Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра інформатики

Кваліфікаційна робота

освітній ступінь - бакалавр

на тему: «**Retrieval Augmented Generation for Ukrainian Government Services: A Comparative Evaluation of the Approaches**»

Виконав: студент 4-го року навчання,

Спеціальності
122 Комп'ютерні науки

Маринич Антон Олегович

Керівник ст. в. Курочкін А.В.,

Рецензент
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою

_____

Секретар
ЕК_____
«____»_____20_____ р.

Київ – 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
доцент, к.ф.-м.н.
_____С. С. Гороховський
(підпис)
„____”_____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на кваліфікаційну роботу
студенту Мариничу Антону Олеговичу
факультету інформатики 4 курсу

ТЕМА: Retrieval Augmented Generation for Ukrainian Government Services: A Comparative Evaluation of the Approaches.
Вихідні дані:
Зміст ТЧ до кваліфікаційної роботи:

Вступ
1. Теоретичні основи
2. Огляд літератури
3. Методологія
4. Експерименти
5. Результати і аналіз
6. Висновки і подальша робота
Джерела

Дата видачі „___” _____ 2025 р.

Керівник _____
(підпис)

Завдання отримав _____

(підпис)

Тема: Retrieval Augmented Generation for Ukrainian Government Services: A Comparative Evaluation of the Approaches.

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу	Жовтень 2024	
2.	Огляд технічної літератури	Січень 2025	
3.	Генерація набору даних	Лютий 2025	
3.	Проведення порівняльних експериментів	Март 2025	
4.	Написання текстової частини роботи	Травень 2025	
5.	Захист кваліфікаційної роботи	Червень 2025	

Студент: **Маринич А. О.**

Керівник: **Курочкін А. В.**

“_____”_____

TABLE OF CONTENTS

ABSTRACT

Retrieval Augmented Generation or RAG is a method that is used to improve the quality of retrieval for LLMs, to avoid hallucinations and be aware of all the changes in the data. This approach integrates LLMs with external data source by building a vector index. This thesis presents a comprehensive study on how different RAG approaches perform in Ukrainian Governmental Services domain. I establish a non-RAG baseline using GPT-4.1-mini model and iteratively perform tests on different configurations of RAG approaches. I have also created a dataset with 500 open questions about Ukrainian Governmental Services using GPT-o4-mini-high model. My best results comparing to the baseline are 13.25% improvement in LLM Judge Score using CRAG with Hypothetical Document Embedding and Reranking and 10% improvement on Factual Correctness using CRAG with Reranking.

INTRODUCTION

In modern world of AI and LLMs there are new ways how to improve user interactions with big sources of data. This is also applied to the government services since information about them might be vital for the user. However, this is the area where LLMs can hallucinate and generate made-up answers. This would be a disaster since government related documents are quite strict and any mistakes can lead to bad consequences. The government related data also changes overtime which is not always reflected on the LLMs knowledge.

To omit those issues, we will use Retrieval Augmented Generation. It is a powerful method that integrates data sources into LLM retrieval.

This topic is relevant and important since Ukraine is moving towards a digital era. Therefore, there is a need to build a chat-bot that would answer questions about Ukrainian Government Services. The goal of this work is not to develop such an application but to explore the possibility to use RAG to improve the retrieval quality of such a bot. Since there are not many studies available in this area, it is very important to evaluate how different RAG approaches perform on such a task. Potentially, the results from this study could be used to guide the future development of such an application.

The object of this research are the RAG approaches along with GPT models that I used for retrieval and evaluation dataset generation.

The subject of this research is the process of developing and optimizing a RAG pipeline that would improve retrieval quality for questions about Ukrainian Government Services.

The main goal of this work is to iteratively review and evaluate RAG techniques on enhancing the retrieval on Ukrainian Government Services and to do so we need to create am evaluation dataset, find a reliable data

source and find the best evaluation metrics to make sure that our results are accurate. After finding the best RAG configurations I should evaluate its performance comparing to the relevant studies.

When it comes to the relevant studies there is not a lot to find. The field of Ukrainian RAG is unexplored, so I managed to find only one study that I can compare my results to. They achieve a 10% improvement in accuracy after comparing non-RAG baseline to the RAG approach.

The methodological grounds of this research are to use LangChain as my main tool to build RAG systems. I will also use Ragas Framework to get the evaluation metrics like semantic similarity, answer relevancy, answer correctness, factual correctness, LLM judge score and so on. I will also use different approaches like Naïve RAG or RAG with Reranking, or Corrective RAG to try to get the best result. The performance of the system will be tested on a dataset that I will generate. The testing process means running all the questions from the dataset on different RAG configurations with the same model temperature to be able to compare the results. The answers from the system will be compared to the reference answers from the dataset and used to compute the Ragas metrics.

SECTION 1. THEORETICAL FOUNDATIONS

1.1 Key Principles and Purposes of Retrieval Augmented Generation

What is RAG? Key Purposes of RAG.

Retrieval Augmented Generation is a powerful method for improving LLM retrieval in certain domain areas using information from external data sources. Those data sources can contain structured information, unstructured information, and semi-structured information. A good example of a structured information source is a table or a relational database. Semi-structured information can come in the JSON format. Finally, an example of unstructured data source for RAG could be a PDF document or just some text.

There are a few purposes of why this approach exists. Firstly, when the model does not have information about a certain fact, it might just make it up or hallucinate. For example, there were a lot of cases [1], when lawyers used LLMs in their work and those AI tools just made up some cases or facts about those cases which led to reputation and financial loss.

Another reason why businesses might want to use RAG is when the information that they need to retrieve is changing rapidly and LLM updates cannot keep up with those changes. If the organization uses RAG, they just can update the data source, so there is no need to retrain LLMs which costs a lot of time and money.

Privacy concerns can also be solved using RAG. For example, if a business wants to use LLMs against some private data that they possess, they can use local LLMs with RAG to avoid the leakage of confidential

information. That's how you can make sure that sensitive records will never leave your infrastructure.

RAG also helps to keep the transparency and explainability of the answers since we can always investigate what sources were used to give a certain answer.

Lastly Retrieval Augmented generation just generally improves the response quality and relevance, making it a great tool to boost the user satisfaction.

Key Principles of RAG.

The information from the data source gets split into smaller pieces that are called chunks. Those chunks are then stored in a vector database as an index, so more similar chunks are closer to each other in a vector space. Then when the user makes a query, the most similar to the query chunks are getting retrieved basing on the similarity metric. After which the LLM answers the user's query using retrieved information as a context.

Of course, usually the pipeline is not that simple and it can get more complicated and more efficient if we use a Reranker model which basically reorders the chunks, so we can get less chunks of a better quality. Alternatively, we can also do some prompt augmentation to achieve better retrieval quality. There are many techniques and methods that help to improve the Retrieval Augmented Generation System performance which we are going to review and test later in this work.

1.2 Core System Components

In this part we are going to review what are the essential parts of the RAG system and list the most popular options for each one of them.

Vector Index

First, the documents get split into chunks which can be done in different ways. The simplest way to do it is to have even predefined chunk size (for example, 1000 tokens) and a predefined chunk overlap number (for example, 100 tokens). The other popular strategy is to have chunks ends at the points where the sentences end. The most sophisticated strategy is to have a semantic-based chunking which is usually done by an LLM which makes the chunks look more semantically logical which means that they are separated by the topic shift. It is also possible to combing those strategies since they all have their advantages and disadvantages.

The next thing that happens is embedding generation. This happens using an embedding model. This means that a vector space is created where the more similar pieces of information have a smaller distance between each other.

Lastly, the newly created embedding is stored in a vector database. I used FAISS to store and retrieve the most similar chunks to a query because it was created by Facebook specially for such purposes.

Chunk Retrieval

In the heart of chunk retrieval lies the k parameter. It is the number of chunks that gets retrieved. There are a few approaches how this retrieval can happen. I will cover a few of them.

The first approach is to use the k-nearest neighbors' retrieval. The simplest way to do is to just do the exact k-nearest neighbors. When you have millions of chunks that might not be the quickest option. In this case you might want to approximate KNN. It is not as accurate but it is much faster. The most important parameter to tune in this case is k.

The second approach is sparse retrieval. It uses more traditional NLP techniques. The top k chunks are retrieved using TF-IDF or BM25 score. It

is a bit less accurate than dense retrieval, but it captures some of the things that the other approaches do not capture.

The third approach is dense retrieval. This is the approach that I use most of the time in this work (I also use hybrid in one experiment). The main idea of this approach is to build an embedding space using some model. And after this we just retrieve the k nearest vectors to the query vector. This approach is the most accurate one because it takes into account not only the overlap of the tokens but also their semantic similarity. However, it comes with a price: it costs money to build an index, it might work a bit slower. In some languages due to the issues with an embedding model there might be bad performance. This approach is also less interpretable.

The fourth and the last approach that I am going to cover is hybrid search retrieval. Basically, it is just a combination of the sparse and dense search retrievals. I used a variant of this approach where I do sparse retrieval and get m chunks and I do a dense search and get n chunks. After this I use those m+n chunks for retrieval. It is supposed to cover the flaws of both approaches. In reality it performed worse than the normal dense search retrieval.

In some cases, you might want to implement dynamic retrieval to improve the quality of your responses. Normally, the k parameter is fixed and is chosen using hyperparameter tuning. An alternative approach would be to change k depending on a query. The idea is to change k depending on the length of the query, depending on the model that is used for answer generation. Another decisive factor that impacts the k could be the BM25 score. You can also change the k parameter dynamically basing on the feedback from the previous retrievals.

Reranking

The main concept of reranking is to have a model that receives top n chunks from the index. After this it reranks them and outputs top k. Both n and k can be very different depending on the case. They are hyperparameters.

The most common approach to use reranking is to just use a language model as reranker. The most common choice here are Cohere AI models. In this work I used the rerank-3.5 model. It is a very powerful technique, and it improved my performance drastically.

There is also an approach when reranking is used basing on the BM25 score. It is called lexical reranking. In some cases, you might want to implement both cross-encoder and lexical reranking. But as with hybrid search it might make the performance of your system worse.

Query Enchantment and Expansion

Sometimes the query from the user is dry and does not trigger the right areas in the index. This means that the words in the query do not approach the right chunks. This issue can be solved using Query Expansion. The main idea of this technique is to generate enriched queries to improve retrieval. It is done through generating a lot of synonyms for each word. In my case it did not bring any improvements which means that the flaws in my system are elsewhere.

Another powerful technique in this area is Keyword Extraction. It helps to improve the performance by extracting the keywords from the query. We use less tokens which is better, sometimes we might improve the embedding quality, we reduce the amount of noise in the data (some useless stop words) and we actually focus on the important words that carry some semantic meaning. The main downsides of this method are loss of nuance, loss of multi-word phrases. It also requires a well-written stop-word list which is a problem when it comes to Ukrainian.

Lastly, I am going to explain what Contextual Rewriting is. It is a technique that improves the performance of a Retrieval Augmented Generation system in case the question requires some information from the previous questions. The easiest way to implement this method is to create simple rules, like when you see a word from a certain list of words, you should call an LLM and ask to regenerate this question using the text from the last question (or a few last questions) to get the context from them.

Answer Generation

The first option how to enhance the answer generation is to use chain-of-though for better retrieval quality. The approach I used in my work was as follows: after the system is asked a question it creates an answer and a list of gaps in the answer. If there are no gaps, we just get the answer. If there are gaps, we try to generate the questions that would theoretically fill those gaps. After the generation of this questions, they get asked to the RAG system. We receive some answers and use them along to answer the main question. Then the process repeats (if there are still gaps, we continue). We break this loop only when there are no gaps in the answer or if we exceed the limit of iterations that is set as a hyperparameter. I chose 3-4 as the number of iterations in this work.

Another way to enhance the system during answer generation stage is to use prompt templates. First of all, you need to define the role of the Large Language Model, it helps to improve the quality of retrieval a lot. You can also include the example questions with reference answers into the prompt. Usually, it helps but, in this work, it proved to be redundant. If you can define the limits and shape of your ideal answer, you should include it to the prompt too. You should also specify what is the model supposed to do when it does not know what to answer. In some cases, we might want the model to

just confess that the knowledge is not there. But sometimes it is also fine if it generates something and might randomly be not far from the truth.

Lastly, the most obvious way to affect this stage of RAG is to simply use another Large Language Model for retrieval. The choice of the model is a vital step, it is also a hard one. The main challenge is to balance the costs and the answer quality. Sometimes it is very hard to find the right ratio. The most accurate models are cloud based and can be used via API interface. Usually, you pay for all your calls and the amount of money depends on the goodness of the model. On the other hand, it is also possible to install certain LLMs or SLMs locally. The performance will be much worse, but the prices are much nicer since you only must pay for the electricity for the computations. However, those models can be customized in a lot of ways.

The smartest approach here is to combine the usage of LLMs and SLMs. For example, simple decision-making parts could be delegated to an SLM, and more sophisticated work could be done using an LLM. The most common approach here would be to use a less powerful model for first-stage answer. Then we evaluate the answer and if it is good enough, we just send it to the next stage (or to the user). However, if the quality is bad, we use a much more powerful model. This helps to decrease the API costs and improve overall efficiency of the system.

Post-Processing and Validation

After answer generation there is still room for improvement. To enhance your Retrieval Augmented Generation system, you might want to do some work after the you have the final answer.

First, even if RAG is used the possibility of hallucinating is always there. You might want to address the existence of such hallucination using a simple fact-checking algorithm. You just need to split your answer into separate facts and use another LLM to evaluate the correctness of those facts.

Basically, you ask another LLM to answer “Yes” or “No” about a certain fact. If the algorithm identifies a hallucination there are a few ways how it can be treated: we can try to rewrite the answer, try to just delete the problematic part, or decide not to answer at all.

Alternatively, instead of using another Large Language Model we can just go with the RAG system itself and ask it about truthfulness of our facts that we extracted from the query.

The next step in checking the quality of the answer could be to inspect if the main requirements for the answer are met: the language in which the answer must be given, the size of the answer, the special format (for example, JSON).

The next part would be deciding what to do in case the retrieval failed and there are no relevant chunks. The easiest way to treat this is just to write something like “I do not know”, but it is not always acceptable. Another way to address this would be to generate an answer purely using an LLM. The other choice would be to find the most relevant chunks among irrelevant ones. And lastly, it could be helpful to help some kind of message coming out if the system generates results that are not basing on the retrieved chunks, so they know that the answer might be inaccurate.

1.3. Ethical, Bias & Reliability Challenges

There are some no-code related issues that we need to address when developing systems that use RAG. Firstly, we are going to dive a bit into ethical concerns.

A study [8] suggests that you must be prepared for your data from the RAG index to be available to whoever wants it to be available. Using special prompting techniques people can extract information from the index. Even

though there are ways to protect your RAG systems a bit, it is impossible to make sure that nothing will happen. However, the same study says that the data protection in the systems that use RAG is much better than the protection for the data that LLMs were trained on (which is also sometimes sensitive).

There is also a bias issue in RAG. Since such systems usually have a lot of parts like index, or LLM, or a reranker model, all of them introduce more and more bias. This bias might come from the LLM itself, however it might also come from unevenly distributed data in the index.

I have already mentioned it but the fact that we use a RAG system does not mean that we avoided all the possible LLM hallucinations. We still have them, and we still need to address them when needed because in high-stakes environment one model hallucination may cost someone's life.

There was an interesting case when RAG system was producing very questionable outputs because its index is always changing, and it gets scraped from some web source. Some people spammed a lot of malicious messages on that web source and the RAG system started giving weird answers.

The users might also send private information in queries to the RAG system. That means that the user's sensitive data should be safely transmitted and encrypted all the time to avoid sensitive information leakage.

I also wanted to mention that one of the main advantages of RAG is that if there is a change in the documents, we do not have to fine-tune the whole LLM. So, whenever there is a change in the documents it is very important to actually change the documents, because otherwise it is not entirely clear why we would use RAG at all.

Secondly, I would like to review a few problems about the bias in the RAG system, why it might emerge and what to do if it does appear.

There are two main sources of bias and they are: retrieval bias and generation bias. Retrieval bias is caused by the data source where the model gets its data. For example, the index might be too focused on a specific dialect or on a specific area and if ask questions not about the stuff where our index is an expert then we would get a low-quality answer. To avoid this you should check if information in the index is more or less evenly distributed. Also, retrieval bias is caused by the embedding models.

Generation bias is a type of bias that appears on the answer generation side of the pipeline. It is mostly a cause of bias in the training data of the LLM that is used. The way we write our system's prefixes for the questions might also affect them a lot, so we got to be careful with that and test the system with different prompt prefixes to see if there is any bias. To fix most of those issues we just need for someone to check if they exist quite regularly. For example, we need to check some demographic or a wide variety of geographic regions.

One of the possible ways to deal with generation-level bias is to call another simpler model that would be fine-tuned on a specific dataset to use it as bias-detection classifier, so if we receive some weird answers, they will not go through the system to the end user. If a lot of resources are available it is also possible to generate a lot of queries from user's question and modify them, so the answer is given from a different perspective. By that I mean adding some prefixes like (answer as a person who lives in rural India). After getting all the answers, they should be merged, and the final answer should be generated from them.

In this part I am going to inspect and define a few reasons, why Retrieval Augmented Generation could be unreliable. Some of those reasons are not updated data sources, poorly formulated queries that do not retrieve the right chunks even if they are available in the index. Other reasons on the

indexing side could be because we have k chosen as 1, so we only retrieve one document that contains not enough information to answer user's question. Or, alternatively, the chunking strategy was poorly chosen and the most important sentence for the retrieval is separated and contained in two or more chunks and therefore the sense of this sentence is lost in the index.

There are also problems that can arise on the generation side. The most common problem is when a Large Language Model adds some information to the answer that is not backed up by the documents from the data source. Another common problem is when question requires information from different chunks, they could be correctly retrieved but incorrectly merged. Some LLMS also rely on their knowledge more than on the retrieval index which basically ruins the whole purpose of RAG. The answers for the questions can change drastically depending on the prompt formulation, that is why prompt engineering is so important and I am going to show it in this work.

Possible ways how to avoid those reliability issues are: update the sources, add reranking, come up with a sufficient chunking strategy, probably with bigger chunk overlap parameter. To avoid the generation side problems, we can tune the temperature of the model (in this case, the lower the better), generate multiple answers for user's query and then choose the best one using an LLM or let the user decide what the best answer is. Another popular technique in this area to have a verification chain with follow-up question that would help to clarify some information in the answer and, hopefully, improve the answer quality.

1.4 Conclusion for Section 1

In this section we have reviewed the key principles and components of Retrieval Augmented Generation system. The main goal of RAG is to integrate external data sources into retrieval using powerful Large Language Models. The documents from the data sources get chunked and stored in the vector index, making sure that the most relevant ones get retrieved on a user's query. We reviewed the most common search strategies like dense, sparse and hybrid searches. We inspected the reranking possibilities, query expansion technique, chain-of-thought answering method and many more.

However, RAG does have its pitfalls. There are ethical issues and considerations that must be carefully addressed: data exposure, user privacy, bias, and fairness. There are also several reliability issues that can arise: model hallucination, unreliable retrieval, outdated or incomplete retrieval index noisy questions, poorly written prompts and so on. I inspected the possible ways how to mitigate all those issues.

Armed with this theoretical groundwork we are ready to explore RAG for Ukrainian Government Services use case. Now we are aware of the popular design choices and trade-offs.

SECTION 2. RELATED WORK.

2.1 Studies about RAG Using Data Sources in Ukrainian

There are not so many studies dedicated to using Retrieval Augmented Generation for the documents in Ukrainian. Possible reasons for this could be that there is simply less data in Ukrainian which would make it harder to apply RAG for some specific domains, there is also might be not a lot of funding for such types of research since, for example, English speaking scientific industry has much more money to offer to the research. Other reasons might be brain drain, smaller research community, lack of benchmarks. But the main reason, probably the most obvious one, is that this method is quite new. Last year, when I was writing my thesis there were no works about RAG on data sources in Ukrainian [2]. This year there are two more.

First, I will describe the study that is related to mine, but I am not able to compare my results with theirs since we were using Retrieval Augmented Generation for different purposes. The study focuses on using RAG to automatically calculate the sentiment score for some comments in Ukrainian. The finds are that RAG does not improve the calculation of the sentiment score but slightly improve the results for neutral comments [3].

Second, I found a great study connected to the UNLP that used Retrieval Augmented Generation with an interesting twist. They used pseudo-embeddings and the retrieval was done using n-grams. Surprisingly such a simple approach manages to bring a +10% in accuracy. They used some documents from Wikipedia and Ukrainian school program and evaluated the performance of the system using Ukrainian multiple-choice ZNO questions [4].

Third, as I have already mentioned I wrote a similar work a year ago [2] about application of Retrieval Augmented Generation for legal documents in Ukrainian. I have used Ukrainian traffic rules as a case-study and I have tested it on questions from Ukrainian driving tests. I have tested only some basic RAG approaches, like Reranking. The best system was 9% better than non-RAG baseline and it was created using Reranking. There was also a 16% improvement using Separated RAG approach, but it was only possible there because traffic rules can be easily sorted by topic. In this work it would be almost impossible to implement this approach.

Last I wanted to mention a study not about Ukrainian language but about using RAG for non-English languages in general [5]. It uses Retrieval Augmented Generation with a dense retriever, optionally, with a Reranker model (BGE-m3). It checks the improvement in retrieval comparing to a non-RAG approach in 13 languages. The improvements differ a lot depending on a language. For example, in some languages like Thai and Japanese the quality of the answers decreases when using RAG. This makes sense because those languages differ a lot from English in all ways. The languages where the performance improved the most are Finnish (+15.5%) and Russian (+12.9%). The average improvement is 6.3%.

Language	No-Retrieval	User-Lang RAG	Impro
Arabic	26.4	36.3	+9.9
Chinese	21.4	22.5	+1.1
French	48.4	56.3	+7.9
Finnish‡	29.7	45.2	+15.5
German	47.8	54.8	+7.0
Italian	51.5	56.8	+5.3
Japanese	31.7	28.8	−2.9
Korean	21.5	31.5	+10.0
Portuguese	48.4	54.9	+6.5
Russian†	38.1	51.0	+12.9
Spanish	52.5	57.3	+4.8
Thai‡	12.4	10.1	−2.3

Figure 2.1 - Improvement in retrieval accuracy (percentage points) when using User-Language RAG versus no-retrieval baseline, broken down by language.

2.2 Studies about Using RAG for Government Services

There are a few studies that use Retrieval Augmented Generation for something associated with government services.

Firstly, I would like to review a paper about GovRAG by Yu and Chen [6]. The goal there was to build a Framework that would help answer government-related questions better than baseline Large Language Models.

The Framework was tested on different datasets with such questions and managed to achieve a 10-15% boost in performance on average.

Secondly, there is a study that tests different configurations of Retrieval Augmented Generation to enhance the retrieval for questions about National Defense Authorization Act (NDAA) [7]. They do very similar work to what I am doing in this paper, but they have no baseline measurements, so it is impossible to understand the improvement in the results and compare them to this work. However, they mention that the GPT-4 RAG System achieves 77% on faithfulness evaluator and 73% on relevance evaluator.

2.3 Conclusion for Section 2

Despite, there is not a lot of interest in Retrieval Augmented Generation using documents in Ukrainian I have explored related studies. One of them was using RAG to enhance the sentiment analysis on comments in Ukrainian. Another study used Ukrainian school data to improve the answer quality for ZNO exams and achieved about 10% improvement. I also did a little review on my study from last year where I managed to achieve 16% improvement on Ukrainian traffic law questions. The study that was investigating the improvements that RAG brings to non-English documents showed an average 6% improvement.

Turning to government-service domains, RAG frameworks like GovRAG demonstrates a 10-15% boost over pure Large Language Models and the NDAA study shows 73-77% rates on faithfulness and relevance.

SECTION 3. METHODOLOGY.

3.1 Frameworks and Data Sources Overview

The most popular frameworks for Retrieval Augmented Generation are LlamaIndex and LangChain. There are also RAG tools introduced in the HuggingFace Transformers library. There is even a special library from Google called AI Edge which is suited for Android [8]. You can run RAG systems locally on the phone offline using Google Small Language Models like Gemma.

My previous work on Retrieval Augmented Generation [9] was written using LlamaIndex because there I implemented only a limited range of basic techniques and methods to improve the performance of my RAG system. On the other hand, now I needed to choose a more complex library which would allow me to try new, more advanced RAG techniques. It is an open-source library which was introduced in 2022. It provides a wide variety of tools to develop RAG-based solutions using Python. The only possible drawback of this framework is that there is just so much stuff available that it can be confusing for a person who is not familiar with this area.

I found a great source of data with all the descriptions about Ukrainian Government Services on the Governmental website [10]. There were two download options in JSON and EXCEL formats. The data in those two sources is identical. There is information about 2160 services. Each service has a list of properties. Those properties are Id of the service, thematic area, sector, name, level, keyword, short description, spatial, moderation status, creation date, owner of the service, service provider, legal base, refusal grounds, events, refusal appeal person, is appealed in court, refusal appeal rules, produces, input documents, application ways, receiving ways,

processing durations and, finally, costs. I decided to use the JSON document because it is much easier to parse it in Python than the EXCEL document. After some tests I dropped some redundant features that do not really impact the retrieval quality. A good example of such feature is the service id. It does not provide any meaningful information to the Large Language Model.

3.2 Creation of the Evaluation Dataset.

To understand how good the quality of the responses of out Retrieval Augmented Generation System are, we need to have a set of questions and answers that would test the knowledge of our system.

Although in all the existing works that deal with documents in Ukrainian use multiple choice questions for evaluation, I decided that it would not give an accurate understanding of how good our system is. So, I decided to make a dataset which would consist of open questions with suggested answer for each question. Creating such a dataset manually would take way too much time, so I decided to use GPT-o4-mini-high. I fed the text from the data source into the model piece by piece. Each time I submitted information about one service and asked the model to generate about 3 good quality questions that would check the knowledge about that service. A chose random services for those operations. Using these methods, I created a dataset with 500 questions which is available on my GitHub page for reuse [11]. All the questions in the dataset were manually checked. The main concern that I have about this dataset is that I am not sure if it fair that we evaluate the system that uses LLMs using questions that are generated by another LLM. The good thing is that we at least use different LLMs so technically I would not consider this being data leakage of any kind.

A significant issue that I faced with during the creation of the dataset is that the model was forgetting the prompt details quite fast, and it was creating questions that cannot be answered without the context. I was deleting such questions and asking the model to always mention the context.

The main property of those questions is that they are quite straightforward and do not require some complicated logic to answer them. They basically just check if the system is aware of the facts from the data source. This is quite different from the questions that I used in the traffic rules work [12]. Those questions required some thought process. The questions in this dataset are usually quite similar to each other. The most common categories of the questions are to ask about the cost of a certain service or to ask about the list of the documents that you need to apply for a particular service or a duration of doing a service.

3.3 Evaluation Approach and Evaluation Metrics

To evaluate the performance of a certain configuration of the system I iteratively asked the questions to the system. The questions are asked separately, and the system has no information about the previous questions. After all the questions were asked, I computed the average evaluation metrics for all the answers. My code in Python lets me choose the number of questions that would like to test on. Additionally, I can choose the random seed to secure that the test sample is the same across all tests.

I chose the Ragas framework [13] for the evaluation. I used all the evaluation metrics that would make sense for my project purposes. I will through every single one of them iteratively.

Answer Relevancy

Judging by the name of the metric it is not hard to guess what it means. It calculates the level of relevancy of the answer against the user's query.

Semantic Similarity

The values of this metric lie between 0 and 1. This metric measures the semantic resemblance between the answer to the question and a reference answer that is already written in the dataset. A bi-encoder model is used to calculate the semantic similarity score in this case. This works in such a manner: we get the system's answer to a certain question and vectorize it, we also vectorize the right reference answer. Then we calculate cosine similarity between those vectors and get the semantic similarity score [14].

Answer Correctness

Like all the other metrics, this one also is in the range from 0 to 1. It is basically a score that is computed as a weighted combination of factuality and semantic similarity. The default weights are 0.75 and 0.25 respectively. Theoretically, the weights can be adjusted but I used the default ones [15].

Factual Correctness

This score is again generated against the answer to a user's query and a reference answer. It varies from 0 to 1. The key principle of how this metric works is that we divide the answer and the reference on 1 or more claims or facts. Then we check if those facts match and basing on that we get the factual correctness score. There are a few parameters that I did not change during the evaluation process, but it is important to mention them. Those parameters are atomicity and coverage. Atomicity basically means the degree to how many claims we want to divide our answer. Coverage is a parameter that is a degree of specificity in the claims. Low coverage means that the facts and claims are more generalized [16].

Non LLM String Similarity

This is a metric that does not uses LLMs to calculate the distance between an answer and a reference. Instead, it uses a traditional string distance methods like Levenshtein or Hamming. It is probably not as accurate as the LLM based metrics because it does not capture a lot of semantic connections between the words or sentences. However, it is very fast, cheap and easy to compute since we do not make any LLM calls. It will not be the most important metric, but it is a good idea to use it nevertheless [17].

BLEU Score

BLEAU Score or Bilingual Evaluation Understudy is another metric that I used. Basically, it measures the overlap in n-grams (usually $n<4$) between a machine answer and a human answer. In our case it is of course a machine answer and a reference answer also created by a machine. This metric is commonly used to calculate the accuracy of a translation, but it is applicable in Retrieval Augmented Generation too. This metric also does not use the help from Large Language Models, so it is not always perfectly accurate, because it might not capture some of the deep semantics, but it should add some depth and completeness to my experiments and analysis. The greatest advantage of this metric is that it is fast and easy to compute [18].

ROUGE Score

The ROUGE Score or Recall-Oriented Understudy for Gisting Evaluation Score is another Non-LLM metric that I used in this work. It is somewhat similar to the BLEU score since it also works with n-grams but in this case it calculates the n-gram precision, recall and f1 score which makes it a bit more robust. This is also not the most accurate metric, so we are going to take this score with a little grain of salt. This score relies too much

on the exact match of some tokens, so we might miss some semantic overlaps [19].

Nv Accuracy (LLM Judge Evaluation)

This is probably the most accurate metric among the eight metrics that I chose. I will refer to it in this work as the LLM Judge Score or LLM Judge Evaluation Score. The metric just uses a Large Language Model as a judge to evaluate how good the response is against a reference answer. The task for the model is to classify the response into 3 bins: inaccurate answer, partial answer, correct answer. This means that the answer gets one of 3 scores: 0, 2, 4 respectively. After this the answer is converted into a score that ranges between 0 and 1. I used the same Large Language Model as a Judge as for Retrieval Augmented Generation. So, in most cases it was GPT-4.1-mini [20].

The most important metrics

I use 8 metrics which is a lot and some of them are not as accurate and reliable as the others. To determine which ones are the most important I just ran a test on 100 questions and manually checked if the scores from each metric made sense. After doing this I determined that the most accurate metric by far is the LLM Judge Score (or nv_accuracy in Ragas) and Factual Correctness. To prove my point, I am going to demonstrate some of the questions from this testing and comment on the scores from different metrics.

```
Q: Чи потрібне доручення, щоб отримати дозвіл на імпорт,
якщо подаю документи через представника?

LLM answer: Так, для отримання дозволу на імпорт через
представника потрібне доручення,яке підтверджує повноваження
представника діяти від вашого імені.

Reference: Так. Щоб представник міг забрати готовий дозвіл,
необхідно нотаріально оформити доручення на його отримання.

Scores:
  answer_relevancy: 0.957
  semantic_similarity: 0.906
  answer_correctness: 0.826
  factual_correctness(mode=f1): 0.400
  non_llm_string_similarity: 0.255
  bleu_score: 0.028
  rouge_score(mode=fmeasure): 0.000
  nv_accuracy: 0.750
```

Figure 3.1 – An example test case from the dataset

From Figure 3.1 we can conclude that the answer is quite good but not 100% correct. The LLM Judge Score (nv_accuracy) perfectly reflects my thoughts about the quality of this answer. On the other hand, BLEU and ROUGE scores perform poorly as well as Non-LLM String Similarity metric.

```
Q: Чи стягується оплата за державну реєстрацію смерті?

LLM answer: Так, за державну реєстрацію смерті в Україні стягується плата.
Розмір оплати може варіюватися в залежності від регіону та конкретних умов.

Reference: Державна реєстрація смерті надається безоплатно.
Scores:
  answer_relevancy: 0.000
  semantic_similarity: 0.910
  answer_correctness: 0.228
  factual_correctness(mode=f1): 0.000
  non_llm_string_similarity: 0.290
  bleu_score: 0.042
  rouge_score(mode=fmeasure): 0.000
  nv_accuracy: 0.000
```

Figure 3.2 – An example test case from the dataset

The test case shown in Figure 3.2 is a perfect example of a question that was answered incorrectly. However, the semantic similarity metrics seems to show that the answers are close to each other when they are the exact opposites. Here we can also see the flaws in the work of Answer Correctness and Non-LLM String Similarity metrics. The nv_accuracy score and the Factual Correction metric perfectly evaluate this test case.

```
Q: До якої установи потрібно подавати заяву та
комплект документів для отримання спеціального дозволу
на забудову площ залягання корисних копалин?

LLM answer: Заяву та комплект документів для отримання
спеціального дозволу на забудову площ залягання корисних
копалин потрібно подавати до Державної служби геології та надр України.

Reference: До відповідної Ради народних депутатів за
територією розташування корисних копалин.

Scores:
  answer_relevancy: 0.983
  semantic_similarity: 0.885
  answer_correctness: 0.521
  factual_correctness(mode=f1): 0.000
  non_llm_string_similarity: 0.216
  bleu_score: 0.035
  rouge_score(mode=fmeasure): 0.000
  nv_accuracy: 0.000
```

Figure 3.3 – An example test case from the dataset

In Figure 3.3 we can see that the system gives an answer that is 100% incorrect which is reflected in the LLM Judge Score and in the Factual Correctness Score. However, Answer Correctness once again proves to be unreliable and Answer Relevancy with Semantic Similarity also do not do a great job in this test case.

```
Q: Скільки часу відведено на розгляд заяви та видачу
дубліката контрольної картки після її подання?

LLM answer: Зазвичай, на розгляд заяви та видачу дубліката
контрольної картки в Україні відводиться до 30 днів.

Reference: Розгляд заяви та видача дубліката проводяться
не пізніше ніж у 30 календарних днів з дня подання
повного пакета документів.

Scores:
  answer_relevancy: 0.881
  semantic_similarity: 0.922
  answer_correctness: 0.980
  factual_correctness(mode=f1): 0.500
  non_llm_string_similarity: 0.333
  bleu_score: 0.050
  rouge_score(mode=fmeasure): 1.000
  nv_accuracy: 0.500
```

Figure 3.4 – An example test case from the dataset

On a Figure 3.4 you can see a rare case when I do not agree with the LLM Judge Score, I think the system answer is almost identical to the reference answer, so I would except at least a 0.75 score but it is only 0.5. Factual correctness is also 0.5 However, in this case Answer Correctness and ROUGE Score perform really well with the results close to 1.

```
Q: Підприємство планує розширити перелік послуг під уже
зареєстрованою маркою — чи треба знову подавати нову заяву?

LLM answer: Так, підприємству потрібно подати нову заяву
про розширення провадження виду господарської діяльності
до відповідного органу, якщо воно планує розширити перелік
послуг під уже зареєстрованою маркою.

Reference: Так. Якщо ви додаєте нові класи товарів чи
послуг, необхідно подати окрему заявку або заяву про розширення,
оскільки кожен клас підлягає окремій експертизі та сплаті збору.

Scores:
  answer_relevancy: 0.860
  semantic_similarity: 0.899
  answer_correctness: 0.475
  factual_correctness(mode=f1): 0.440
  non_llm_string_similarity: 0.242
  bleu_score: 0.065
  rouge_score(mode=fmeasure): 0.000
  nv_accuracy: 0.500
```

Figure 3.5 – An example test case from the dataset

The test case from Figure 3.5 is like the previous one. The answer is almost correct but the Factual Correctness and LLM Judge scores are around 0.45-0.5, which is way too low. It is quite interesting to notice that none of the features perform well in this test case apart from the Answer Relevancy and Semantic Similarity. But, for example, Semantic Similarity is always somewhere in those numbers, so we should not pay too much attention to it.

```
Q: Який розмір адміністративного збору за зняття з реєстрації місця проживання?

LLM answer: Розмір адміністративного збору за зняття з реєстрації місця
проживання в Україні становить 0,1 розміру мінімальної заробітної плати.

Reference: Адміністративний збір становить 1,5 % прожиткового мінімуму
для працездатних осіб (45,42 UAH) і сплачується через
портал Дія під час формування заяви.

Scores:
  answer_relevancy: 0.883
  semantic_similarity: 0.914
  answer_correctness: 0.228
  factual_correctness(mode=f1): 0.000
  non_llm_string_similarity: 0.280
  bleu_score: 0.018
  rouge_score(mode=fmeasure): 0.286
  nv_accuracy: 0.000
```

Figure 3.6 – An example test case from the dataset

The test case in Figure 3.6 shows that when a RAG system gives and answer that is 100% incorrect some of the features do not perform quite well. Mainly, we see that Answer Relevancy and Semantic Similarity did not really change from a previous test case, where the answer was almost correct. ROUGE score does not show good results here too. Answer Correctness and Non-LLM Semantic Similarity scores are also inaccurate. Our chosen, the most reliable metrics show a good result here which is 0%.

```
Q: Хто може подати заяву про взяття нерухомого майна на облік як безхазяйного?

LLM answer: Заяву про взяття нерухомого майна на облік як безхазяйного можуть подати особи
які мають законний інтерес до цього майна, зокрема, органи місцевого самоврядування,
державні органи, сусіди або інші зацікавлені особи.

Reference: Заяву подає орган місцевого самоврядування тієї територіальної громади,
на якій розташоване нерухоме майно.

Scores:
  answer_relevancy: 0.837
  semantic_similarity: 0.904
  answer_correctness: 0.393
  factual_correctness(mode=f1): 0.250
  non_llm_string_similarity: 0.245
  bleu_score: 0.027
  rouge_score(mode=fmeasure): 0.000
  nv_accuracy: 0.500
```

Figure 3.7 – An example test case from the dataset

In the Figure 3.7 we can see a very interesting test case, when the answer is partially correct but contains some inaccurate information. We can see the BLUE and ROUGE scores do not capture that. Answer Relevancy ad Semantic Similarity show poor performance once again. Only Answer Correctness, Factual Correctness and LLM Judge Score show good results. I would say that the most accurate score is achieved by Factual Correctness.

3.4 Conclusion for Section 3

In this section I laid out the main choices that I have made when designing this work. Firstly, I explained the choice of using LangChain as my main framework for everything RAG-related. Secondly, I described the contents of the data source about Ukrainian government services that I found on a government website. I also listed some of the features from the services and mentioned that there are more than 2000 services in this dataset. This dataset is stored in JSON format. I also described an approach of creation of

my evaluation dataset. I generated and manually checked 500 questions and reference answers about this domain using GPT-o4-mini-high model. I stored this dataset in JSON format which is very convenient if you use Python.

Thirdly, I went through the list of evaluation metrics that I used and described them and outlined their advantages and disadvantages and some of the most important parameters.

Lastly, I did a very thorough analysis of performance of my evaluation metrics. The goal of this analysis was to determine which metrics are more reliable than the others and which ones are completely unreliable. The most reliable metrics were LLM Judge Score and Factual Correctness. I will use only them to describe my results from now on. The non-LLM metrics proved to be completely unreliable for my case.

SECTION 4. EXPERIMENTS.

4.1 Baseline Configurations

Before diving in our RAG experiments, it is important to mention what baseline are we comparing our results to and what are the reasons why we chose this baseline configuration. In this work I solely rely on the GPT models**.** The final choice was to use GPT-4.1-mini for all the experiments. To come to this conclusion, I needed to test some other models without RAG. The key criteria's for choosing the right model were affordability and quality of the answers. I evaluated my non-RAG systems using the same approach comparing to the normal RAG evaluation.

The main reason for choosing GPT models is that they are leading benchmark leaderboards and they are easily integrated into my Python code via LangChain framework. The other reasons are that this LLM is not so bad at Ukrainian language comparing to the other models.

I have tested a few of the newest GPT models like GPT-4.1, GPT-4.1-mini, GPT-4.1-nano and GPT-4o with GPT-4o-mini. I needed to fit the budget and GPT-4.1 and GPT-4o did not meet those needs. Moreover, the performance of GPT-4o was not amazing (38% LLM Judge Score) and I managed to surpass it using RAG on GPT-4.1-mini. The performance of GPT-4.1-nano was very bad (10% worse LLM Judge Score comparing to GPT-4.1-mini), so I decided not to choose it for the experiments. The performances of GPT-4o-mini and GPT-4.1-mini were comparable (the difference was 2% on LLM Judge Score), but GPT-4.1-mini was slightly better, so I decided to go with it as my main model because the prices are affordable too.

Another important choice, that I needed to make was to choose the right temperature for the model. Basically, it is a degree of creativity and randomness of the output of the model. For the low temperatures (below 0.2) the model just chooses the highest probability token and goes with it. If the temperature is higher, it means that we randomly sample from our probability distribution and the results we get are quite unpredictable.

In this work I decided to go with temperature=0.0 because we need the same outputs from test to test to achieve reliable results. Moreover, the domain that I chose (Ukrainian Government Services) requires the model to be as little creative as possible since we do not want random factors here.

In some of the RAG works the performance of the GPT models is compared to some local LLMs like Llama but I decided not to do this experiment since in all the cases that I saw the Llama performance was worse than the GPT one.

4.2 Naïve RAG Approach

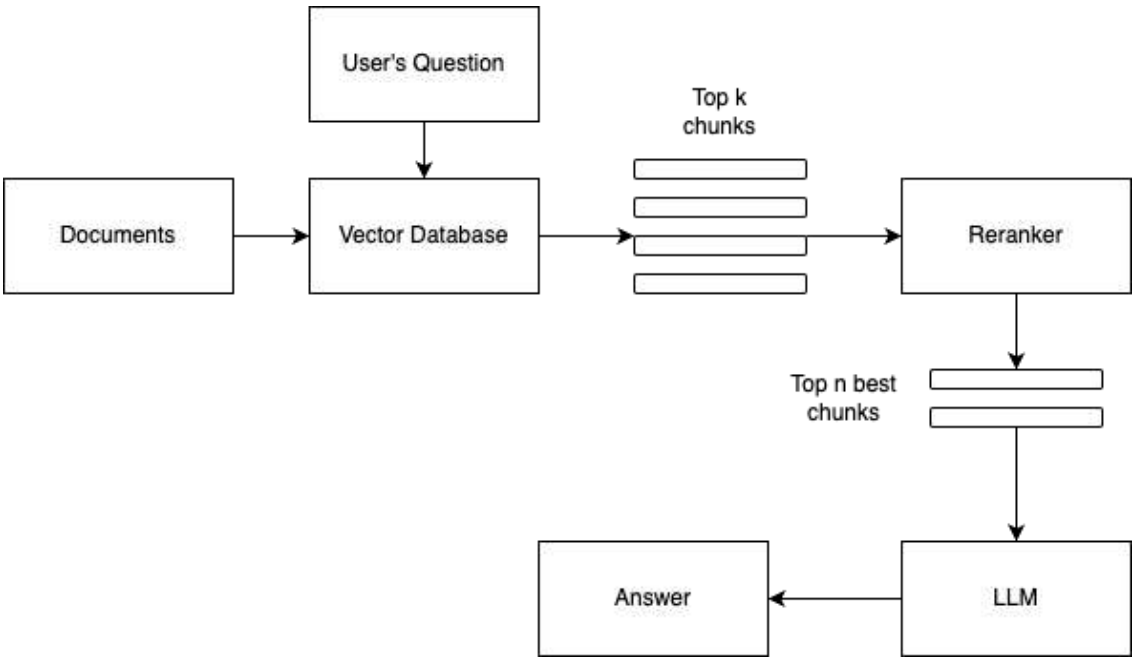

Figure 4.1 – Structure of Naïve RAG System

First, I tested a basic RAG system with no reranking, just pure dense retrieval. I did hyperparameter testing on three parameters: top k retrieved chunks, chunk size and chunk overlap. The main goal of this part of the research was to determine how basic hyperparameters in RAG impact the quality of the retrieval in this case study.

A small chunk size makes the model focus on the particular parts of the text, but we risk splitting an important part of the text in two parts. On the other hand, long chunks contain more information but with that information comes noise might distract the model from the most important facts in the text.

A small chunk overlap means that we do not have a very big index because there is not a lot of repeated information there. However, that could mean that some critical information might get split in half between the chunks. A big chunk overlap brings the risk of having a big index with a lot of repeating information.

The choice of k is also tough since having too low k might lead to relevant chunks not getting retrieved at all, but it also reduces the computational resources that it takes to produce one RAG answer. On the other hand, if we choose a k that is too high, we might face the context overload issue when has too many chunks retrieved, and it does not know where the most important information is which impacts the retrieval quality negatively.

4.3 Basic RAG Enchantment Techniques

Reranking

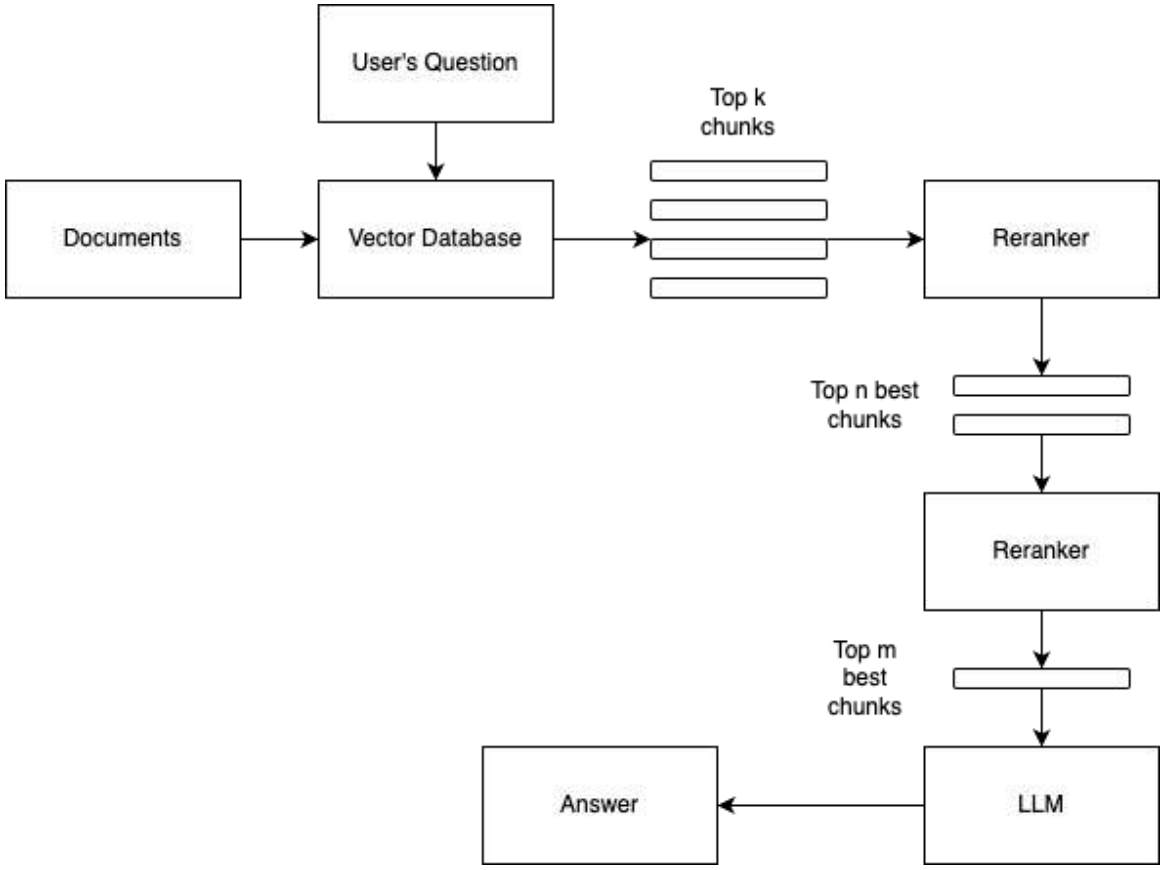

Figure 4.2 – Structure of RAG with Reranking

This is a very basic and common technique. It is simple – we get top n best chunks from the index (usually using dense retrieval), then we pass those chunks onto the reranker model. It reranks them and outputs the best ones. Those chunks then are used as a context to help the LLM generate a relevant result for user's query [21].

In my experiments I used only one model that proved to be very good. It is a Cohere AI Reranker-v3.5. When it comes to hyperparameters, I decided to experiment with both the number of chunks that reranker model gets from the index and the number of chunks that it outputs. The hyperparameter pairs in the testing were (16, 1), (24,1), (12, 1), (8, 1), (4, 1), (8, 2), (8, 4).

Query Expansion

As I have mentioned before Query Expansion is a powerful technique that makes sure that the relevant chunks will get definitely retrieved. The main idea is to generate synonyms along with the words in the question to enrich the query. The expanded version of the query is expected to retrieve more relevant chunks from the index [22].

Iterative RAG

This is method to improve the performance of the RAG system when we input the user's query into the system, then we generate an answer and a list of gaps that need to be filled. After this we ask the RAG system to fill the gaps, we retrieve some chunks that will help us to do so. Finally, we use both the initial retrieved chunks and the "gap chunks" to get the final answer. In case there were no gaps right from the beginning, we just use normal RAG system [23].

4.4 Chunking and Indexing Techniques

Hierarchical Indexing

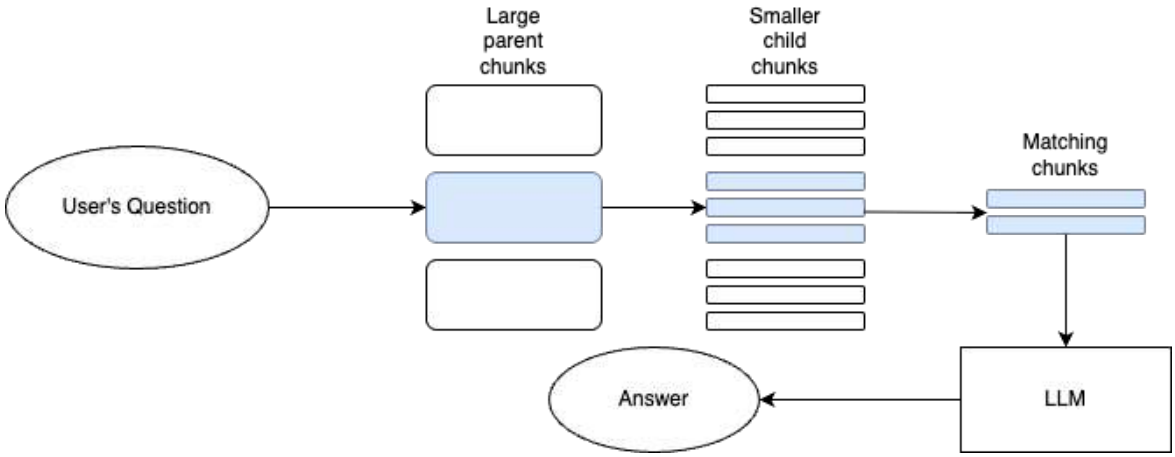

Figure 4.3 – Structure of Hierarchical Indexing

This method introduces multi-staged chunk retrieval since the chunks are organized in a hierarchical manner. The index consists of layers of chunks. First layer consists of gigantic chunks, then the next layer consists of the smaller chunks which are the parts of a bigger parent chunk. This means that when we get a query, we retrieve the chunk from the first layer, then we choose the most relevant chunk among the children of this chunk and so on. It should make our system more efficient, however that comes with a cost of complex retrieval system that is very hard to maintain [24]. In my experiments I used only two-staged retrieval with chunk sizes of 650 tokens and overlap of 200 tokens on the first stage. The second layer is created with chunks of size 200 tokens and overlap 50 tokens.

Hybrid Search

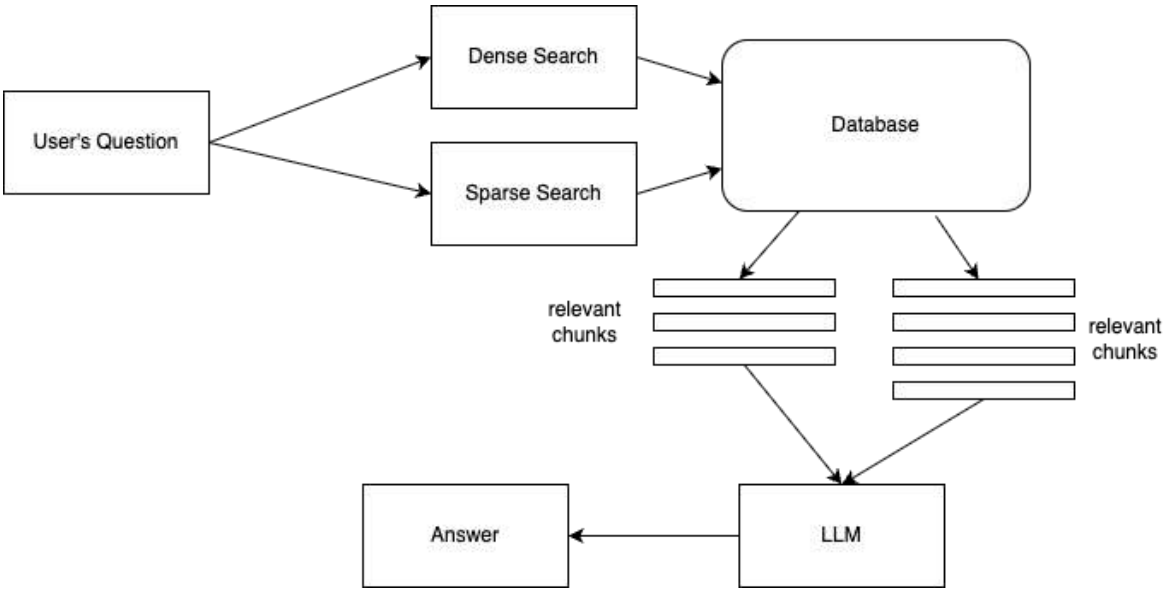

Figure 4.4 – Structure of Hybrid Search

As I have mentioned in the first section of this work, hybrid search means that we do both sparse and dense searches and combine the chunks that were retrieved from both sources [25]. I used different configurations during the testing process. I tuned the number of chunks that got retrieved

from sparse and dense retrievers. I used such combinations during my testing: (8, 8), (4, 4), (6, 2), (2, 6).

Hypothetical Document Embedding

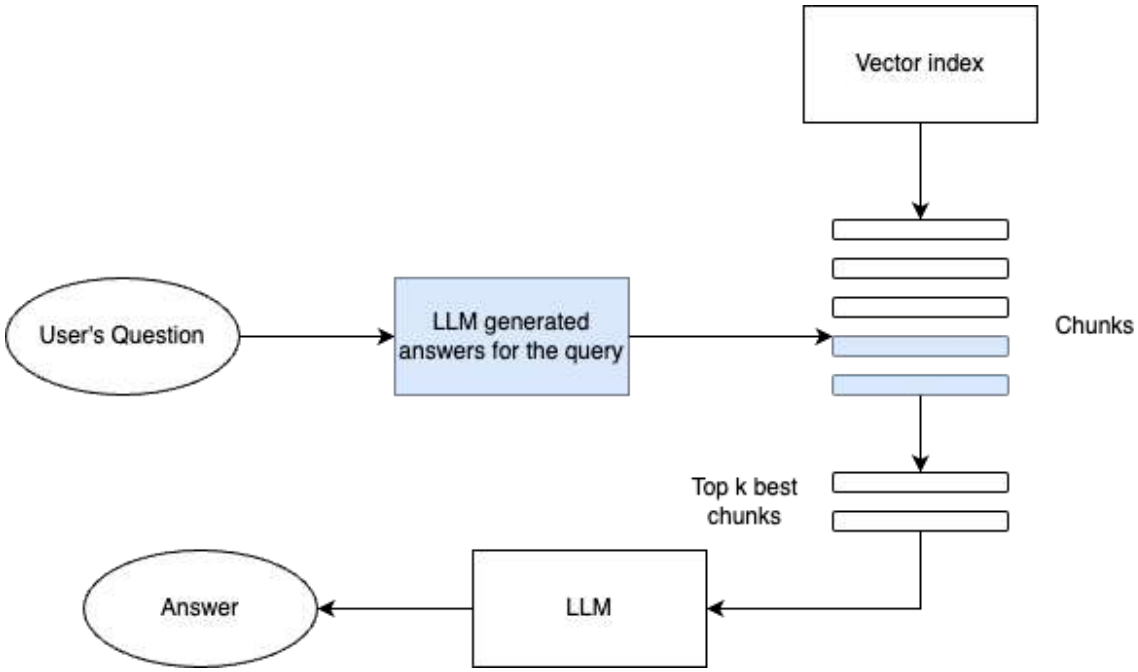

Figure 4.5 – Structure of Hypothetical Document Embedding

The idea in this approach to use both real and hypothetical document embeddings [26]. We do a normal retrieval of chunks from our vector index. After this we ask the LLM to generate some more “fake” documents that would help us answer this question and append them into our hypothetical vector index. After which we retrieve some more chunks from hypothetical vector index and combine them with the chunks we have retrieved before.

In my experiments I tuned the number of chunks that get retrieved from hypothetical and real indices. I used such pairs (first number is a number of real chunks and the second is a number of hypothetical ones): (4, 4), (1, 7), (3, 5), (5, 3), (6, 2), (7, 1).

Hypothetical Prompt Embedding

The concept here is to rewrite the user's prompt and receive some other chunks for a new hypothetical prompt. Let's call them hypothetical chunks. Then we combine them with real chunks that we have received from regular RAG. After which we use chunks from both sources to answer user's query [21].

Similarly, to HyDE approach I chose to tune the and k_real and k_hypo parameters. I tested such pairs: (16, 8), (7, 1), (5, 3), (4, 4), (3,5), (1, 7).

4.5 Prompt and Context Engineering

Prompt Engineering is an extremely important part of RAG system development. In this work I used different prompting techniques to improve the retrieval quality.

```
SYSTEM_PREFIX = (
    "You are an expert in Ukrainian governmental services. "
    "If there is no relevant info in the retrieved docs, answer "
    "concisely from your knowledge without mentioning the source gap.\n\n"
)
```

Figure 4.6 – Simple Prompt Example

First, I just used a simple prompt that simply states the task for the model. You can see it in Figure 4.6.

```
SYSTEM_PREFIX = (
    "You are an expert in Ukrainian governmental services. "
    "If there is no relevant info in the retrieved docs, answer "
    "concisely from your knowledge without mentioning the source gap.\n\n"
    "EXAMPLE QUESTION AND ANSWER:"
    "Question: Хто може подавати заяву на отримання сертифіката лікарського засобу для міжнародної торгівлі?"
    "Answer: Юридична особа (експортер), яка має право надання ліцензії на виробництво або
    реєстраційне посвідчення на лікарський засіб."
    "If there is no relevant info in the retrieved docs, answer "
    "concisely from your knowledge without mentioning the source gap.\n\n"
)
```

Figure 4.7 – A Prompt with an Example Question

Second, I tried to improve it by inserting some sample questions and answers. You can see this prompt in Figure 4.7. I tried different numbers of example questions: 1 and 3.

```
SYSTEM_PREFIX = (
    "You are an expert in Ukrainian governmental services.\n"
    "You are given a question and asked to provide an accurate and concise answer\n"
    "The answer should consist of 1-2 sentences.\n"

    "-----------------------------------------------------------------\n"
    "-----------------------------------------------------------------\n"

    "If there is no relevant info in the retrieved docs, answer \n"
    "concisely from your knowledge without mentioning the source gap.\n\n"

    "-----------------------------------------------------------------\n"
    "-----------------------------------------------------------------\n"

    "This is the question you need to answer: \n\n"
)
```

Figure 4.8 – A Structured Prompt

Third, I tried to structure those prompts better using separator lines. The text of the query is shown in Figure 4.8. I also tried using a structured prompt with examples.

```
SYSTEM_PREFIX = """Ви — експерт із українських державних послуг.
Якщо у витягнутих документах немає релевантної інформації, відповідайте лаконічно
зі своїх знань, не згадуючи про відсутність джерела.\n\n"""
```

Figure 4.9 – A Prompt in Ukrainian

Just out of curiosity I decided to rewrite the prompt in Ukrainian to see the results. You can see the prompt text in the Figure 4.9.

Lastly, I decided to use the Prompt Perfect [27] resource to enhance the quality of my existing prompts and to create a new one. You can see some of those prompts in the Figures 4.10, 4.11 and 4.12.

```
SYSTEM_PREFIX = (
    "You are an expert in Ukrainian governmental services. "
    "Your task is to answer the following questions based on the provided documents. "
    "If the retrieved documents do not contain relevant information, "
    "use your own knowledge to answer concisely without mentioning the lack of source information.\n\n"
)
```

Figure 4.10 – Enhanced Simple Prompt

```
SYSTEM_PREFIX = (
    "You are an expert in Ukrainian governmental services. "
    "Your task is to answer questions accurately and concisely with a 1-2 sentence
    response based on the relevant information provided in the documents.\n\n"
    "-----------------------------------------------------------------\n"
    "-----------------------------------------------------------------\n"
    "If no relevant information is found in the retrieved documents, "
    "answer concisely based on your knowledge without mentioning the lack of source information.\n\n"
    "-----------------------------------------------------------------\n"
    "-----------------------------------------------------------------\n"
    "Here is the question for you to answer: \n\n"
    "Q: Які саме документи потрібно додати до заяви про держреєстрацію створення
    відокремленого підрозділу юридичної особи?\n\n"
    "To answer: "
    "До заяви необхідно додати: 1) паспорт громадянина України (або інший документ, що посвідчує особу); "
    "2) примірник оригіналу (нотаріально засвідчена копія) рішення уповноваженого органу управління
     юридичної особи про створення підрозділу; "
    "3) у разі подання через представника – нотаріально засвідчену довіреність."
)
```

Figure 4.11 – Enhanced Prompt with Example Questions

```
SYSTEM_PREFIX = (
"You are a specialist in Ukrainian governmental services tasked with answering specific
questions accurately and concisely in 1-2 sentences.\n\n"

"Guidelines:\n"
"1. Analyze the given question carefully to understand what the user needs.\n"
"2. Refer to the retrieved documents for relevant information.\n"
"3. If the documents lack information relevant to the question, rely on your
own knowledge to provide the best possible answer without mentioning that the
information wasn't found in the documents.\n\n"

"Examples of typical questions and expected answers:\n"
"-----------------------------------------------------------------\n"
"Question: Хто може подавати заяву на отримання сертифіката лікарського засобу
для міжнародної торгівлі?\n"
"Expected Answer: Юридична особа (експортер), яка має право надання ліцензії на
виробництво або реєстраційне посвідчення на лікарський засіб.\n"
"-----------------------------------------------------------------\n"
"Question: Які основні документи необхідно додати до заяви?\n"
"Expected Answer: Заява за формою додатка 4 до Порядку, дані щодо безпечності,
довідки про перевірки органами контролю, сертифікат GMP (за наявності),
копії ліцензії та реєстраційного посвідчення, технічне резюме тощо.\n"
"-----------------------------------------------------------------\n"
"Question: Який термін розгляду документів для видачі сертифіката?\n"
"Expected Answer: Експертиза документів – до 15 робочих днів; відбір зразків –
до 10 робочих днів; лабораторний аналіз – до 20 робочих днів; остаточне
оформлення сертифіката – до 10 робочих днів після експертизи або аналізу.\n"
"-----------------------------------------------------------------\n\n"

"Evaluate the question below and respond accurately:\n"
)
```

Figure 4.12 – Prompt Perfect Generated Prompt

4.6 Advanced RAG Architectures

Graph RAG (Pseudo-Graph Approach)

The key idea of GraphRAG [28] is to build a knowledge graph on a basis of a vector index that would represent the relations between chunks in our retrieval index.

In my case I did not have enough computational power to make such a graph because of the large size of the documents that I used for retrieval.

Instead of this I built a pseudo-knowledge graph that connects the chunks with high similarity score. I hope it will help to improve the retrieval.

Two-Staged Reranking

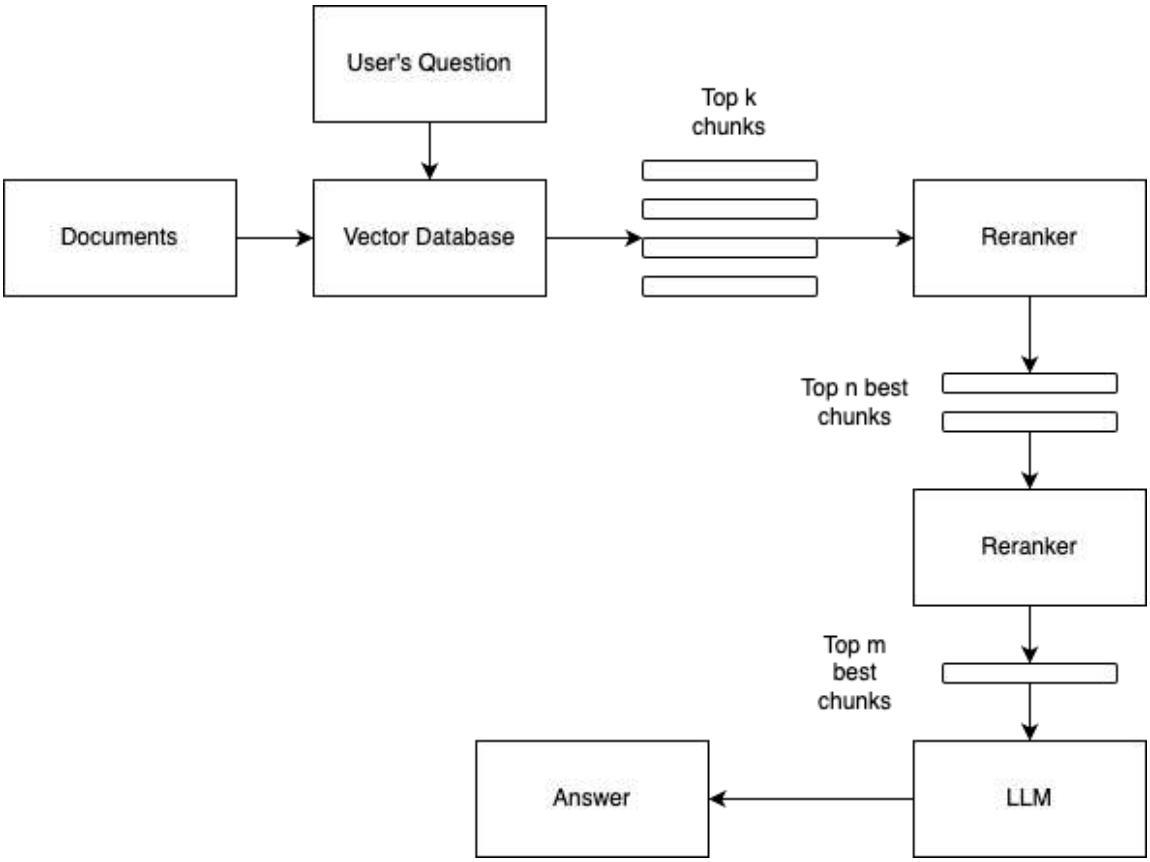

Figure 4.13 – Structure of Two-Staged Reranking

This is an interesting method which basically just uses two rerankers on top of each other. Some chunks get retrieved from the index, after this we filter them through the first reranker and then we filter them through the second reranker. I tried different configurations of the number of chunks on each stage: (50, 12, 1), (20, 8, 1), (24, 8, 4).

Corrective RAG (CRAG)

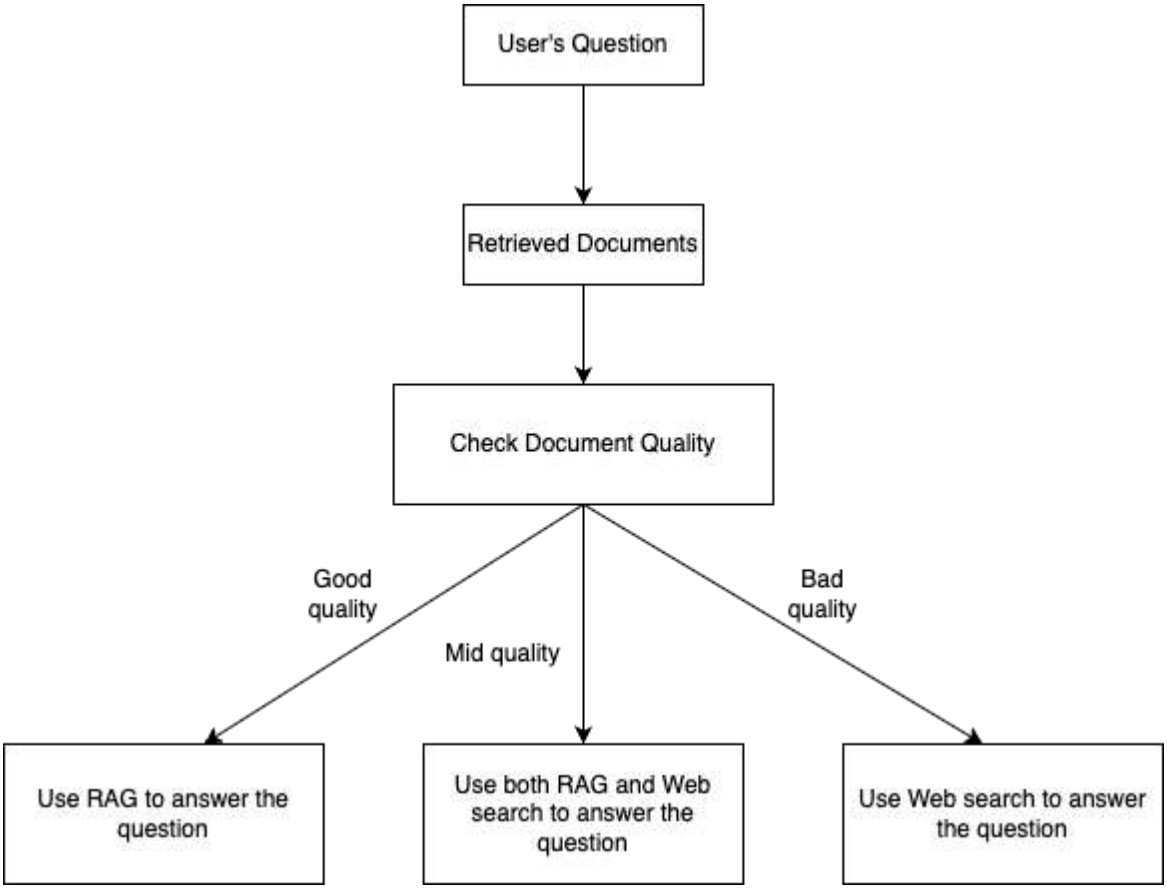

Figure 4.14 – Structure of CRAG

This powerful works like this in my code: if the quality of the answer is above a certain threshold, we just use the regular RAG. If it is below another threshold, we use web search to get any information about the topic and we do not use RAG. If the answer quality is in between the thresholds we use both RAG and web search to answer user's question [29]. In some of the experiments I also combine this approach with Hypothetical Document Embedding technique.

SelfRAG

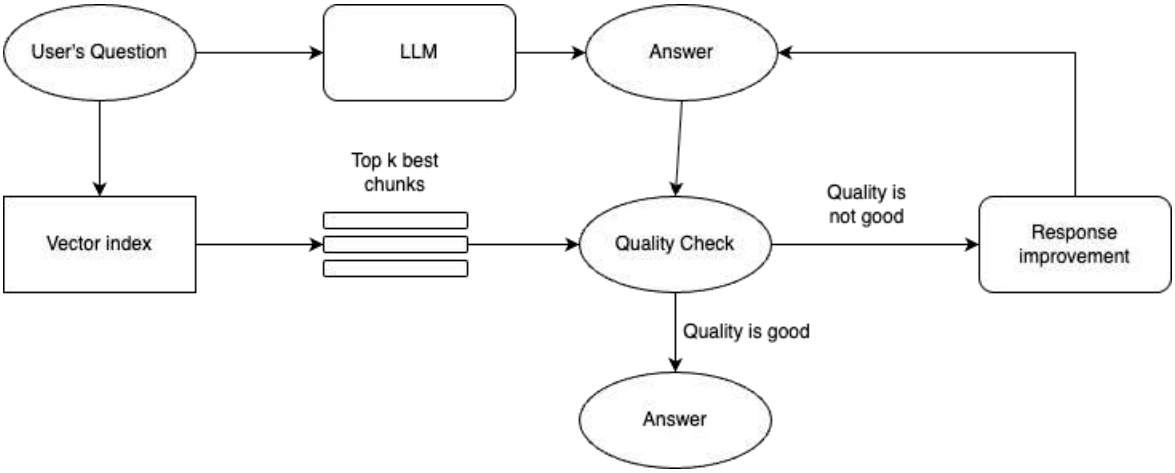

Figure 4.15 – Structure of SelfRAG

The concept of this technique is to implement a chain of making decisions. First, a model decides if it needs the documents from the retrieval index at all. After this we retrieve top 4 chunks, and the model decides if they are relevant or not. For each relevant chunk we generate an answer. Then we evaluate the answers using an LLM Judge and retrieve the best one. Of course, this system makes a lot of calls to the LLM, so it might be costly to use it, but it does improve the performance of the RAG system [30].

```
Question: Чи потрібне доручення, щоб отримати дозвіл на імпорт,
якщо подаю документи через представника?
✓ ref: Так. Щоб представник міг забрати готовий дозвіл, необхідно нотаріально
оформити доручення на його отримання.

Retrieval needed? yes
Retrieved 4 docs
 Doc 1 relevance: irrelevant
 Doc 2 relevance: relevant
 Doc 3 relevance: irrelevant
 Doc 4 relevance: irrelevant
 Candidate 1 → support=fully supported, utility=5
Selected best candidate
→ Final answer: Так, якщо документи подає представник, потрібне доручення.
```

Figure 4.16 – An Example of SelfRAG test case

4.7 Conclusion for Section 4

In this part I chose and established a baseline for this work. I decided to go with GPT-4.1-mini model because it has the best price/quality ratio among the tested models. After this I explored a lot of basic RAG techniques did hyperparameter testing on each one of them, examined different chunking strategies. I also investigated the main prompt enchantment techniques. Finally, I tested some famous advanced RAG architectures like GraphRAG, SelfRAG and Corrective RAG (CRAG).

SECTION 5. RESULTS AND ANALYSIS.

5.1 Baseline LLM and Naïve RAG Performance

LLM Performance

As we can see from the Table 1, the GPT-4.1-mini model is the best at Factual Correctness while GPT-4o is the best at LLM Judge Score. However, the price of GPT-4o makes it impossible for me to do all the tests, so I chose GPT-4.1-mini for further experiments.

Model name	Factual Correctness	LLM Judge Score
GPT-4o-mini	26	28
GPT-4o	26	38
GPT-4.1-nano	24	19
GPT-4.1-mini	27	30

Table 1 – Performance (%) of different GPT models without RAG

Naïve RAG performance

From Table 2 we can see that the best chunk sizes are 250 and 500 and the worst performing chunk size is 100.

Approach name	Factual Correctness	LLM Judge Score
Naïve RAG, chunk size = 600	27	35
Naïve RAG, chunk size = 500	24	36
Naïve RAG, chunk size = 250	26	36
Naïve RAG, chunk size = 100	24	28

Table 2 – Performance (%) of Naïve RAG approaches with different chunk sizes

From Table 3 we generally can see the pattern that the higher our k is, the worse the performance basing on LLM Judge Score. The best k here is 1 and I am going to use it for the whole work.

Approach name	Factual Correctness	LLM Judge Score
Naïve RAG, k=8	26	31
Naïve RAG, k=6	27	33
Naïve RAG, k=4	26	34
Naïve RAG, k=2	25	32
Naïve RAG, k=10	25	30
Naïve RAG, k=1	25	36

Table 3 – Performance (%) of Naïve RAG approaches with different amounts of retrieved chunks

The best configuration of Naïve RAG improved the performance of the system by 6% which is a great result.

5.2 Techniques That Improved Performance

Reranking

I have tried a lot of configurations and different k parameters for input and output of the Reranker model. The best parameters turned out to be 8 chunks for the reranker input and 1 chunk for reranker output. The improvement of the best configuration comparing to the previous best result (Naïve RAG) is 6% which is also amazing.

Approach name	Factual Correctness	LLM Judge Score
Reranking, k = 8 and 2	32	38
Reranking, k = 8 and 1	30	42
Reranking, k = 4 and 1	30	39
Reranking, k = 24 and 1	29	41
Reranking, k = 16 and 1	29	40
Reranking, k = 12 and 1	30	41

Table 4 – Performance (%) of hyperparameter testing for Reranking approach

SelfRAG

Using SelfRAG with Reranker did not make much difference. However, if we just use SelfRAG on Naïve RAG system, we get a 2% improvement comparing to the basic Naïve RAG approach. So, it did not help me to achieve the best result but it proved to be helpful.

CRAG

Experiments with CRAG did bring some good results. CRAG with Hypothetical Document Embedding and Reranking achieved the best result basing on the LLM Judge Score: a 13.25% improvement comparing to a non-RAG baseline. The CRAG with Reranking improved system's performance by 13%. Also, this configuration achieved the highest score in Factual Correctness. The system's performance improved by 10% according to this metric.

Hypothetical Document Embedding

This technique improved the performance of the system by 1% comparing to just using Reranking. The overall improvement comparing to the baseline is 12% basing on LLM Judge Score.

Approach name	Factual Correctness	LLM Judge Score
CRAG with Reranking	37	43
SelfRAG with Reranking	34	39
CRAG with HyDE and Reranking	34	43

Table 5 – The best performing approaches by Factual Correctness (%)

Approach name	Factual Correctness	LLM Judge Score
CRAG with HyDE and Reranking	34	43
CRAG with Reranking	37	43
HyDE_RAG with Reranking	30	43
HyPE RAG with Reranking	30	42

Table 6 – The best performing approaches by LLM Judge Score (%)

5.3 Techniques with Small or Negative Impact

Prompt Engineering

I have received some interesting results here. The best performing prompt is the small default one. Using examples made the performance worse as well as structuring the prompt. Using Prompt Perfect also only made the performance worse. This makes me think that to big prompts affect

the performance of the system and sometimes it is just good to keep the instructions short and concise.

Hybrid Search

This approach did not bring any improvements.

Hierarchical Indexing

This approach made the performance of the system much worse.

Two-Staged Reranking

This was probably the worst idea in the whole work. It is interesting that one Reranking model brought a big improvement in the performance of the system but using two rerankers made the system perform even worse than Naïve RAG. My guess would be that the quality of the second reranker model was just not that good. I used a Pinecone Reranker.

GraphRAG

Since this is not a real GraphRAG approach but a very simplified version of it I did not except it to perform well and it did not bring any improvements.

Iterative RAG

This technique turned out to be very harmful for the system since in the process of iterations the LLM hallucinated a few times and sometimes I got the answers for the questions that I did not answer.

Query Expansion

When I used this technique the performance of the system also got only worse.

5.4 Conclusion for Section 5

As we can see from the Table 7, the best performing configuration for LLM Judge Score is CRAG with Hypothetical Document Embedding and

Reranking (13.25% improvement comparing to the baseline). The best performing system on Factual Correctness is CRAG with Reranking. It managed to achieve a 10% improvement comparing to the baseline.

Naïve RAG improved the system by 6% and Naïve RAG with Reranking was performing even better with 12% improvement. All those numbers are counted using LLM Judge Score.

The techniques that improved the performance of the system were Reranking, CRAG, partially SelfRAG and Hypothetical Document Embedding. All the other techniques did not impact the performance or made it even worse.

Approach name	Factual Correctness	LLM Judge Score
CRAG with HyDE and Reranking	34	43
CRAG with Reranking	37	43
Reranking	30	42
Naïve RAG	25	36
GPT-4.1-mini	27	30

Table 7 – Performance (%) of the most important approaches in this work

SECTION 6. CONCLUSIONS AND FURTHER WORK.

In this work I have inspected the possibility and effectiveness of using Retrieval Augmented Generation to enhance the retrieval quality of questions about Ukrainian Government Services. The main achievements are that I gathered the data from the open data source after which I have generated a list of questions and answers using GPT-o4-mini-high model. Those questions where used to evaluate the performance of my system. This is the first work that uses RAG with Ukrainian documents and evaluates the performance on the open questions. Other researchers can also reuse them since they are available on my GitHub.

When it comes to my results, I have managed to achieve a 13.25% improvement on LLM Judge Score comparing to non-RAG baseline using a CRAG with Hypothetical Document Embedding and Reranking. I have also managed to achieve a 10% improvement in Factual Correctness using CRAG with Reranking.

I have also inspected the state of science in the Ukrainian RAG area and compared my results to theirs. My results were even better since in the work I used to compare my results to the improvement is 10%. I went through the list of relevant metrics in the Ragas Framework and picked out ones that work the best. This is also a valuable finding.

When it comes to the possible approaches how my work could be improved, I think the main ones are: developing a bigger evaluation dataset and make its quality better by involving domain experts, trying new RAG approaches that appear every month, use more powerful models improve the overall system’s quality, finally building the whole GraphRAG and fully implementing this innovative approach. Another good suggestion would be to develop a system where my index automatically updates every day or so.

Overall, I must say that the field of RAG using documents in Ukrainian is very unexplored since I only found 3 existing works on this and one of them is mine. This means that there is a great opportunity to come up with something new and innovative which I encourage everyone to do.

References

[1] “AI Hallucinations in Court Papers Spell Trouble for Lawyers,” Reuters, Feb. 18, 2025. Available: https://www.reuters.com/technology/artificial-intelligence/ai-hallucinations-court-papers-spell-trouble-lawyers-2025-02-18/

[2] A. Marynych, “Application of RAG for Ukrainian legal documents,” Course work, National University of Kyiv-Mohyla Academy, 2023. Available: https://ekmair.ukma.edu.ua/bitstreams/c8eb2428-18a7-4ebb-bbfa-e53def7bfbb5/download#:~:text=using%20Llamaindex%20library,by%20using%20separated%20RAG%20approach

[3] Comparison of Zero-Shot Approach and Retrieval-Augmented Generation for Analyzing the Tone of Comments in the Ukrainian Language. (2023). *ResearchGate*. Retrieved from https://www.researchgate.net/publication/387554681_COMPARISON_OF_ZERO-SHOT_APPROACH_AND_RETRIEVAL-AUGMENTED_GENERATION_FOR_ANALYZING_THE_TONE_OF_COMMENTS_IN_THE_UKRAINIAN_LANGUAGE

[4] Zhadko, O., & Ivanov, V. (2024). *Fine-Tuning and Retrieval-Augmented Generation for Question Answering Using Affordable Large Language Models*. In *Proceedings of the 2024 Workshop on Ukrainian Natural Language Processing* (pp. 73–82). Association for Computational Linguistics. Retrieved from https://aclanthology.org/2024.unlp-1.10.pdf

[5] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert, “Retrieval-augmented generation in multilingual settings,” in Proc. 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, pp. 150–158, 2024. Available: https://ar5iv.labs.arxiv.org/html/2407.01463

[6] C. Yu and F. Chen, “GovRAG: A Retrieval-Augmented Generation Framework for Government Services,” SSRN, 2024. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5111865

[7] M. Smith and L. Jones, “Talk to the NDAA: Enhancing Retrieval for National Defense Authorization Act Queries,” Institute for Defense Analyses, 2024. Available: https://www.ida.org/-/media/feature/publications/t/ta/talk-to-the-ndaa/3001199.ashx

[8] A. Kumar, B. Liu, and S. Patel, “Security and Privacy Challenges in Retrieval-Augmented Generation,” arXiv preprint arXiv:2402.16893, Feb 2024. Available: https://arxiv.org/abs/2402.16893

[9] Google AI, “MediaPipe Generative AI: RAG on Android,” 2023. Available: https://ai.google.dev/edge/mediapipe/solutions/genai/rag/android

[10] Anton Marynych et al., “Ukrainian Government Services Dataset,” NaUKMA EKMAIR Repository, 2023. Available: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/c8eb2428-18a7-4ebb-bbfa-e53def7bfbb5/content

[11] Diia Government, “Official Guide to Diia Services,” 2023. Available: https://guide.diia.gov.ua

[12] A. Marynych, “Thesis Project Questions Dataset,” GitHub, 2024. Available: https://github.com/antoshsha/thesis_project/blob/main/questions.json

[13] B. Raghuvanshi and K. Agrawal, “RAGas: Automated Evaluation Framework for Retrieval-Augmented Generation,” arXiv preprint arXiv:2309.15217, Sep 2023. Available: https://arxiv.org/abs/2309.15217

[14] RAGas Documentation, “Semantic Similarity Metric,” 2024. Available: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/semantic_similarity/

[15] RAGas Documentation, “Answer Correctness Metric,” 2023. Available: https://docs.ragas.io/en/v0.1.21/concepts/metrics/answer_correctness.html

[16] RAGas Documentation, “Factual Correctness Metric,” 2024. Available: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/factual_correctness/

[17] RAGas Documentation, “Non-LLM String Similarity Metric,” 2024. Available: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/traditional/#non-llm-string-similarity

[18] RAGas Documentation, “BLEU Score Metric,” 2024. Available: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/traditional/#bleu-score

[19] RAGas Documentation, “ROUGE Score Metric,” 2024. Available: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/traditional/#rouge-score

[20] RAGas Documentation, “Answer Accuracy (nv_accuracy) Metric,” 2024. Available: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/nvidia_metrics/#answer-accuracy

[21] S. Samia, “What Is Reranking in Retrieval-Augmented Generation (RAG)?,” Medium, Apr 2023. Available: https://medium.com/@sahin.samia/what-is-reranking-in-retrieval-augmented-generation-rag-ee3dd93540ee

[22] Deepset, “Query Expansion in RAG,” Haystack Blog, 2023. Available: https://haystack.deepset.ai/blog/query-expansion

[23] M. Farhadi, “Iterative RAG Explained: Methods and Practical Considerations,” Medium, Jan 2024. Available: https://medium.com/@mehrdad-/iterative-rag-explained-methods-and-practical-considerations-fbf194fae991

[24] K. Sharma, “Hierarchical Indexing in Python: A Guide for Students and Enthusiasts,” Medium, Jun 2023. Available: https://medium.com/@kumud.sharma.0206/hierarchical-indexing-in-python-a-guide-for-students-and-enthusiasts-5852a00b7b33

[25] C. Sakash, “Hybrid Search Is a Method to Optimize RAG Implementation,” Medium, Aug 2023. Available: https://medium.com/@csakash03/hybrid-search-is-a-method-to-optimize-rag-implementation-98d9d0911341

[26] Deepset, “Hypothetical Document Embeddings (HyDE),” Haystack Documentation, 2024. Available: https://docs.haystack.deepset.ai/docs/hypothetical-document-embeddings-hyde

[27] PromptPerfect, “AI Prompt Generator and Optimizer,” 2024. Available: https://promptperfect.com

[28] L. Chen, M. Zhang, and Y. Wang, “GraphRAG: Knowledge Graph-Enhanced Retrieval-Augmented Generation,” arXiv preprint arXiv:2501.00309, Jan 2025. Available: https://arxiv.org/abs/2501.00309

[29] A. Kumar, B. Liu, and N. Rao, “Corrective RAG (CRAG): Adaptive Retrieval Using Hybrid Contexts,” arXiv preprint arXiv:2401.15884, Jan 2024. Available: https://arxiv.org/abs/2401.15884

[30] S. Lee, J. Park, and K. Kim, “SelfRAG: Dynamic Retrieval Decision Chains for Improved RAG,” arXiv preprint arXiv:2310.11511, Oct 2023. Available: https://arxiv.org/abs/2310.11511