database, as it was able to locate these passages from the thousands of others available. The model also did very well when it came to describing Buck Mulligan's appearance.

We tested whether the answers were in fact the result of retrieval augmentation, or if LLM was just able to answer those questions on its own. We did that by asking it the same questions but without the passages. It then answered that the Leopold Bloom's father is named David, which is straight-up wrong. It also describes Buck Mulligan's appearance confidently, but is wrong again.

To conclude our proof of concept, despite some minor problems along the way, the draft of the pipeline we provided proves that this system looks promising.

References

[Chase2022] Harrison Chase. 2022. Langchain, oct.

[Devlin et al.2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

[Dey et al.2023] Nolan Dey, Gurpreet Gosal, Zhiming, Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. 2023. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster.

[Douze et al.2024] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library.

[Es et al.2023] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. 2023. Ragas: Automated evaluation of retrieval augmented generation.

[Gao et al.2020] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The pile: An 800gb dataset of diverse text for language modeling.

[Gao et al.2024] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey.

[Jiang et al.2023] Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore, December. Association for Computational Linguistics.

[Li et al.2024] Xiaonan Li, Changtai Zhu, Linyang Li, Zhangyue Yin, Tianxiang Sun, and Xipeng Qiu. 2024. LLatrieval: LLM-verified retrieval for verifiable generation. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5453–5471, Mexico City, Mexico, June. Association for Computational Linguistics.

[OpenAI2024] OpenAI. 2024. Chatgpt (november 2024 version). `https://chat.openai.com/`. Accessed: 2024-11-27.

[Pan et al.2023] James Jie Pan, Jianguo Wang, and Guoliang Li. 2023. Survey of vector database management systems.

[Reimers and Gurevych2019] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11.

[Saad-Falcon et al.2024] Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. 2024. ARES: An automated evaluation framework for retrieval-augmented generation systems. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 338–354, Mexico City, Mexico, June. Association for Computational Linguistics.

[Sachan et al.2022] Devendra Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. 2022. Improving passage retrieval with zero-shot question generation. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3781–3797, Abu Dhabi, United Arab Emirates, December. Association for Computational Linguistics.

[Team2024] Qwen Team. 2024. Qwen2.5: A party of foundation models, September.

[Touvron et al.2023] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

[Wolf et al.2020] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Perric Cistac, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, oct. Association for Computational Linguistics. If you use this software, please cite it using these metadata.

[Xiao et al.2023] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding.

[Yang et al.2024] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

A Appendix: Division of work

The project requirements demand to provide workload for each team member at any stage of the project. Therefore, in order to comply with that, we list the contributions of each team member so far below:

- **Maja Andrzejczuk:** Abstract, Introduction, PoC, Collecting the data;
- **Piotr Bielecki:** Datasets & Work plan sections, PoC, Collecting the data;
- **Paweł Gelar:** Models & Vector database sections, PoC, Collecting the data;
- **Jakub Kasprzak:** Finding datasets & models, Work plan section, EDA, Collecting the data;
- **Maciej Orsłowski:** Related works, EDA, Collecting the data.

We state that in terms of workload and time assessment, each member has contributed equally to the project.

B Appendix: Wide plots

In this appendix we present the figures that we refer to in the text, which were too wide to nicely place in sections where they are mentioned.

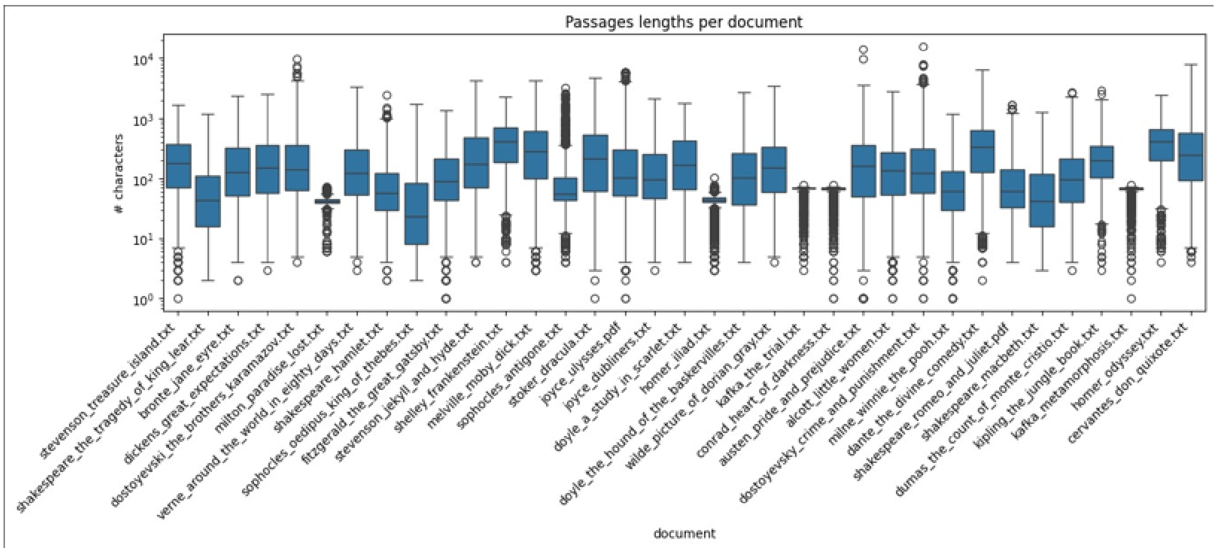


Figure 6: Boxplots of passage lengths per document before improvements. Outliers highlight extreme variability in some documents.

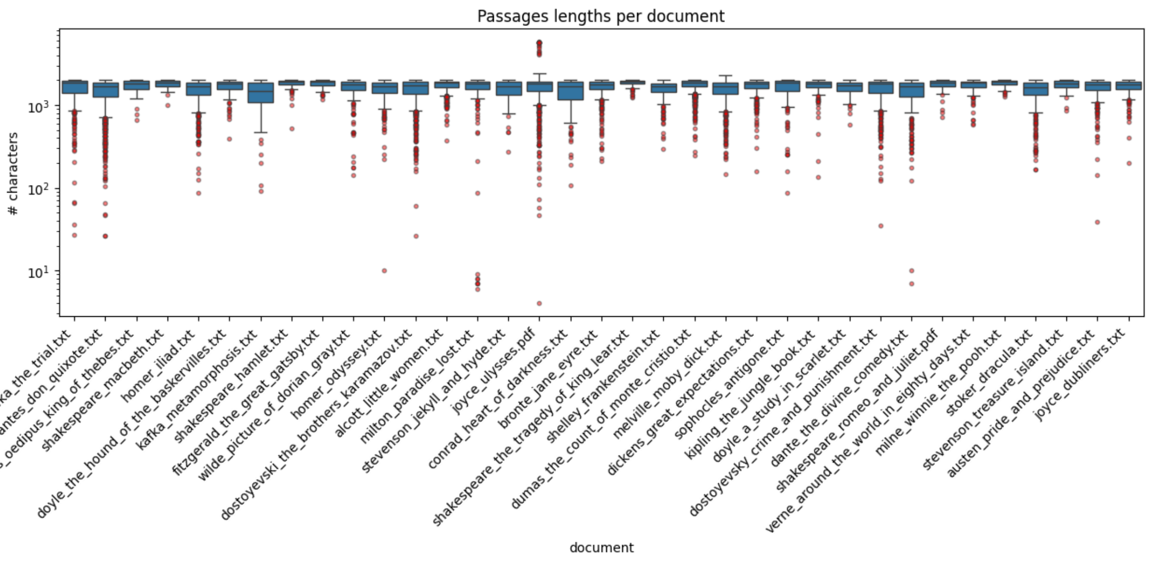


Figure 7: Boxplots of passage lengths per document after improvements. Variability across documents has been significantly reduced.