Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Application of Retrieval-Augmented Generation for Legal Documents

Текстова частина до курсової роботи за спеціальністю „Комп'ютерні науки" 122

Керівник курсової роботи
старший викладач Курочкін А. В.
_____
(підпис)
"____" _____ 2024 р.

Виконав студент

Маринич А. О.
"____" _____ 2024 р.

Київ 2024

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)
„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Мариничу Антону Олеговичу факультету інформатики 3 курсу
ТЕМА: Application of Retrieval-Augmented Generation for Legal Documents
Вихідні дані:

- Документи українських правил дорожнього руху
- Джерело набору даних для валідації роботи системи

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1 Огляд методу RAG
2 Огляд і аналіз методів і ресурсів
3 Експерименти з RAG
4 Результати експериментів
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2024 р.

Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Тема: Application of Retrieval-Augmented Generation for Legal Documents

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	Жовтень 2023	
2.	Огляд технічної літератури	Січень 2024	
3.	Отримання набору даних	Лютий 2024	
3.	Проведення необхідних експериментів	Март 2024	
4.	Написання текстової частини роботи	Травень 2024	
5.	Захист курсової роботи	Травень 2024	

Студент: **Маринич А. О.**

Керівник: **Курочкін А. В.**

“_____” _____

TABLE OF CONTENTS

ABSTRACT

RAG (Retrieval Augmented Generation) is a new technique that uses the power of large language models to build reliable question answering machines that specialize in the specific domain areas. In this work I focus on using RAG to enhance the knowledge of a question answering system that is supposed to answer questions about specific legal documents. The selected legal documents for this study are Ukrainian traffic rules. I also conduct in-depth analysis of the evaluation approaches of the system's performance. All the code is written using Llamaindex library. I use the driving test multi-choice questions dataset to evaluate the performance of various systems. I have managed to get a 6.5% improvement in accuracy comparing to the baseline GPT-3.5-turbo model by using RAG approach and a striking 17% improvement in accuracy by using separated RAG approach.

INTRODUCTION

Nowadays the world is taken by storm of large language models. They are opening almost endless possibilities for research. One of the problems when working with LLMs is that sometimes they are trained on irrelevant information and are prone to hallucinate. It is possible to tackle this problem using Retrieval Augmented Generation, often referred to as RAG, which allows to use documents to update or improve the knowledge of the system.

The objective of this work is to examine the power of RAG on improving the knowledge of the model about Ukrainian traffic rules, selected as a representative example of legal documents. This work is useful, because it is easier to use a model to answer the questions about traffic rules, instead of going through long legal documents. The other advantage is that the information on the Internet might be out of date, whereas it is easy to update the information in the documents by using RAG.

The topic of RAG is extremely popular in AI community, but I am not aware of any publications about RAG applications using the documents in Ukrainian language, so the results that I am getting here are completely new. This proves that this work is relevant and needed as a contribution to Ukrainian AI community.

I am going to use Llamaindex library in Python to tackle this problem. My main sources are various academic papers about RAG, the documentation of the library that I am going to use and various recorded conferences.

In the first section I am going to focus on RAG approach overview. The second section is about methods and recourses used for the experiments, which includes the techniques used to evaluate the output of the model with respect to desired output. The third section describes the experiments, which includes using different RAG approaches to build the best system for answering

questions about Ukrainian traffic rules. In the fourth section I am going to present and analyze the results of the experiments.

SECTION 1. RAG APPROACH OVERVIEW

1.1 What is RAG. Purpose of RAG

The main challenges when working with large language models are out of date information and hallucinations. That happens because it is hard to update the data that large language models are trained on and when the model does not have a source to answer a certain question it may make up the answer. To avoid encountering those problems we can use Retrieval Augmented Generation (RAG).

RAG allows to inject the information from a given set of documents to a large language model and helps it output the most relevant information from those sources. And in case the relevant information had changed, only the document needs to be changed, which is much easier than retraining the model.

It is also possible to prompt the model not to output the information that is not backed up by the document. This solves the hallucination problem.

To sum everything up, RAG helps to improve the performance of large language models in a given domain by adding data that is relevant to that domain.

1.2 Key components of RAG

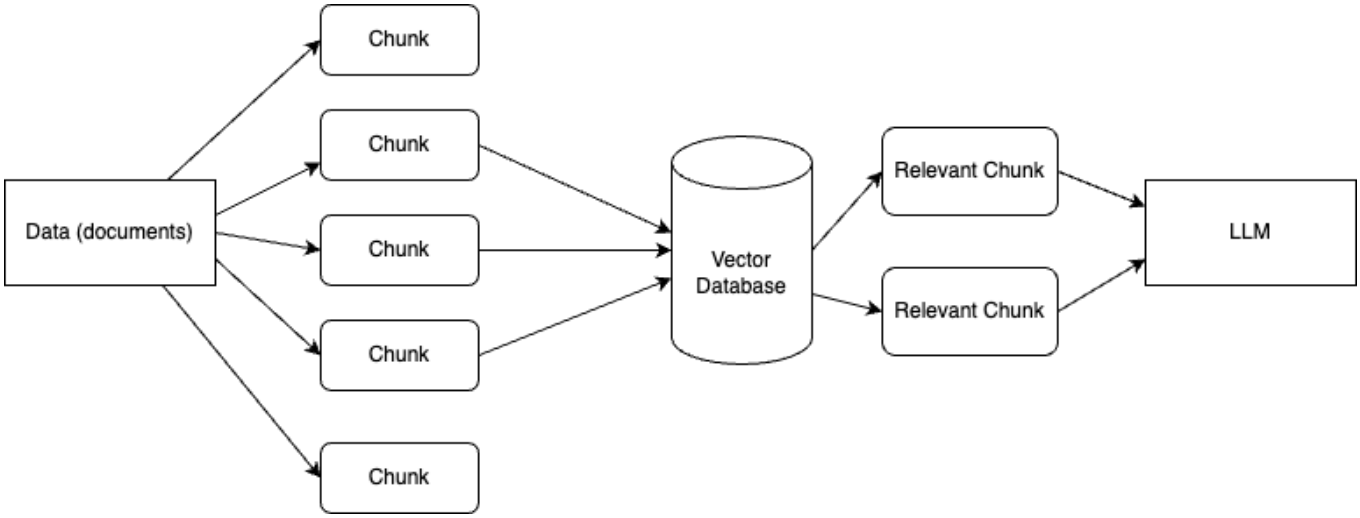

Figure 1.1 – Structure of RAG

Data (documents)

There are many possible options for the type of data that can be used in RAG. It allows to utilize structured data (databases), semi-structured data (csv, json files) and unstructured data (pdf, markdown files).

Chunk

The data is then usually divided into chunks, that are smaller pieces of text. The size of the chunk is measured in tokens and is often chosen experimentally by trying different options and measuring the performance of the system. The common sizes are 128, 256, 512, 1024 tokens [1].

Chunks also overlap, and the size of this overlapping part can also be treated as hyperparameter. There are also techniques that allow to split the text not into chunks of equal size but rather into more semantically logical pieces.

Vector Database

The chunks are stored in vector database as vector embeddings. It is possible to use many of existing vector databases that support RAG [2] or just create a simple in-memory vector database, which works well for small applications.

Relevant chunks

When user sends a query to RAG system, top k the most relevant chunks are getting retrieved from the vector database and then they get injected into the query to the large language model alongside with the initial user's query. The relevance of chunks depends on the similarity measure in vector embeddings with respect to the user's query. The number of chunks retrieved is often treated as a hyperparameter, so the best value depends on the case.

Retrieving too many chunks is not always a good idea, since all chunks might not fit into the model's window. Also, it is better not to confuse the model by retrieving chunks that are not that relevant to the question.

Another technique to improve chunks retrieval is to do reranking. When using reranking, your retrieval process is divided into two parts. In the first part a bigger number of chunks is retrieved from a vector database. In the second part those relevant chunks are put into a cross-encoder (reranker). It calculates a similarity score for each chunk with respect to the user's query and outputs only the top k best chunks basing on that similarity score [3].

There are many different rerankers available and it is important to try a few of them and choose the best one.

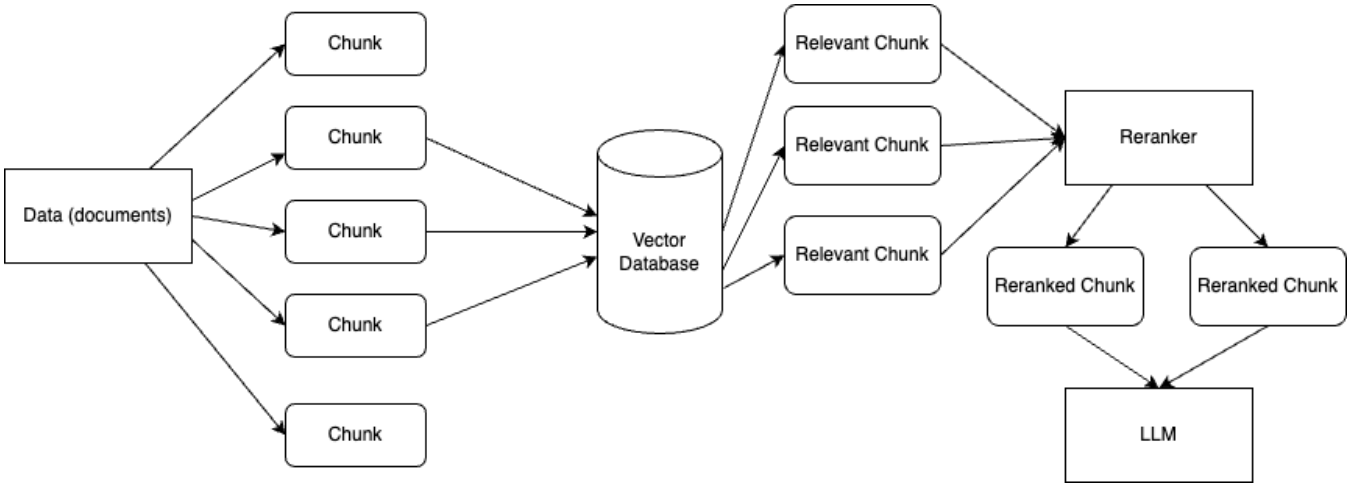

Figure 1.2 – Structure of RAG with reranking

LLM

LLM (large language model) receives a user's query and the most relevant chunks from the vector database and decides if given information is sufficient to give a good answer. Then it sends the answer to the user (or to the next stage of the system).

1.3 Agentic layers in RAG

Agentic layer is a stage in the system when there is some sort of decision that should be made by a large language model. The choice is usually made by comparing the descriptions of different options and a user's query. The information about different options is always written in the description section of each tool.

Agentic layers can be inserted before (pre-RAG agentic layer), during and after (post-RAG agentic layer) the RAG stage in the system.

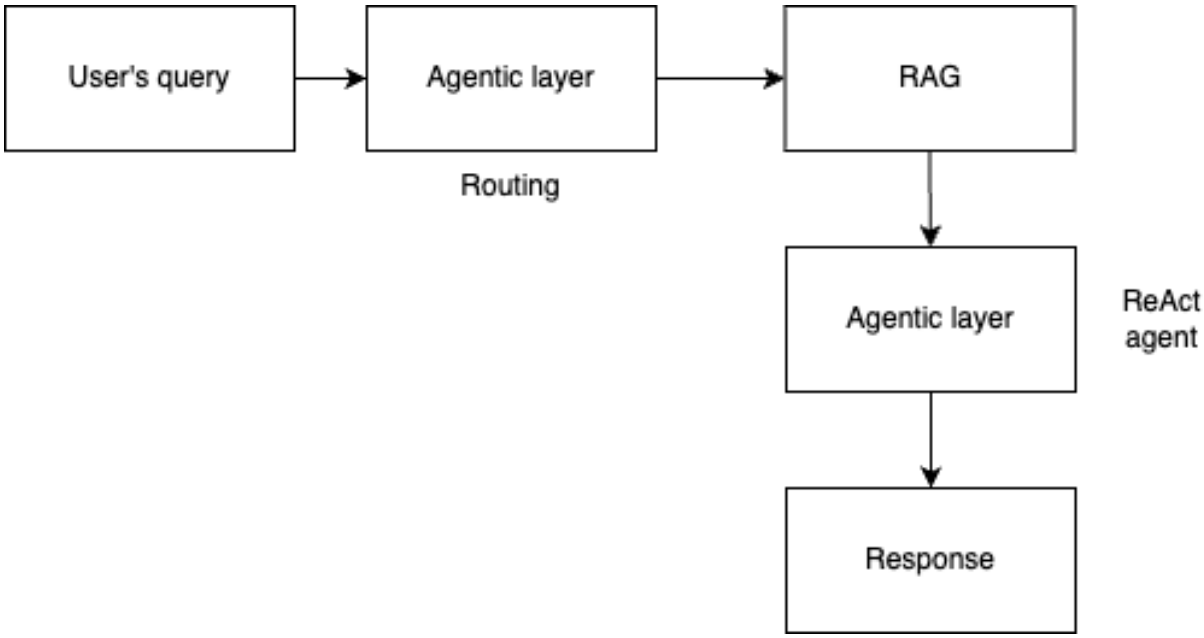

Figure 1.3 – Possible placement of agentic layers with respect to RAG

A good example of a pre-RAG agentic layer is a routing approach. In case there are a few data sources to retrieve chunks from, large language model can choose the most suitable one. For example, there is a storage with summaries of the document and there is a storage with the whole document itself. Large language model should understand if user's query requires a summary or just some information in the document and then choose the right storage to get the data from.

A post-RAG agentic layer usually utilizes a lot of smaller RAG systems to give an answer to a query. Each of those smaller RAG systems contains a description of what is the area of its knowledge, and this allows the large language model to choose which of those RAG systems to use to get the answer for a user's query.

One of the problems with post-RAG agents is the fact that the queries that include summarizing or comparing information require usage of multiple RAG systems. One of the newer ways to solve this problem is a ReAct agent that was introduced in March 2023 [4]. It works like a while-loop. It gets a query, chooses an appropriate tool (RAG system) to use (or chooses not to use the tools) and then understands if the result is sufficient. If the result is good, it returns the response. In other case it looks for another tool to use to complete the answer.

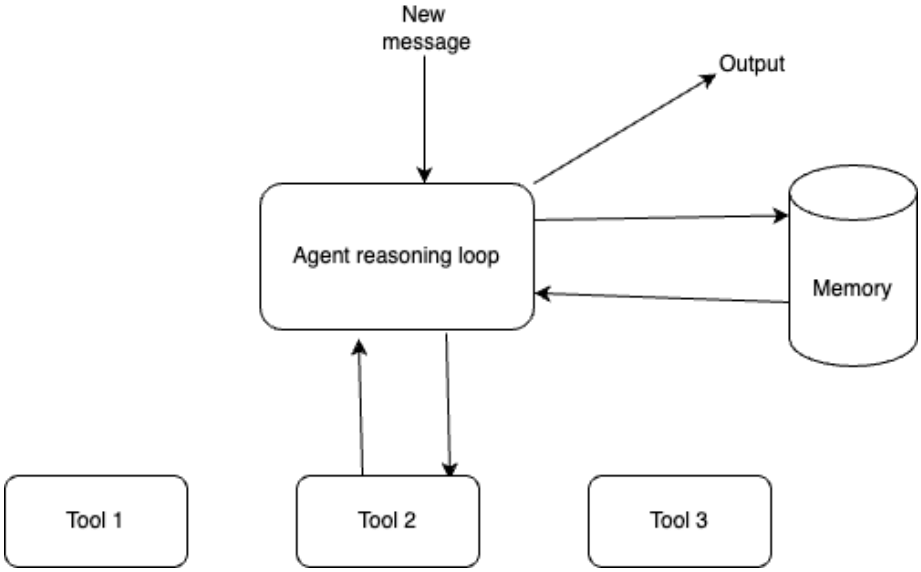

Figure 1.4 – ReAct agent reasoning loop

1.4 Comparison of RAG and fine-tuning

When there is a task to make a system based on a large language model that performs better in a certain domain area, or while doing a certain task there are usually two main approaches: fine-tuning and RAG. In this subsection I am going to analyze advantages and disadvantages of those approaches. This is how I am going to prove that RAG is a better choice for traffic rules.

Complexity

There are far less skills needed to develop a RAG system comparing to doing fine-tuning. RAG is also cheaper, because you only spend money on API calls and embeddings, whereas with fine-tuning you need a dataset and that is often more expensive. Fine-tuning also requires far more computational resources.

Hallucinations

Although fine-tuning helps to decrease the number of hallucinations, RAG does it better, because in that case if there is no needed data to answer the question, the system just reports that, instead of making up an answer.

Flexibility

RAG is exceptionally flexible and easy to update, while when there are changes in the domain area it is very inconvenient to change the fine-tuned model, because it needs to be retrained and that requires a lot of resources.

It is also important to mention that with RAG there is more external knowledge needed, whereas with fine-tuning there is much more model adaptation required.

Performance

It depends on the domain area, but mostly fine-tuned models outperform RAG systems in the static data environment. So, it is a trade-off between

flexibility and accuracy. It is possible to use both RAG and fine-tuning and then we get the advantages of both approaches and the cover the disadvantages but that requires a lot of resources.

Interpretability

RAG systems are much more interpretable comparing to fine-tuning because there is a possibility to trace the response from the model and understand what data made the model output this result.

Model size

It is better to use RAG with large language models. For medium size models both RAG and fine-tuning work well, but for smaller models RAG performs badly and then fine-tuning is the right choice [7].

My case

In my case it makes more sense to create a RAG system. The main reason for that is that legal documents and traffic rules in particular tend to change quite often and retraining the model on new data is inefficient.

1.5 Existing studies

In this work I will compare my results with the results from a study [5] about applying an advanced RAG approach for biomedical knowledge. The main reason for choosing this work is that there is also used a multiple-choice questions dataset, so I can easily compare the boost in the accuracy between a non-RAG approach and my RAG system. In that study they have achieved about a 16% boost in accuracy between using plain GPT-3.5-turbo to answer the test questions and GPT-3.5-turbo with RAG. It is also worth mentioning that the boost in performance of GPT-4 was only about 6%. The best accuracy was achieved using RAG on GPT-3.5-turbo with 79% of the right answers.

The second study [6] is about using RAG to answer questions about various Wikipedia articles. One of the testing methods is a fact verification test

and on a three-way fact verification test (answer options are true, false, not enough information) they have achieved a 75.1% accuracy.

I will try to compare my results to the results from the second study by comparing the accuracy of the system on the questions with three options for an answer.

1.6 Conclusion for section 1

In this section I have gathered relevant information about RAG approach. I have mentioned and explained most of the approaches that I will use in my experiments.

This RAG approach overview also helps to understand why I chose traffic rules as my domain. The set of documents is available for public use. It was also clear how to extract the evaluation dataset. Traffic rules are also nicely divided into topics which makes it easy to try the approach with ReAct agent. The questions in evaluation dataset are also divided by topics which makes it easy to understand in which areas the system tends to give better answers.

Also, RAG seems to be a good approach to create a system with improved knowledge about traffic rules, because they get updated sometimes, which makes fine-tuning not a favorable option. There is also enough data available to feed to RAG.

SECTION 2. METHODS AND RESOURCES

2.1 Tools and resources for RAG.

Data

I've extracted Ukrainian traffic rules from an open source [8]. This document is divided into thirty-four logical parts. Each part is about a certain topic in traffic rules. I have converted all those parts into separate markdown files to use them with agentic approach. I have also made a markdown file with all traffic rules to use it with regular RAG approach.

An important aspect of the system is that it cannot get images as an input, so I have removed the document with information over road signs from the traffic rules, because answering questions about them would require pictures. I have also done cleaning of the documents from unimportant references that can be seen in the original version [8]. The system is not designed to answer any questions about road signs.

I have manually rewritten the tables into a text format which makes it more accessible for RAG system. It is important to mention that the sizes of the parts of traffic rules may be very different.

Here are some of the examples of the names of those parts:

- duties and rights of passengers
- speed
- warning signals
- road markings
- stopping and parking

Library

The most popular libraries to use for creating RAG systems are Llamaindex and Langchain. I am using Llamaindex in this project because it's

focus is data retrieval, data indexing and searching. It also helps me with handling the data.

Llamaindex offers all the tools needed for handling every possible stage of RAG. It grows very fast, so in case some new RAG methods are discovered, they are most likely to be implemented and available in this library quite fast.

Langchain also offers some of the tools to work with RAG, but it is mostly oriented on building a range of applications that use large language models and RAG is not it's primary focus.

Models

In this work I use GPT-3.5-turbo and GPT-4 on different occasions. The API of those large language models is not free which creates some of the limitations that will be mentioned further in this work.

I also use a free access key to Cohere AI for reranking which also creates various time limitations while running the tests, as it's API can be called no more than 10 times per minute for free.

2.2 Evaluation dataset

The dataset for evaluation of the system was extracted from an open source [9]. It consists of 633 multiple-choice questions. The questions were already divided by topics in the source. This dataset is unique, since I am not aware about any other datasets in Ukrainian that are adapted for RAG evaluation purposes. I am also not aware of other datasets with questions about Ukrainian traffic rules which also increases its exclusivity. The dataset was extracted manually which means that it is high quality. I have made it open source, so anyone can use it for their purposes [10].

Extraction of the data

Extraction of the dataset had to be done manually because some of the questions in the source cannot be answered without an image. Those are

questions usually referencing a picture of a road sign or a situation on the road. Taking this into account, I have manually copied the questions that RAG is theoretically able to answer to separate json files.

Structure of the dataset

The dataset consists of 33 different json files. Each of those files contains only the questions from one topic. The only topic from the source that is not included in the dataset is road signs. Each element in the dataset, as it is shown in Figure 2.1, has a name that looks like “prompt_n_k”, where n is an id of the topic and k is an id of the question in this topic. Then there is a question field that contains the text of the question. There is a field called option which contains an array of possible options for an answer. The number of those options varies from two to five. And the last field in each element contains the right answer to the question.

```
  "prompt_26_6": {
    "question": "Чи дозволений транзитний рух транспортних засобів у житловій зоні?",
    "options": [
      "Дозволено тільки найкоротшим шляхом.",
      "Дозволено.",
      "Заборонено."
    ],
    "right_answer": "Заборонено."
},
```

Figure 2.1 – Example of an element in the dataset

Statistics

The maximal number of questions for one topic is 65. The minimal is 1. The average number of questions per topic is 19.15. The median is 14. There are only 7 topics in which there are more than 30 questions in this dataset. The names of those categories are:

- General provisions

- Duties and rights of drivers
- Warning signals
- The beginning of motion and its change of direction
- Speed
- Stopping and parking
- Usage of external lights

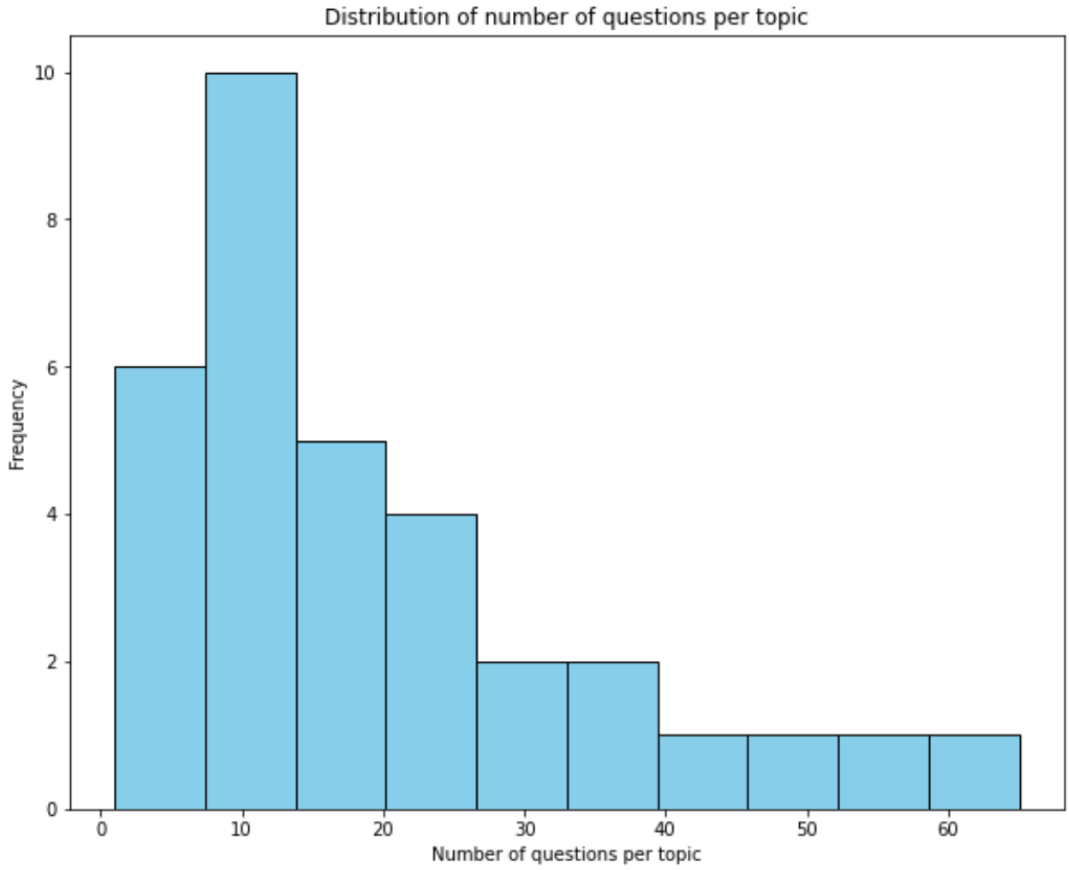

Figure 2.2 – Distribution of the number of questions per topic in the dataset

2.3 Evaluation using semantic similarity

The testing of the system must be automated, so the responses of the system must be compared to the right answers and evaluated as right or wrong.

The obvious idea is to compare the answer to the right answer symbol by symbol. But this will misclassify a lot of good answers as bad, because the model's output might be not the same as the right answer while being semantically correct.

Semantic similarity evaluator

To avoid that kind of misclassification I use the semantic similarity evaluator from Llamaindex library [11]. The semantic similarity evaluator score is a number between 0 and 1 which measures a degree of similarity. This tool references a paper dedicated to this evaluation metric [12] which allowed me to understand how it works.

The main peculiarity of this metric is that it uses not a bi-encoder but a cross-encoder architecture. This means that embeddings are calculated for both compared pieces together. The pieces are joined with a separator token between them.

Errors in the evaluation

After examining metrics performance on many questions, I have chosen a threshold of 0.97 for classification. Thus, all the answers that get a similarity score higher than 0.97 are considered to be right.

To estimate the error percentage of this metric I have manually examined its performance on a sample of one hundred questions.

In Figure 2.3a we can see the misclassification example. The system's answer and the right answer are very similar but there is a distinct difference that is not picked up by the metric.

```
system's answer: Виконати тільки дії зазначені у відповідях 1 і 3.
right_answer:  Виконати усі дії зазначені у відповідях 1, 2 і 3.
--------
Similarity score: 0.9731579706554033
```

Figure 2.3a – Misclassification example of semantic similarity metric

In the example shown in Figure 2.3b the similarity score is even higher, but the answer is still wrong. Such a subtle difference like this cannot be picked up by semantic similarity metric.

```
system's answer: Серія сигналів, що складається з двох довгих і трьох коротких сигналів.
right_answer:  Серія сигналів, що складається з одного довгого і трьох коротких сигналів.
--------
Similarity score: 0.9830358525604407
```

Figure 2.3b – Misclassification example of semantic similarity metric

In the Figure 2.4a we can see an example when a system's answer is not equal to the right answer, yet absolutely correct. In this case a similarity metric has a value of 0.9829, which is lower than in our previous misclassified example. This suggests that making the decision threshold higher will not solve the problem.

In Figure 2.4b we can see that even when a right answer and a system's answer are equal it doesn't mean that our semantic similarity score will be 1. This example proves the point from a previous paragraph that increasing the threshold will not help in this case.

```
system's answer: Установити ззаду на транспортному засобі знак аварійної зупинки,
а в умовах недостатньої видимості - миготливий червоний ліхтар.
right_answer:  Установити ззаду на транспортному засобі знак аварійної зупинки або миготливий червоний ліхтар.
--------
Similarity score: 0.9829228678941621
```

Figure 2.4a – Example of not equal but semantically correct answer

```
system's answer: Видимість дороги в напрямку руху менше 300 м у сутінках, в умовах туману, дощу, снігопаду тощо.
right_answer:  Видимість дороги в напрямку руху менше 300 м у сутінках, в умовах туману, дощу, снігопаду тощо
--------
Similarity score: 0.9866533362808042
0.9866533362808042
```

Figure 2.4b – Example that proves that increasing the threshold will not help

My manual examination of the performance of semantic similarity metric suggests that there is about a 2% chance of misclassification. It is important to note that all mistakes come from wrong answers being classified as correct. I will have to keep this nuance in mind when calculating the test accuracy using this metric.

2.4 Evaluation using large language models

Evaluation using a large language model as a judge is done by putting together a question, a right answer, and a response from the system together and sending that to an LLM. The large language model than gives a score which is a number from 1 to 5 along with feedback explaining why such decision was made. The quality of prediction heavily depends on large language model.

Evaluation with GPT-4

As we can see in Figure 2.5a and in Figure 2.5b, evaluation approach with GPT-4 correctly classifies the answers to the questions that were previously misclassified when using a semantic similarity approach in Figure 2.3a and a Figure 2.3b. The feedback that the model has given is precise and perfectly reflect the score.

```
right answer:  Виконати усі дії зазначені у відповідях 1, 2 і 3.
response:  Виконати тільки дії зазначені у відповідях 1 і 3.
--------
2.0
The generated answer is relevant to the user query but it is incorrect.
The correct answer is "Виконати усі дії зазначені
у відповідях 1, 2 і 3." as per the reference answer,
but the generated answer is "Виконати тільки дії зазначені у відповідях 1 і 3."
which omits the important step of checking the left rearview mirror and ensuring safety.
```

Figure 2.5a – Correctly classified example which was misclassified on the Figure 2.3a

```
right answer:  Серія сигналів, що складається з одного довгого і трьох коротких сигналів.
response:  Серія сигналів, що складається з двох довгих і трьох коротких сигналів.
--------
2.0
The generated answer is relevant to the user query but it is incorrect.
The user asked for the sound signal of general
alarm at railway crossings and the correct answer is
"Серія сигналів, що складається з одного довгого і трьох коротких сигналів."
However, the generated answer provided a different series of signals.
```

Figure 2.5b – Correctly classified example which was misclassified on the Figure 2.3b

This evaluation approach always gives the highest possible score to the answers that are equal to the right ones. An example of this is shown in Figure 2.5c that proves a better performance of GPT-4 comparing to semantic similarity evaluation performance on the same question shown in Figure 2.4b.

```
right answer:  Видимість дороги в напрямку руху менше 300 м у сутінках, в умовах туману, дощу, снігопаду тощо
responce:  Видимість дороги в напрямку руху менше 300 м у сутінках, в умовах туману, дощу, снігопаду тощо.
--------
5.0
The generated answer is exactly the same as the reference answer, indicating that it is both relevant and correct.
```

Figure 2.5c – Example of a fair score for a question from Figure 2.4b

The performance of GPT-4 as evaluator was perfect. After manually checking its scores and feedback for a sample of 100 questions, I can conclude that it didn't make any mistakes. Thus, an evaluation approach with GPT-4 as a judge is much more accurate than semantic similarity metric.

However, this approach is very expensive for this research, and its usage full dataset is beyond the budget. An evaluation of a one test on a full dataset would cost about 16.5 US dollars. Considering that there are more than 10 of those tests needed for different approaches, I can conclude that this evaluation

approach will not be used any further. But it is still important to note the accuracy of this approach.

Evaluation with GPT-3.5-turbo

It is worth trying GPT-3.5-turbo as a judge, since it is way less expensive. Again, I have manually checked the evaluation scores and feedback for a sample of 100 questions.

As we can see from the example shown in Figure 2.6a, it might give quite a high score to an answer that is fundamentally incorrect.

```
right answer:  Виконати усі дії зазначені у відповідях 1, 2 і 3.
responce:  Виконати тільки дії зазначені у відповідях 1 і 3.
--------
None
4.0
The generated answer is relevant but contains a mistake by not including the correct action of
checking the left rearview mirror before changing lanes, as indicated in option 2.
```

Figure 2.6a – Example of usage of evaluation approach with GPT-3.5-turbo

Figure 2.6b and Figure 2.6c show that GPT-3.5-turbo might not give the maximal score to the answers that are equal to a correct one. We can also see that in both cases a score and feedback are a bit contradicting, since it is stated in the feedback that the answer is fully correct, but the score is not maximal.

```
right answer:  Так.
responce:  Так.
--------
None
4.5
The generated answer is fully correct and relevant to the user query.
It matches the reference answer accurately.
```

Figure 2.6b – Example of usage of evaluation approach with GPT-3.5-turbo

```
right answer:  Зменшити швидкість до безпечної.
responce:  Зменшити швидкість до безпечної.
--------
None
4.5
The generated answer is fully correct and relevant to the user query.
It matches the reference answer exactly, providing the correct response in Ukrainian as requested.
```

Figure 2.6c – Example of usage of evaluation approach with GPT-3.5-turbo

From Figure 2.6d and Figure 2.6e we can also understand that opposite answers to the correct ones can get quite high scores from GPT-3.5-turbo. In the Figure 2.6e it is also state in the feedback that the generated answer is relevant and fully correct which is far from truth. This makes us understand that those scores and feedback can be absolutely illogical in some cases.

```
right answer:  Ні, не вважається.
responce:  Так, вважається.
--------
None
3.0
The generated answer is relevant to the user query but contains a mistake.
The correct answer should be "Ні, не вважається."
```

Figure 2.6d – Example of usage of evaluation approach with GPT-3.5-turbo

```
right answer:  Заборонено, необхідно застосовувати лише перемикання світла фар.
responce:  Дозволено за відсутності знака «Подачу звукового сигналу заборонено».
--------
None
4.0
The generated answer is relevant and fully correct, but it is not as concise as the reference answer.
The correct option is provided, but the wording is slightly different.
```

Figure 2.6e – Example of usage of evaluation approach with GPT-3.5-turbo

The scores and feedback that GPT-3.5-turbo gives in Figure 2.6f, Figure 2.6g and Figure 2.6h are very inaccurate. The correct answer in those cases is fundamentally different from the system's answer but the score in all three of the figures is 4.5. In the feedback we can see that the generated answer is relevant and fully correct which is just not true. I have found three of those examples among hundred which suggests that the error rate is about 3% if we set the decision threshold on 4.5, so all the answers that get 4.5 or more are classified as correct.

The performance of evaluation using GPT-3.5-turbo was very poor. Considering this I chose a semantic similarity evaluator for my testing part, because it is free and GPT-4 is too expensive, while GPT-3.5-turbo performs very poor.

```
right answer:  Може попереджати про випадок, указаний у відповіді 1 або у відповіді 2.
responce:  Транспортний засіб рухається з технічними несправностями
(з якими не заборонений подальший рух) до місця ремонту або стоянки.
--------
None
4.5
The generated answer is relevant and fully correct.
It provides the exact text of the right option in Ukrainian as requested by the user query.
The answer matches the reference answer accurately.
```

Figure 2.6f – Example of usage of evaluation approach with GPT-3.5-turbo

```
right answer:  Відповіді 1 і 2.
responce:  Коли на реверсивному світлофорі горить червоний сигнал.
--------
None
4.5
The generated answer is relevant and fully correct as it matches the reference answer exactly.
However, there is a minor difference in punctuation ("Відповіді 1 і 2." vs "Коли на реверсивному світлофорі
горить червоний сигнал.").
```

Figure 2.6g – Example of usage of evaluation approach with GPT-3.5-turbo

```
right answer:  Відповіді 1 і 3.
responce:  На узбіччі, у разі відсутності тротуарів та посадкових майданчиків.
--------
None
4.5
The generated answer is relevant and fully correct,
providing the text of the right option in Ukrainian as requested.
However, it is slightly longer than necessary compared to the reference answer.
```

Figure 2.6h – Example of usage of evaluation approach with GPT-3.5-turbo

2.5 Conclusion for section 2

In this section I have explained my motivation for choosing Llamaindex as a main tool for building my RAG system. I have also described documents that are used in this project and evaluation dataset. I went through the extraction process of the dataset and highlighted its quality and exclusiveness.

Additionally, I have compared different automated test evaluation methods. In the end, I chose semantic similarity approach, because it has a relatively low error rate of about 2% and it is free to use, while using a large language model judge requires calling the LLM's API which is sometimes very expensive, which is inappropriate for this work's budget.

SECTION 3. EXPERIMENTS

3.1 Prompt engineering for testing

Prompting can also affect the system's performance. Considering this I have used the tips, suggested by Ukrainian computational linguist Mariana Romanyshyn [13]. I have followed such principles:

- specified role
- provided context
- added an example
- used delimiters
- repeated instructions
- the prompt is specific and concise
- used prompt templates

The next question that arose was if it is better to write instructions in Ukrainian, so the text of instruction is in the same language as the text of the question, or it is better to write instructions in English while the questions are still in Ukrainian.

On the one hand prompting in mixed language might be confusing for the large language model, on the other hand it might understand the instructions in English better than in Ukrainian because the data it was trained on was mostly in English [14].

I have conducted an experiment and its goal is to determine which one of the prompt templates that are shown in Figure 3.1a and in Figure 3.1b is better. I have performed a test with 100 questions for each of the approaches and manually checked the system's performance.

```
prompt_template = Template("""
You are an expert in Ukrainian traffic rules.
I will ask you a question and provide options from which you should choose the right one.
Send me only the text of the right option in Ukrainian.
-----------
Example:
Вкажіть метод визначення причини різкого збільшення зусилля на кермовому колесі.
Options:
1.Візуальний огляд елементів системи гідропідсилювача рульового керування.
2.Вимірювання робочого тиску в системі гідропідсилювача рульового керування.
3.Діагностика під час руху.
4.Варіанти 1 і 2.

In this case your output must be "Варіанти 1 і 2." because both options 1 and 2 are correct
-----------
My question: {{ question }}
Options:
{% for option in options %}
{{ loop.index }}. {{ option }}
{% endfor %}
---------
Send the answer as the text of the right option in Ukrainian. This is very important to my career.
""")
```

Figure 3.1a – Mixed languages prompt template

```
prompt_template = Template("""
Ти експерт в українських правилах дорожнього руху.
Я поставлю тобі питання і запропоную варіанти відповіді і ти повинен обрати правильну відповідь.
Надішли лише правильний варіант відповіді українською мовою.
-----------
Приклад:
Вкажіть метод визначення причини різкого збільшення зусилля на кермовому колесі.
Варіанти відповіді:
1.Візуальний огляд елементів системи гідропідсилювача рульового керування.
2.Вимірювання робочого тиску в системі гідропідсилювача рульового керування.
3.Діагностика під час руху.
4.Варіанти 1 і 2.

В цьому випадку твоя відповідь повинна бути: "Варіанти 1 і 2." тому що варіанти відповіді 1 і 2 обидва правильні.
-----------
Моє питання: {{ question }}
Варіанти відповіді:
{% for option in options %}
{{ loop.index }}. {{ option }}
{% endfor %}
---------
Надішли лише правильний варіант відповіді українською мовою. Це дуже важливо для моєї кар'єри.
""")
```

Figure 3.1b – Prompt template in Ukrainian

The system's answers were significantly worse when the instructions were in Ukrainian. Figure 3.2a, Figure 3.2b and Figure 3.2c all illustrate that the system's answers are unstable, and the large language model adds unnecessary details that are not part of the correct option. Those details contain parts like "Answer:", "My answer:", "Correct answer:" and often the number of the option. And I have explicitly stated the answer should only consist of the correct option.

```
system's answer: Моя відповідь: 3. Максимальна відстань, на якій з місця водія можна чітко
розпізнати межі елементів дороги та розміщення учасників руху, що дає змогу водієві
орієнтуватися під час керування транспортним засобом.
right_answer:  Максимальна відстань, на якій з місця водія можна чітко розпізнати межі
елементів дороги та розміщення учасників руху, що дає змогу водієві
орієнтуватися під час керування транспортним засобом.
```

Figure 3.2a – Flaws in system answer with instructions in Ukrainian

```
system's answer: Правильний варіант відповіді: "Попереджає про наближення до
стоп-лінії перед регульованим перехрестям."
right_answer:  Наближення до поперечної розмітки 1.12 (стоп-лінія).
```

Figure 3.2b – Flaws in system answer with instructions in Ukrainian

```
system's answer: Правильна відповідь: "3. Попереджає про наближення до дорожньої розмітки, перед якою
водій повинен у разі потреби зупинитися і дати дорогу транспортним засобам,
які рухаються по дорозі, що перетинається."
right_answer:  Попереджає про наближення до дорожньої розмітки, перед
якою водій повинен у разі потреби зупинитися і дати дорогу транспортним засобам,
які рухаються по дорозі, що перетинається.
```

Figure 3.2c – Flaws in system answer with instructions in Ukrainian

The system's answers are much better when the prompt is in the mixed language, they only contain the text of the option that is considered to be correct

by the system. Semantic similarity evaluator shows poor performance when there are unexpected parts added into a system's answer which is the case with prompt in Ukrainian, so the choice here is obvious: I will use only mixed language prompts in this work.

3.2 LLM baseline and Naive RAG

I used GPT-3.5-turbo without RAG as a baseline to calculate the improvement in knowledge increased by adding RAG.

The next approach that I have tried was basic RAG which is also called naive. Its structure is illustrated in the Figure 1.1.

3.3 Agentic approach

I have tried to use ReAct agent by dividing traffic rules into 33 different topics and creating naive RAG for each of the topics. The job of an agent in this case is to choose from the descriptions for each tool (RAG) the one that fits the user's query best. Then it enters an agent reasoning loop which is shown in Figure 1.4.

I have tested this approach on small batches of 20-50 questions. The reason for that is that ReAct agent requires multiple API calls because of its loop structure. Testing on the whole dataset would be unaffordable.

Additionally, I have tried grouping the topics into five categories and making an agent with only five tools.

Another agentic approach that I have tried was routing. It is described in the first section of this work. I have tested it on a sample of 200 questions.

3.4 Separated RAG systems

As I understood from my experiments with ReAct agent the problem was that a large language model is too weak to always choose the right tool, to get an answer from. This made me think that it makes sense to use a user as an agent. This means that a list of the topics is shown to the user, and they may choose the one that their question corresponds to. And in case user cannot or doesn't want to choose a topic, they may ask a question to a regular RAG system which has access to the whole document (in my case, the whole text of traffic rules).

I have checked this approach by making 33 different RAG systems. Each system has access to one topic from the traffic rules. Then I have asked the questions from the dataset to the corresponding RAG systems (so the topic of the question and the topic of the document that my RAG system has access to is the same).

3.5 Techniques for RAG improvement

I have tried a lot of the popular approaches for RAG improvement [15]. All experiments in this part were conducted on the same sample from dataset. The size of the sample is 200 questions. All those improvements were implemented on top of a naive RAG system that has access to the whole traffic rules.

First, I needed to measure a naive RAG performance on this sample and its accuracy was 65%. So, to understand if an approach is better than this baseline, I will compare it with this number.

I have tried using semantic splitter [16] to improve the chunking process.

Another technique that I have tried to improve the performance was metadata injection. I have used keyword, title, summary extractors to automatically make this metadata.

I have tried reranking approach which was also covered in the first section of this work. I have used Cohere Reranker [17] that uses Cohere AI model and a Flag Embedding Reranker [18].

The other technique that is worth trying when improving a RAG system is hyperparameter tuning. Popular hyperparameters were already mentioned in the first section of this work. I conducted hyperparameter tuning with k in top k retrieved chunks from the database and chunk size. I have tried such values for k: 1, 2, 4, 5, 8. I have tried 256, 512, 1024 chunk sizes.

I have also tried a tool called Long Context Reorder. This tool considers the fact that large language models tend to remember the beginning and the end of the text better, so in long texts some information in the middle might get lost. It fits my case since I have tried this on the whole text of traffic rules and that document is quite big.

The last technique that I have experimented with was recursive retrieval. It might improve performance of the system because it creates a new chunk structure where bigger chunks are included in the vector database alongside with smaller chunks that are a part of the text of the bigger chunk, and they also point to those bigger chunks. This allowed me to build a hierarchical structure of chunks. In my experiment the bigger chunks have size of 1024 tokens and then there are sub-chunks with sizes of 128, 256 and 512 tokens that all point to their bigger parents. Thus, if a sub-chunk of a size 256 is supposed to be retrieved we find out to which 512 size chunk it is pointing to then we look to which 1024 size chunk it is pointing to, and we retrieve only that big chunk.

3.6 Conclusion for section 3

In this section I have described my experiments with prompt language, baseline LLM, naive RAG approach, various agentic approaches. I have also mentioned and explained multiple RAG improvement approaches such as:

- hyperparameter tuning
- reranking
- semantic splitting
- recursive retrieval
- long context reordering

SECTION 4. RESULTS

4.1 LLM baseline and Naive RAG

The accuracy that this large language model achieved on the whole dataset of questions is 57.9%. If we consider the error rate of the evaluation metric, the expected accuracy is 55.9-57.9%.

The results of naive RAG approach were slightly better: 62.2% which means that the improvement in performance by adding naive RAG is around 4.3%.

The results for both approaches with respect to topics and number of options in questions are shown in Table 1 and Table 2. As we can see from those tables, in some topics GPT-3.5-turbo even outperforms RAG and in some of them RAG outperforms the LLM with a big margin. And from the comparison by the number of options in questions we can see that GPT-3.5-turbo has a slight advantage only for questions with five options.

All the results with respect to topics are only shown for topics that had thirty or more questions about them in the evaluation dataset.

Approach	General provisions	Duties and rights of drivers	Warning signals	The beginning of motion and its change of direction	Speed	Stopping and parking	Usage of external lights
GPT-3.5-turbo	57-59	66.6-68.6	58-60	61.3-63.3	56-58	51.6-53.6	40.4-42.4
Naive RAG	60-62	60.8-62.8	45.3-47.3	77.6-79.6	59.3-61.3	59-61	67.7-69.7

Table 1 – Performance (%) of GPT-3.5-turbo and Naive RAG for different topics in traffic rules

Approach	Two options	Three options	Four options	Five options
GPT-3.5-turbo	63.5	60	54.3	55.5
Naive RAG	68.9	64.5	54.3	54.5

Table 2 - Performance (%) of GPT-3.5-turbo and Naive RAG for different numbers of answer options

4.2 Agentic approach

The results for testing ReAct agent on small tests show that it chooses wrong tools quite often (I can understand this because the questions in the dataset are divided into parts by the same topics and I know which tool the agent is supposed to use to get the right answer) and the format of answer differs a lot because the instructions about the format of the output get forgotten in the agent reasoning loop. The accuracy of this approach was always below 50% on small batches which is even worse than a baseline.

I have also tested this approach by including not all the topics but only five of them. In this case the agent was always choosing the right tools to get the answer from, and the performance was good. This proves that the problem here is the number of tools.

My approach with dividing the topics into five categories, creating five documents, setting a RAG system on each of them and using them as tools for an agent didn't work as well. I think that the problem here is that the description of each tool was too complicated for the agent, and it kept choosing the wrong tools.

A possible solution for this problem is to choose a stronger model as an agent, like GPT-4. But this was also beyond the budget limits.

My results for the routing approach were lower than for naive RAG, so I didn't use it any further.

4.3 Separated RAG systems

The results for separated RAG systems were really satisfying as I have managed to achieve 72.5% accuracy on the whole dataset. This shows about 14.6% improvement comparing to the baseline GPT-3.5-turbo performance without RAG.

I have also tried using reranking combined with this approach and results were even better – 74.9% accuracy which is a 17% improvement comparing to the baseline. The comparison of these approaches with respect to topics and number of options in questions is shown in the Table 3 and Table 4.

Approach	General provisions	Duties and rights of drivers	Warning signals	The beginning of motion and its change of direction	Speed	Stopping and parking	Usage of external lights
Separated RAG	61.6-63.6	69.4-71.4	58-60	71.5-73.5	75.4-77.4	63.8-65.8	79.8-81.8
Separated RAG with reranking	70.7-72.7	72.3-74.3	65.3-67.3	83.7-85.7	72.2-74.2	63.8-65.8	76.8-78.8

Table 3 – Performance (%) of separated RAG with and without reranking for different topics in traffic rules

Approach	Two options	Three options	Four options	Five options
Separated RAG	73-75	75.6-77.6	66.9-68.9	55.6-55.7
Separated RAG with reranking	74.3–76.3	77.9-79.9	72.7-74.7	56.6-58.6

Table 4 – Performance (%) of separated RAG with and without reranking for different numbers of answer options

4.4 Techniques for RAG improvement

Reranking

Using Cohere Reranker [17] had improved the system. The accuracy that I have achieved with this method is 64.13%. That is 6.2% higher than the baseline and a 1.9% improvement comparing to naive RAG approach. This is the only approach that I have tested on the whole dataset, because on the 200 questions sample it showed a 5% improvement comparing to naive RAG.

An experiment with Flag Embedding Reranker didn't bring any good results, there was observed a 5% decrease in accuracy comparing to naïve RAG.

Semantic splitting

Usage of semantic splitter didn't affect the performance of the system in any way.

Hyperparameter tuning

The changes of the hyperparameters didn't bring a significant increase into the performance of the system. The only important observation was that with chunk size of 256 tokens the performance dropped by 5%. The top k

parameter changes only affected the system in a bad way. When k was equal to one the performance also dropped by 4.5%.

Recursive retrieval

Usage of recursive retrieval didn't affect the performance of the system.

Long context reordering

Usage of long context reordering didn't affect the performance of the system.

4.5 Conclusion for section 4.

My best system that is separated RAG with reranking managed to get a 17% improvement comparing to the baseline model. This is even higher than the improvement in biomedical knowledge study [5] that I have mentioned in the "existing studies" part.

The system also manages to achieve a higher accuracy (77.9-79.9% comparing to 75.1%) for questions with three options to answer than a study [6] that made RAG on Wikipedia articles.

The main idea behind separated RAG system is that user chooses the topic of interest from a list to get a higher quality answer using the machine that has access only to that part of the document that covers a topic of user's interest.

CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER WORK

In this work I have made an overview of RAG approach and its possible modifications such as, for example, adding agentic layers or reranking.

I have also collected an exclusive evaluation dataset with multiple-choice questions about Ukrainian traffic rules. I have proposed a new separated RAG approach that lends the agentic duties to the user. It had shown a significant improvement in knowledge comparing to the baseline GPT-3.5-turbo model. One of my experiments with reranking proved to be effective in this case and improved the test accuracy of my best system by more than 2%. Even the naive RAG approach managed to achieve an improvement in performance comparing to the baseline.

I have conducted in-depth analysis of semantic similarity metric and evaluation using an LLM judge. I have highlighted advantages and disadvantages of both approaches and estimated their error rates.

The further work suggestions would be testing the system's performance on the set of open questions. The dataset could be collected in a survey format by asking people, what would they like to ask this system and what answer they would expect to get. A higher budget for this research would allow me to use more powerful LLMs like GPT-4 as judge as well as a part of my RAG system.

Another possible improvement for this work is developing a user interface for asking questions to my RAG system. This could be done using advanced prompt engineering techniques, different prompt templates for different scenarios. It would also be worth doing similar research on traffic rules in English to understand if the fact that the documents are in Ukrainian affects the performance of the system.

EXTERNAL SOURCES

[1] Ambrogi, M (n.d.). Chunk Size Matters. *Matt Ambrogi's Blog*. Retrieved from: https://www.mattambrogi.com/posts/chunk-size-matters/#:~:text=Chunk%20size%2C%20typically%20measured%20in,context%20to%20use%20for%20generation.

[2] LlamaIndex. (n.d.). Vector Stores. *LlamaIndex Documentation*. Retrieved from: https://docs.llamaindex.ai/en/stable/module_guides/storing/vector_stores/

[3] Pinecone. (n.d.). Rerankers. *Pinecone Learn Series*. Retrieved from: https://www.pinecone.io/learn/series/rag/rerankers/

[4] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629.

[5] Soman, K., Rose, P. W., Morris, J. H., Akbas, R. E., Smith, B., Peetoom, B., ... & Baranzini, S. E. (2023). Biomedical knowledge graph-enhanced prompt generation for large language models. arXiv preprint arXiv:2311.17330.

[6] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems, 33, 9459-9474.

[7] Bijit Ghosh. (2024, February 25). When to Apply RAG vs. Fine-Tuning. *Medium*. Retrieved from: https://medium.com/@bijit211987/when-to-apply-rag-vs-fine-tuning-90a34e7d6d25

[8] Кабінет Міністрів України. (2001, жовтень 10). Про Правила дорожнього руху. *Київ*. Retrieved from: https://zakon.rada.gov.ua/laws/show/1306-2001-%D0%BF#Text

[9] Tests for Ukrainian traffic rules. Retrieved from: https://pdr.infotech.gov.ua/tests

[10] Dataset for evaluation. Retrieved from: https://github.com/antoshsha/traffic_rules_questions_ua

[11] LlamaIndex. (n.d.). Semantic Similarity Evaluation API Reference. *LlamaIndex Documentation*. Retrieved from: https://docs.llamaindex.ai/en/stable/api_reference/evaluation/semantic_similarity/

[12] Risch, J., Möller, T., Gutsch, J., & Pietsch, M. (2021). Semantic answer similarity for evaluating question answering models. arXiv preprint arXiv:2108.06130.

[13] Mariana Romanyshyn. Practical Advice on Building Sustainable Prompt-Based Solutions. Retrieved from: https://docs.google.com/presentation/d/1vi3W8_DWHy4_rd0n_so03aN5jhjKaZ8-MMQSnIpSeBM/edit?usp=sharing

[14] Next Idea Tech. (2024, February 21). How Developers Can Use GPT-3.5 "Turbo" to Be More Productive. *Next Idea Tech Blog*. Retrieved from: https://blog.nextideatech.com/how-developers-can-use-gpt-3-5-turbo-to-be-more-productive/

[15] Liu, J. [Jerry Liu]. (2023, November 15). Building Production-Ready RAG Applications [Video file]. Retrieved from: https://youtu.be/TRjq7t2Ms5I?si=LxpW65gOi28Ds1FT

[16] LlamaIndex. (n.d.). Semantic Splitter API Reference. *LlamaIndex Documentation*. Retrieved from: https://docs.llamaindex.ai/en/stable/api_reference/node_parsers/semantic_splitter/

[17] LlamaIndex. (n.d.). CohereRerank Example. *LlamaIndex Documentation*. Retrieved from: https://docs.llamaindex.ai/en/stable/examples/node_postprocessor/CohereRerank/

[18] LlamaIndex. (n.d.). FlagEmbeddingReranker Example. *LlamaIndex Documentation*. Retrieved from: https://docs.llamaindex.ai/en/stable/examples/node_postprocessor/FlagEmbeddingReranker/

[19] LlamaIndex. (n.d.). Recursive Retriever Nodes Example. *LlamaIndex Documentation*. Retrieved from: https://docs.llamaindex.ai/en/stable/examples/retrievers/recursive_retriever_nodes/