Universiteit Leiden

Master Computer Science

Retrieval-augmented generation for answering IT helpdesk questions

Name:	Michael Wheeler
Student ID:	s2041065
Date:	[19/11/2024]
Specialisation:	Data Science
1st supervisor:	Suzan Verberne
2nd supervisor:	Zhaochun Ren

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

This study evaluates the utility of using large language models to answer new IT helpdesk related questions using retrieval-augmented generation. We investigate different retrieval algorithms and indexation settings. In addition, we generate summaries using a large language model in order to reduce noise in the prompt. Furthermore, this research tries to evaluate the generated text using automated statistics but also through new model-based evaluation and manual evaluations. Our results show that generation is hard to evaluate using statistical models when using real-life data. In these cases it is better to use manual or model based evaluation. We conclude that the retrieval algorithm finds relevant information to solve a new problem but the large language model can have trouble in extracting these pieces of text from the given context. We think the model focuses too much on the question when it lacks domain knowledge to form a correct answer.

Contents

1 Introduction

In the world of IT customer support, the adage 'turn it off and on again' is often the quickest way to resolve a problem. However, when this simple fix doesn't work, helpdesk employees must rely on their own knowledge to solve issues for which they may not immediately know the solution. Another tool that technicians have in their disposal are so called Knowledge Bases (KB). A KB holds answers to frequent user questions. By querying the KB with certain keywords an answer might come up that is relevant to the user's question. The problem with a KB is that it can never hold every user question that might be asked and that new problems take time to get added. Therefor, the problem that this research tries to solve is to help helpdesk employees answer questions that they do not know the answer to.

In this research we make use of a helpdesk ticketing system called TOPdesk. A ticket is an interaction between the user and the support team. The IT domain that we operate in is that of Leiden University. TOPdesk has multiple functionalities but for our purpose we focus on user generated tickets. Employees or students of the university with IT issues can seek assistance by calling, emailing, visiting the in-person helpdesk, or submitting a ticket through the online system. Our research focuses specifically on these submitted tickets.

The on-boarding process for new helpdesk employees can take a long time. This is due to the fact that there are many things to learn. In order to learn, a new employee will sit next to an experienced employee and listen in to calls to understand how they handle various tickets. After a few sessions, the new employee will enter the queue while the expert sits next to them and helps in case it is needed. After a few sessions of this a new employee will start to work independently. This is a time consuming process and at the end there are still gaps in their knowledge that will only be relieved with time. This process puts a strain on the organisation and on the other helpdesk experts that have to train these new employees. A new employee might spent longer on solving tickets or be unable to solve them at all without asking colleagues. A larger knowledge base is not the answer to these problems. Not every problem can be described and adding new entries may take a long time. There have been cases of LLMs helping with the onboarding process of new employees but not yet in this field [2].

Our primary goal is to develop an approach that streamlines the on-boarding process for new team members but also facilitates in the creation of high quality tickets. Our vision is that Large Language Models (LLMs) can help new employees by suggesting the appropriate steps to take in order to help solve problems. This will also accelerate the resolution process for tickets. The LLM is a key part of our strategy that will evolve and grow alongside our employees by using the data that expert employees have previously put into the system.

Our approach makes use of Retrieval-Augmented Generation (RAG). This technique has seen a success in open domain question answering; therefore is promising to us in this work as well [25]. We will retrieve solved tickets by comparing the newly posed question to previously solved questions. Once we have gathered the 5 most relevant questions we add their solutions and ask our language model to come up with an answer to the new question with answers to the solved questions as context. We use an LLM to generate new answers. We have chosen to use a Dutch fine tuned model called 'GEITje' [22]. [1] GEITje is based on Mistral 7B and trained on a further 10 billion tokens of Dutch text [7].

This study addresses the question: '*Can we answer new IT related helpdesk questions using an LLM and RAG?*'. To address this question the following sub questions are considered.

[1] `https://huggingface.co/BramVanroy/GEITje-7B-ultra`

RQ1: Which retrieval algorithm has the best recall considering the data set?

RQ2: Can we use previously solved tickets as a knowledge base?

RQ3: How do we reduce noise in the prompt?

RQ4: How do we format the system prompt to get the best results?

RQ5: What is the most reliable to evaluate our generations results: statistical methods, model-based methods or manual evaluation?

Our main contributions can be summed up as follows. We demonstrate the differences between different retrieval algorithms and find that a dense retrieval algorithm gives us the best recall and precision. Furthermore, we propose that retrieval can be done based on a query-query retrieval strategy because of the natural redundancy in the data set. We find that the previous tickets are being used as an internal knowledge base by the helpdesk employees but our LLM struggles to retrieve the relevant information when given these previous tickets. Noise reduction through summarization by an LLM is an adequate strategy to limit the prompt length. A clear system prompts with enumerated steps helps in generating similarly formatted answers. Lastly, the statistical evaluations like ROUGE don't give us a proper insight into the performance when working with a non-curated data set, instead model-based or manual evaluation has to be performed.

2 Related Work

2.1 Retrieval for question answering

Karpukhin et al. use dense passage retrieval for open domain question answering [13]. Their system works in 2 stages. First they retrieve a set of contexts that probably contain the answer to the question. Afterwards a machine reader is used to select the most promising answer. They address the question whether a BERT model can answer a question without anymore pre-training. They found that their model outperformed other state-of-the-art models on most datasets. This work differs from ours in that they know that there is a relevant passage in the dataset. Our retrieval will focus on finding comparable older questions that might be relevant. We find that similarity between questions are not easy to judge. Zhang et al. face the same issue [29]. Their setting involves a community of a diverse set of users. Our setting has a strict amount of people answering questions but each with their individual style. They mention that BM25 has a hard time to find similarity in questions when the wording is different but the meaning is the same. They also find that low-quality answers pollute training data. Their solution is to build a translation-based model which is not something we implement.
Our data set consists of long and complicated dialogues where the questions have no easy yes or no answer.
Iyyer et al. [6] try to solve the issue of answering complex inter-relatable questions. They solve the problem by using a dynamic neural semantic parsing framework that they train using a weakly reward supervised guided search. The system relies on prior content in order to answer a question.

2.2 Large language models for question answering

Lingke is a fine-grained multi-turn chatbot for customer service [31]. This is a chatbot for an e-commerce company that sells many different kinds of product. Its utility is that it can respond to the customers question and if it doesn't have a response it will 'chit-chat'. Since there are many documents they select the sentences of the relevant documents that most likely have the correct information. This is comparable to sequential matching networks with slightly better response selection. This work is relevant to us since we also deal with the problem that a relevant answer might not exist in the retrieved context. In this case we want our LLM to handle it appropriately.
Liu et al. try to measure the effectiveness of prompt tuning [18]. The researchers present a novel empirical finding that properly optimized tuning can be effective in NLU tasks. They also say that it matches the performance of a fine-tuned model while having only a small percentage of tuned parameters. The main conclusion is that properly optimized prompt tuning can be comparable to fine-tuning.
Hallucination is still a big problem in LLMs [16]. The problem Li et al. is trying to solve is that of hallucinations in the responses of LLMs. They do this by retrieving a set of responses and then comparing these responses using different metrics in order to end up with the response that has the highest chance of having the ground truth in it. The results were not conclusive.
In Lester et al. they condition frozen language models to perform specific tasks [14]. Only a small number of task specific paramaters are tuned. They found that this technique is competitive with few-shot learning for ChatGPT-3 when the model size is large enough. This technique is also more efficient computationally compared to fine-tuning the entire model. We do not use paramater specific prompt tuning and will focus on adjusting the system prompt in order to get the most out of the model.
It is important to remember the deficiencies in modern LLMs. Liu et al. [17] give a nice example of an important shortcoming. They evaluated two settings: multi-document question answering and key-value retrieval. They found that when the important information is in the middle of the context the performance degrades significantly. When the relevant information is in the beginning or end of the context the model performs better. We must also be aware of this since some tickets are quite lengthy.

2.3 Retrieval-augmented generation for question answering

RAG was first developed by Lewis et al. [15]. In this paper they mention that LLMs store factual knowledge in its parameters and perform well on fine-tuned NLP tasks. However, they realize that LLMs underperform on knowledge intensive tasks. To combat this they combine pre-trained parametric seq2seq model and non-parametric memory. The non-parametric memory is a dense vector index of Wikipedia passages, obtained by a neural retriever. They find that their approach outperforms the baseline approach on three different open domain QA tasks.
Brynjolfsson et al. shows why this method can be so powerful [3]. They employ an AI tool to help technical support employees to answer questions. They fine-tune chatGPT on a large set of customer-agent conversations. The AI is a tool used by the employee to generate responses to the customer. The customer does not see the AI's generated response directly. In some cases the AI does not find a suitable answer in its training data and will not give a response. They found that especially low-skill workers' productivity and solution finding is increased compared to high-skill workers.

Another reason to use RAG in the question answering domain is because it reduces hallucination. That is what Shuster et al. show [24]. They experimented with a neural retriever on complex multi-turn dialogue context. They verify their findings by using human evaluation to show that hallucination reduces when using this method.
This technology has also seen success in the medical field. Wang et al. uses RAG [26] in coincidence with chatbots. They claim that chatbots currently provide high level generic responses and are unable to deliver individual guidance and clinical diagnosis. Using retrieval augmented generation the researchers try to improve the specificity of the responses. This is done by using recent medical research and trusted medical sources. They do not mention their retrieval system.
Jiang et al. [8] propose an alternative approach to Retrieval Augmented Generation (RAG). LLM's often suffer from hallucinations, a problem that RAG aims to address. Typically, retrieval is performed only once before the LLM generates a response, which can still result in hallucinations within the generated text. To combat this, the researchers introduce an iterative method that predicts upcoming sentences to anticipate future content, particularly when a sentence includes low-confidence tokens. Their technique outperforms the baseline in four knowledge-intensive tasks.
Siriwardhana et al. experiment with training the generator and retriever components for a specific domain [25]. They propose RAG-end2end, an extension to RAG that can adapt to a domain-specific knowledge base by updating all components of the external knowledge base during training. They add an extra training signal to introduce domain specific information to the generator. The novel part of this algorithm is the joint training of the generator and retriever. They achieve significant performance increases compared to the original RAG model.
There are different ways of leveraging the old information to answer new question. These researchers use a knowledge graph (KG) to search through old solutions to answer their questions. [28] They combine RAG with a KG to ensure that the retrieved solutions are relevant to the new problem. The KG also maps tickets to each other when they reference each other. They show bu using relevant evaluation criteria that their method outperforms the baseline and is currently being used by LinkedIn's customer service team and has reduced their median resolution time.

2.4 Prompt engineering

Prompting is a relatively new subject matter in the natural language processing sphere. There are some studies that delve into its workings to find out how it works. An example of this is the review done by Chen et al. [5]. They recommend 5 basic strategies in order to get the most out of an LLM through prompting. They offer some more insights to more advanced prompting but for this research those do not apply. In this paper the researchers also discuss prompt optimization and techniques one could implement.
For our research we only looked at the most basic recommendations and followed those instructions. We manually reviewed the output of the prompt and tweaked the prompt if the output did not match our expectation. This is not a very robust technique but after reviewing many generated texts we feel that the final result is of sufficient quality.
The basics of prompt engineering involves giving clear and concise instructions, denoting a role for the LLM to take on, trying the same prompt a few times, one-shot or few-shot prompting, and lastly LLM hyperparameter configuration e.g. top-p and temperature.
We use all of the above features except for one and few-shot prompting.

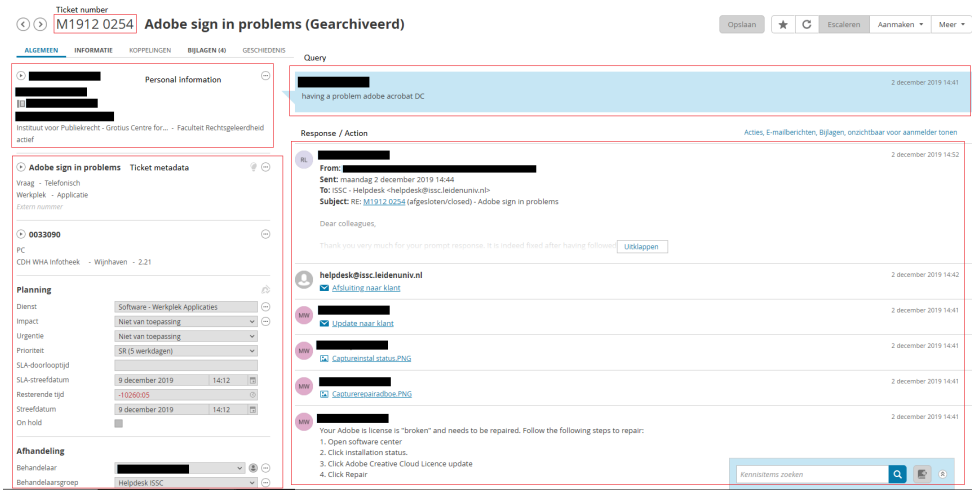

Figure 1: Example of a ticket in Topdesk. In red outlines we denote the different fields that are contained in the ticket. In total there are 5 fields: Ticket number, personal information, ticket metadata, query and the response.

3 Data set

In this section we describe the data set that we have used for testing our new method.

3.1 Topdesk

This dataset was procured for us through Leiden University. The tickets are created by users through a system called Topdesk and are of real problems that users experienced during 2021-2023. This data set has not been released to the public.
The dataset in Topdesk consists of tickets. An example of a ticket can be seen in Fig. 1. A ticket starts with a request from a user that the helpdesk will try to solve. A request has many ways of reaching the helpdesk, either through phone, mail, a form or in person through a helpdesk counter. Once a request is entered into Topdesk, one or multiple technicians will try to solve the problem. In case an issue cannot be solved based on only the request some more information is sought from the user in order to clarify their circumstances and their issue. If a helpdesk employee is unable to solve the problem they will escalate the ticket to a different department. This department solves the problem and should transfer the ticket back to the helpdesk employee. This is done because the helpdesk is the point of contact and has to translate technical solutions to a clear explanation. In practice this almost never happens. When it does happen the helpdesk employee will send an email to the user and close the ticket.

3.2 Exploratory data analysis

The dataset is a medium sized semi structured excel file. It consists of 95839 rows. Each row denotes one ticket and some metadata. The file contains the following columns: Ticket number, short description, department, status, registration date, closing date, category, evaluation, request, action, and feedback explanation. We make use of the request, action and category fields. The action field contains every message from both the user and employees in plain text which is what makes the data semi structured. Every message has a timestamp and name which is how we separate them into helpdesk and user messages.

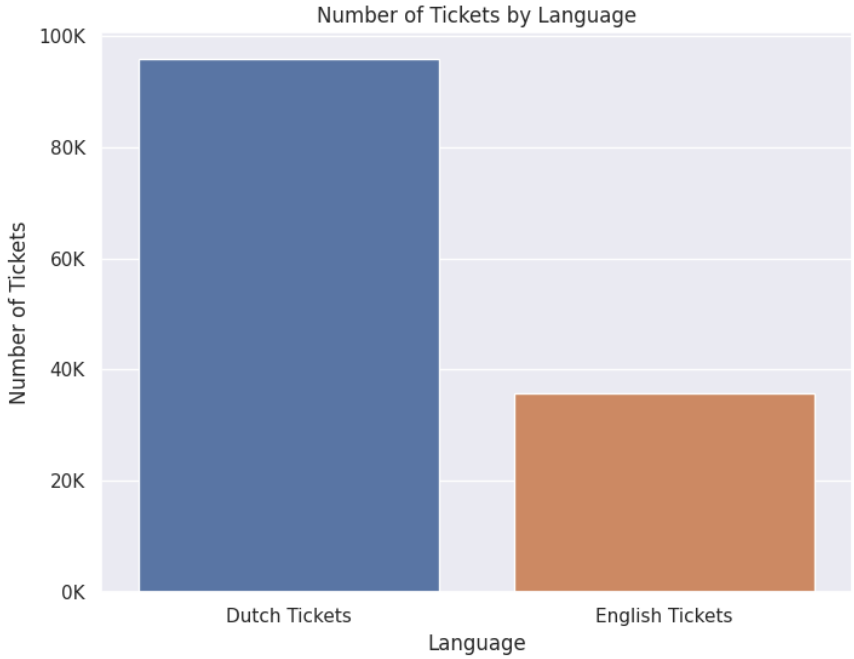

Figure 2: The distribution of total amount of tickets by language. We make a distinction between tickets that are primarily in Dutch versus English.

In Fig. 2 the distribution of Dutch versus English tickets can be found. To distinguish English and Dutch tickets we use the Python package Langdetect[2]. This package detects language of a text using naive Bayesian filter with 99% precision. This allows us to detect the language in the request and split the data by language. The dataset has about two times more Dutch tickets than English tickets. This is expected since the data comes from a Dutch university. The helpdesk also only has native Dutch speakers working for them so every English ticket is due to an English speaking user. In this research we will only regard the Dutch tickets. The following visualizations were created using the set of Dutch tickets.
Our next Figure 3 shows the length distribution of tickets in characters. As can be seen the request has less characters on average than the action field. That corresponds to the fact that the request is one message. There still is quite a large distance between the variation in the median size of a request and its quartiles which can be explained by the different ways a message is received by the helpdesk. For example, when requests come in by phone then a helpdesk employee summarizes the question of the customer as opposed to when a request comes in as an email with all the signatures and extra noise that an email comes with. The action field is on average far larger than the request field. The action field is comprised of a set of messages that form a conversation between user and helpdesk employee. In most cases a helpdesk employee will request more information from a user in order to answer their question. More information means more back and forth messaging which increases the ticket size.
In Fig. 4 we display the amount of messages it takes on average to solve a ticket. We have at least one message from the customer; this is the original request. Interesting is that there are some cases where there are no messages from the helpdesk. In this case, a call gets escalated to a different department without the helpdesk informing the end user or sending them a message.

[2]`https://pypi.org/project/langdetect/`

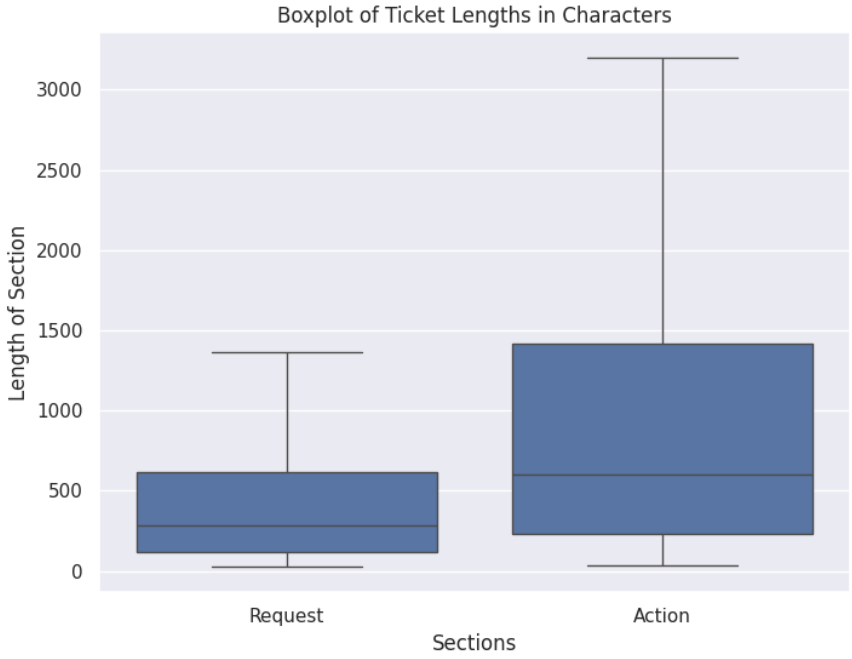

Figure 3: The distribution of length of a ticket in characters divided up by the request field and action field.

On average, there are as many messages from the helpdesk as from the user. This makes sense when you consider the tickets as a dialogue. The last response in a ticket is often an automated server message that lets the user know a ticket is being closed automatically since it has been open for too long. If a user informs the helpdesk that their solution resolved the issue, the helpdesk's final message can simply be a thank you for the update and a confirmation that the ticket is being closed.
The final Figure 5 displays the different categories of tickets that enter the system. These categories cover a wide range of problems. For example the first category: 'User accounts' might be about a forgotten password or a locked account. The dataset is quite skewed towards the top 3 problem categories. The final category Other is not a real category in Topdesk but there were many instances where there were only 100-250 or so examples of a certain category so we aggregated them in the Other category. The nature of the this system imposes duplicate question generation. Stack Overflow also has problem with duplicate questions [1]. But, in Stack Overflow duplicates get marked and the user gets referred to the original question. In Topdesk there is no such method because the users don't have access to other tickets. Therefore, we are confident that a retrieval system can match new questions to previously answered ones.

3.3 Challenges

In order to understand the challenges of our data we must first describe the general structure of a ticket. A ticket is a problem and a solution. In its simplest form a customer calling can be helped with their problem within 8 minutes. After these 8 minutes the heldesker will describe the customers problem and their solution in a problem field and an action field. This simple form describes one problem, one solution and one call. But there are multiple ways for a customer

Figure 4: The average amount of responses that one ticket has in order to get solved. There is at least 1 customer message since that is the first request that enters the Topdesk system.

to reach the Helpdesk. They can come to one of 5 Helpdesk counters, mail a question or use a form on the website. These different interaction processes make it so that a ticket will never follow the same structure. Also, when a problem comes in online then a technician's first response might not be the solution to the problem but a question asking for more information. This is because, at the university, as in many bureaucratic organizations, many issues can be resolved through administrative tasks rather than being purely technical problems.

We have identified the following challenges with our data for use in a RAG pipeline: Firstly, there is quite a significant amount of noise in the tickets. Not only does not every message contain relevant information, but some might even contain unrelated information. Then since we are directly reading emails there are some niceties as in greetings and signatures that only contain noise. This is mostly a problem for word-based retrieval model but also is an inconvenience for text generation. This problem can be summarized by the fact that the tickets in essence are conversations. Another problem is that an issue might not be resolved within a ticket. Sometimes the user ends up no longer responding if they either fixed it themselves or the problem went away by itself. This leaves the helpdesk with a ticket that will close after a period of no reaction by the user. Something else that might happen is that a user has two or more questions in one ticket. In theory a technician should create two or more tickets if this happens but in practice that is not the case. This is not an incredibly frequent event, nevertheless it does increase noise. The biggest problem is when a solution is given in the ticket but this solution is of low quality. This happens when a problem is resolved, but the solution isn't properly documented in the action field. Instead, it's communicated directly to the user via phone or email, leaving the ticket without a recorded resolution. In some cases the ticket is not solved but is closed automatically due to no reaction after a period of inactivity. When we

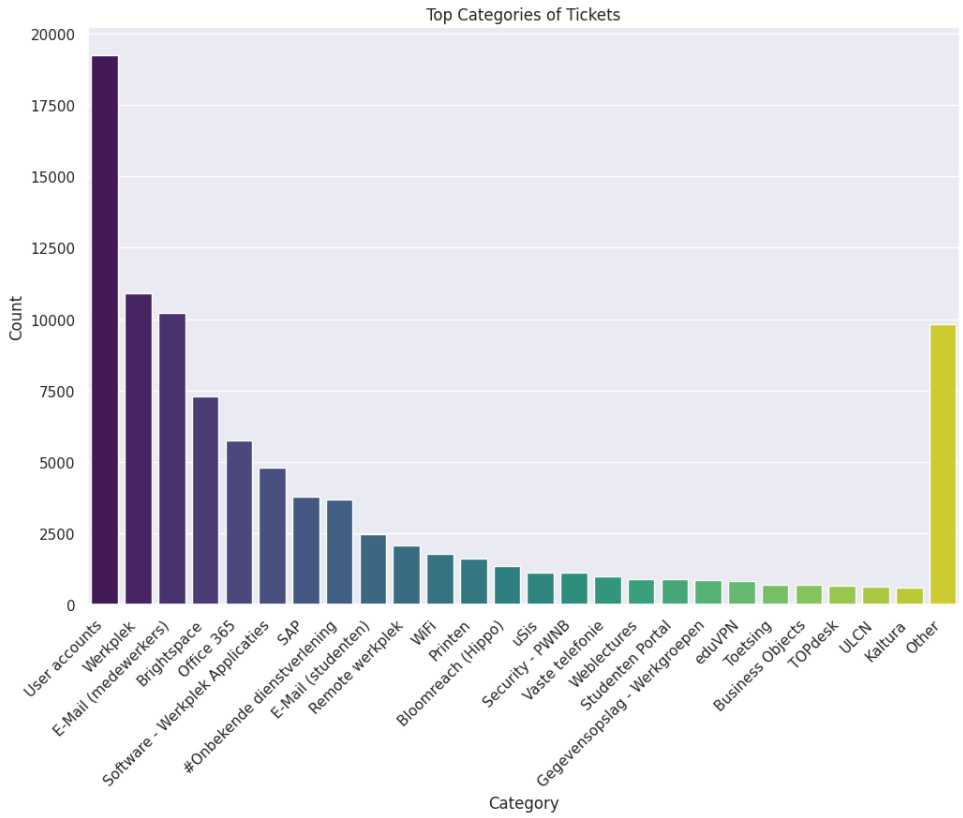

Figure 5: The count of types of tickets that come into the helpdesk.

retrieve these answers they offer no value for an LLM.
The database of old tickets was never meant to be used by an LLM for this use case so these challenges are somewhat to be expected. Especially for some questions that the helpdesk might get twice a week an expert could be fast with the handling of a ticket. This could impact the performance of our model, but in practice, if questions are sufficiently common then even a new employee would not use this tool.

4 Methods

How are we going to create a system that is capable of helping experts to answer a question that they have never seen before? First we realize that because the helpdesk employee has never seen the question, that does not mean that it has never been asked before. This assumption is the backbone of our model. We have created a RAG model that can help an expert with their questions. In this section we talk about how we have prepared our data for the sake of assisting our model, how we structured our retrieval system, and how we instruct the LLM to give properly formatted answers.

4.1 Overview of the framework

Before we dive deep into each specific section of our framework we would like to promptly display our RAG pipeline so that we have a clear overview. This is shown in Fig. 6.

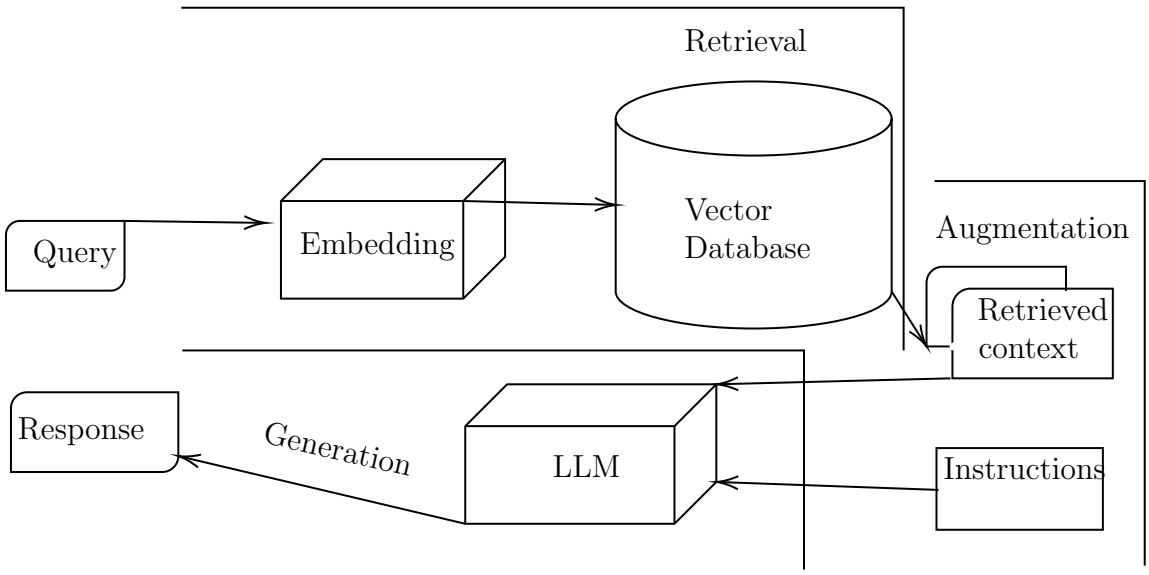

Figure 6: An overview of the RAG pipeline. We start with a query that we embed into a dense vector representation. Using FAISS we look up the nearest neighbours of this query representation. Once k documents have been found we feed it together with instructions into our LLM. The LLM generates an output that is send to the helpdesk employee.

4.2 Data Preparation

In order to solve a problem a ticket has the form of a conversation where the helpdesk employee requests more information from a user to help solve their problem. Tickets are also not constrained to a single helpdesk employee. There are situations where multiple people join the conversation to give a probable solution to the user's problem. This introduces the problem that in some cases there are two or more possible solutions in one ticket. This also happens when the helpdesk employee escalates a ticket to a different department. Escalating a ticket refers to the process of moving a customer service or technical support request to a higher level of authority or expertise. This action is taken when the initial level of support is unable to resolve the issue, or when the problem requires more specialized knowledge or if certain rights are needed to perform administrative duties e.g. giving a user admin rights on their machine. The tickets that get escalated are important because the information that is contained in this ticket is not in the internal knowledge base otherwise they would not need to be escalated. But this knowledge can help other helpdesk employees that come across the same problem and are not aware of a previous solution.

As seen in Fig. 1 there is a lot of semi structured text that needs to be modified before we data is useful to index. In the dataset there are two main columns that contain the relevant data. Column one is called 'Request'. The request field refers to the query as such we will refer to this column using query. The second important column is called 'Action'. One would expect that this refers to the answer to the query but it is not that simple. We will be referring to this field as the answer. The reason that it is not so simple is because the answer field is a conversation between a helpdesk employee and a user. So this field not only has possible answers but also more questions or other information gathering type of interactions.

The first step we take is filtering out the English tickets. Our model is trained on Dutch text and we assume better performance with Dutch text. Our next steps involve removal of personal information that can be used when querying the database. To this effect we remove names, dates,

times, and email addresses. We use regex rules that get apply to every row in the dataframe in order to remove this information. To remove names we use '\b[A-Z][a-z]+,\s[A-Z][a-z]+\b', '\b\w+,\s[A-Z]\.[A-Z]\.', and '\b\w+,\s[A-Z]\.'. These patterns correspond to the following patterns: Full last name with one initial, full last name with two initials, and full last name with full first names. To remove dates we use '\b\d{2}-\d{2}-\d{4}\b'. To remove timestamps we use '\b\d{2}:\d{2}\b'. We also remove e-mail addresses with '\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'. Lastly, we remove stop words using the `nltk` library for Dutch stop words [3].
It is almost impossible to remove every single name instance since these can be misspelled or not following the same regex so we leave some in the dataset. As for anonymization problems there are none since the data never leaves our domain. We don't use the ChatGPT API, instead we use local open-source downloaded models so the data remains in our system.

4.3 Retrieval

In this section we will discuss different retrieval methods and why we think these work best for our goals.
The importance of proper retrieval can not be understated. Without a functional context the model will never be able to create a useful response no matter the prompt or the underlying model's abilities. In this work we experiment with both sparse and dense retrieval methods. An overview of our RAG pipeline is shown in Fig 6.
We experiment with BM25 and ME5 based on question-answer pairs. This is different from our indexation strategy of the generation component for this research. We use three different indexing methods to check whether the retrieval is robust enough. The three different settings can be found in Table 1. In this setting each query has an answer so that we can calculate the precision, recall and NDCG. In contrast, our generation setting happens when the query does not have a 'correct' answer in the index but we rely on redundancy to provide us with similar questions.
What is not shown in the image is the indexation of the tickets. We discuss that process in the following sections.

Table 1: The definition of each setting for the retrieval comparison. In each setting we leave out the original question of the user.

Setting name	Setting description
Setting 1	Our index consists of every message sent by the helpdesk
Setting 2	Our index consists of every message sent by the user
Setting 3	Our index consists of the entire ticket

4.3.1 Sparse indexation

The index for our sparse retrieval methods is created by using PyTerrier [20]. Terrier is a well-known open source library for indexing large corpora. Originally Terrier was developed in Java but as of 2020 there exists a Python binding which is what we use.

[3] `https://www.nltk.org/`

PyTerrier is capable of building an index from a set of documents. It also offers an easy to use API for querying and experimentation. For sparse retrieval we use PyTerrier's implementation of BM25 [23]. BM25 estimates the relevance of a set of documents given a query. It's a word-based method using overlapping instances of words in the query and in the document. The advantage of BM25 is that it is fast and cheap to use. The drawbacks are that the context of the words is not considered when matching a query and document.

4.3.2 Dense indexation

As with sparse indexation we first create an index. We create an index by using the Facebook AI Similarity Search library (FAISS) [10]. FAISS is an efficient similarity search engine for dense vectors. We choose to use this technology since it is easy to implement and is efficient for nearest neighbour search. This also enables us to choose the best dense retriever without a lot of effort.
The dense retrieval models we experiment with is Multilingual-E5-large (ME5) [27] . We implement this models using the beir python package [11].

4.3.3 Indexing and query strategy for generation

In most RAG applications the retrieval is based on question answer similarity. That approach might seem appropriate in this case. But it is not, we want to simulate what generation would look like in a real word application. In the real world when a new question comes in you can not be sure that the answer is in the database. Therefor, we decided to index 90% of our tickets. The other 10% is what we use to query the database. This 10% of tickets represents new tickets that have not been solved. This way we take out all the ground truth answers and rely on previously solved tickets to be used as context for the LLM. The part of the ticket that we index is the first user message which corresponds to the request field discussed earlier. We query the database based on this field as well but then with the non-indexed queries.
We call this task query-query retrieval. In the previous section we describe a question-answer retrieval method. But for the generation we decided to do a query-query based similarity search. We are not forced into this choice since we can still find similar answers to questions that are not in the database but we assume that when looking for similar redundant questions we get a more accurate representation of the solution set. Secondly, we consider the tickets to be documents instead of question-answer pairs. When we do RAG for open domain question answering the documents considered are usually part of a knowledge base like Wikipedia. These are not question-answer pairs instead the LLM tries to look for relevant passages in the Wikipedia page or other information source.
We build this model to be used by the helpdesk employees so that they can put the entire user query in the search bar and get a valid answer. This is not a traditional search where keywords are the main search component. Keywords could still be used but that is not what the system was designed for.
Since we are turning our queries into a dense vector representation and then searching through that space by doing a nearest neighbour search it seems that queries that are close to each other have the same semantic meaning. If we were to index the answers to the question then these are further away from each other in the search space. The solution for this is to use specific pre-trained query answer models but there were not a lot of those for the Dutch language.
Once we have found queries that are similar to our query, we extract the entire document from a secondary database that holds the content of the entire ticket.

System prompt

Je bent een hulpsysteem voor een medewerker van de ICT hulplijn van de universiteit Leiden.
Je krijgt zometeen 1 nieuwe vraag te zien en k dialogen waarin de vraag die gesteld is lijkt op de vraag die jij zou moet beantwoorden. Probeer relevante informatie uit die k dialogen te halen. Niet elke dialoog is even relevant. De k dialogen zijn opgehaald met een ophaal algoritme en zijn niet allemaal even relevant voor de nieuwe vraag.
Jou taak is om de nieuwe vraag te beantwoorden met de mogelijk relevante antwoorden op de k eerdere vragen.
Deze stappen moet je nemen:

1. Lees de nieuwe vraag
2. Lees de oude k dialogen
3. Bepaal of in de oude antwoorden relevante informatie zit
4. Genereer een antwoord op de nieuwe vraag.

Figure 7: Example of the RAG pipeline system prompt. English translation in Fig. 10

4.4 Model

The LLM model that we use is called Geitje 7B ultra[4]. This is a Dutch instruction-tuned model based on Mistral 7B. It was fine-tuned on Dutch data and a synthetic dataset created by gpt-4-turbo.

4.5 Text Generation

Once we have created our index of queries, embedded the new query and retrieved the relevant documents it is time to generate an answer to the new query. In this phase the model uses the retrieved context together with a system prompt to create a sufficiently accurate output. Fig. 7 shows there is an example of the system prompt that we use to instruct our model. We instructed the model to read the content carefully and decide for itself whether a passage in the ticket is relevant to the problem or not. The second part of the instructions were the instructions again but this time in a list to make sure that the model behaves consistently. In the hyperparameters for text generation we did leave some room for exploration so no two runs are exactly the same.

4.5.1 Summarization

We notice that Geitje struggles with the noise in the context. So we decided to cut some of the noise out. We do this by summarizing the tickets. Summarization is the process of extracting the most relevant parts of a piece of text. There are multiple ways of generating summaries. LLM's have proven to be quite adept at this task [9]. There are also other tools to create

[4] `https://huggingface.co/BramVanroy/GEITje-7B-ultra`

System prompt

Je bent een hulpsysteem voor de IT Helpdesk van de Universiteit Leiden. Medewerkers van de IT Helpdesk moeten gebruikers helpen met het beantwoorden van hun vragen. Jij moet de medewerkers helpen met het beantwoorden van de vragen. Je moet dus niet direct antwoord geven op de vraag die wordt gesteld. Let op Je krijgt eerst wat context te zien die bij de vraag hoort, lees deze context goed hier zitten soms handige tips in voor het beantwoorden van de daaropvolgende vraag. Nu wat meer informatie over het format van de vragen.
Je krijgt zometeen 2 dingen te zien:
Het eerste wat je te zien krijgt is een samenvatting van een oude vraag die lijkt op de nieuwe vraag. Deze samenvatting is mogelijk relevant voor het nieuwe probleem.
Het tweede dat je te zien krijgt is de nieuwe vraag van de klant. Probeer deze vraag te beantwoorden aan de hand van de samenvatting die hierboven staat. Als de samenvatting niet relevant is voor deze vraag geef dat aan en probeer zelf een plausibel antwoord te geven.
Instructies voor het construeren van je antwoord:

- Negeer niet relevante tickets
- Geef geen algemene overzicht van de vraag en het probleem
- Ga direct naar de stappen die de helpdesk medewerker kan proberen bij de klant om het probleem op te lossen

Figure 8: Example of the RAG pipeline system prompt where we substitute the ticket for a summarization of the ticket. English translation in Fig. 8

summaries like Text-To-Text Transfer Transformer (T5) [21]. We experimented with these two techniques and found similar results and ended up choosing LLM generated summaries done by Geitje.
Once we have created the summary we do the same process as in the original RAG pipeline but replace the noisy context with clear summarized ones. We do transform the system prompt in order to deal with this, see Fig. 8. Summarization has the added benefit of shortening the original text.

4.6 Evaluation

There are many ways of evaluating a model's performance. But, not every metric is relevant. Here we discuss which metrics we use to evaluate the quality of our pipeline and what they tell us about the strength of our prompts in combination with the retrieval context. There are statistical scorers and model-based scorers. These metrics are used in conjunction in order to judge a model's strength [4]. We use a mixture of model-based and statistical based scorers to evaluate our model.

4.6.1 Statistical evaluation

In the statistical scorer family we have a choice between BLUE, ROUGE, METEOR and Levenshtein distance. We briefly introduce these metrics.

- BLEU (BiLingual Evaluation Understudy) scorer compares the output of the model to either annotated ground truths or expected output. It does this by calculating the precision for each matching n-gram of the generated content versus the expected output and calculates the geometric mean, applying a brevity penalty if needed.
- ROUGE scorer has primarily been used to judge text summaries. ROUGE calculates the recall of the overlap between the generated text and expected outputs. It returns a proportion between 0-1 of n-grams in the reference that are present in the model's output.
- METEOR (Metric for Evaluation of Translation with Explicit Ordering) scorer is more extensive since it assesses both n-gram overlap (recall) and and n-gram matches (precision), also adjusts for word order differences between the outputs. Furthermore, this technique uses an external database to account for synonyms. We end up with a final score that is the harmonic mean between the precision and recall otherwise known as the F1 score.
- Levenshtein distance scorer calculates the minimum number of single character edits required to change on word into another. This measure is mostly used to evaluate spelling corrections.

Based on these brief description and similarities between the metrics we choose to use BLEU and ROUGE as our statistical evaluation methods.
The drawback of statistical evaluation is that calculating the overlap between the LLM response and the ground truth does not say much about the relevancy of the answer. The overlap could be zero but the reaction could still be correct. This is even more the case when we use summarizations of the answer because the LLM will be less likely to repeat parts of the ticket's original answer in its response. A different problem with statistical methods and this data set is that we are not sure whether the correct answer was given in the ground truth. We already mentioned that the tickets contain a lot of noise, so does the ground truth. The ground truth is made up of the set of helpdesk response to the original question. That includes information gathering responses and other noise that might be introduced in a conversation. This noise will also negatively impact statistical evaluation.

4.6.2 Model evaluation

There are two ways in which we evaluate our results by using a neural model. The first is called BERTScore [30]. The second is method uses a the power of a different model than Geitje to evaluate the answer given by Geitje, this method is called G-Eval [19].

BERTScore is used as a metric to automatically evaluate generated text. For each token in the generated text BERTScore calculates a similarity score for each token in the candidate text. Different from statistical methods BERTScore uses contextual embeddings. This means that the words don't need to exactly overlap in order to be considered correct but their contextual

System prompt

Je bent een taalmodel dat een ander taalmodel zijn antwoorden moet evalueren. Je krijgt 3 dingen te zien: De vraag van de klant, de groundtruth en een gegeneerde text. Nu is het jouw taak om te bepalen of de gegenereerde text relevant is gebaseerd op de vraag gegeven de ground truth. Geef een relevantie score van 0-2. Geef alleen integers dus geef een score van 0, 1 of 2. 0 betekent: de gegeneerde text is niet relevant vergeleken met de ground truth, 1 betekent: de gegenereerde text komt enigzins overeen met de ground truth,, 2 betekent de gegenereerde text komt volledig overeen met de ground truth. Geef de output in JSON format. Een voorbeeld hier van is: r̈elevantie¨: s̈core¨: Geef ALLEEN de output als score en geen verdere redenering.

Figure 9: G-Eval system prompt. English translation in Fig. 12

interpretations have to be similar. The calculated embeddings are of significance so we cannot use any BERT model. We use a BERT model that is specifically trained on the Dutch language.

G-Eval judges the generated text based on a prompt without the need for a reference. See the prompt in Fig. 9. G-Eval is an LLM evaluating the output of another LLM. We do not need a reference text for this to work. For example when given a candidate string the prompt could be 'Is this answer relevant to the question ...' and then give a certain context. In our case we do have reference texts so we use it like BERTScore where it judges the contextual overlap between the candidate and the reference answer. In order to implement G-Eval we use the Confident AI library[5]. The model we choose to evaluate Geitje is Llama 2 13B.

4.6.3 Manual evaluation

Our third evaluation type is manual evaluation done by us. This is only done on our best performing setting. Manually evaluating sub-optimal settings is impractical. We evaluate manually because a humans interpret answers differently than a model. Also, even though an answer might seem correct to a model using G-Eval it might not be.
G-Eval uses coherence to score their input targets. In manual evaluation we score the generated output by relevance to the ground truth and relevance to a helpdesk employee. I work for the helpdesk and can accurately say if a generated answer is relevant and will help solve the user question. The problem is that there is only one opinion in this part of the evaluation. This might skew the observations to be biased according to my knowledge. I try to be aware of this bias and score according to a specific criteria: if I were a new employee would the answer help me in assisting the user.

5 Results

In this section we show the results of our retrieval methods and analyze the difference between zero shot generation and retrieval-augmented generation. Furthermore, we investigate different

[5] https://github.com/confident-ai/deepeval

prompting strategies. Lastly, we manually evaluate 100 generated outputs to identify potential issues and determine what the bigger bottleneck is between the retrieval and generation.

5.1 Retrieval

Our generation can only be as good as our retrieval since that is where we find our correct answers. We study the differences between retrievers and indexation strategies extensively. We compare a sparse retriever versus a dense retriever in different settings. These results only give an indication of how well our actual retrieval works with regard to the query-query retrieval strategy. We can evaluate the retrieval in this setting since each query has an answer. It is not possible to do this for the case where we generate outputs using Geitje since in that setting we remove the answer from the dataset. Also, in that setting we use a query to index a database of queries and choose the highest ranking similar queries.
Our results for retrieval using different settings are shown in Table 2. In Table 1 there are the explanations for each setting. We use three different indexation strategies for our experiments with the retrievers for a few reasons. Firstly, we want to know if the dataset is suitable to be queried. Our assumption is that there is a high level of redundancy in the dataset since multiple people will likely come across the same problem.
We also want to know how much information we need to embed in order to find the right answer for a ticket. We treat every message as an answer even though some messages do not contribute directly to an answer but are simply there for the purpose of gathering information about the problem. We cannot say that the last message is the solution because the last message can also be a concluding message (closing the thread) that does not contain the solution.
All the above mentioned assumptions and facts result in that we test three different settings. For setting 1 we index the messages sent by the helpdesk ('the answer'). In setting 2 we index the messages sent by the user ('the question'). In the final setting we index the entire ticket. For all these setting we do not include the original question since that is what we use for querying.

Table 2: Retrieval algorithm results based on different settings. We run each setting twice, once for the sparse retriever and once for the dense retriever.

Dataset	Setting	Retriever	Precision@1	Recall@5	NDCG@5
Tickets	Setting 1	BM25	0.11	0.17	0.14
Tickets	Setting 1	multilingual-e5-large	0.17	0.29	0.23
Tickets	Setting 2	BM25	0.17	0.28	0.23
Tickets	Setting 2	multilingual-e5-large	0.31	0.44	0.34
Tickets	Setting 3	BM25	0.15	0.23	0.19
Tickets	Setting 3	multilingual-e5-large	0.21	0.32	0.27

We measure the precision of the first retrieved document, the recall@5 and the normalized discounted cumulative gain of the first 5 documents. The most important metric is recall@5 since these retrieved documents will form the context based on which the LLM is going to generate answers. If there is no correct answer in the first five documents then essentially we're

hoping that the LLM's latent knowledge has the answer already stored but that is unlikely. If that were the case then retrieval-augmented generation would provide no added benefit.
The recall at five statistic is a bit misleading since it is possible that although we haven't found the correct document containing the answer for the ticket, it could be possible that there are other acceptable answers within the retrieved documents. But, this is not something we can measure.
It is immediately clear from Table 2 that a dense retriever outperforms the sparse retriever. Setting 2 where we index the users' messages performs the best overall. We think this might be due to personal writing styles. Every user is different and has their personal way of describing a problem. Their vocabulary matches better to the query. The embeddings are based on the output of a black box so it is difficult to interpret why it responds better to certain data. We also see that when adding the helpdesk employees' messages to the index the performance degrades. This could be due to an overload of data making the embedding less precise. In this case the helpdesk employees' messages are a source of noise.
The experiments in section 5.2 are based on a query-query retrieval with multilingual-e5-large as retriever.

5.2 Retrieval-augmented generation

5.2.1 Automatic evaluation

Table 3 shows results for RAG when appending complete tickets to the RAG context. We expected that adding more tickets to the input will result in higher ROUGE, BLEU and Bert scores. Based on our results this does not seem to be the case. For all four settings we find no significant difference between the automated evaluations. This does not mean that the generated text is similar.
In order to evaluate our text generation we first compare the generated output compared to the reference texts. We use the standard evaluation metrics BLEU, ROUGE, and BERTScore. BLEU measures the n-gram overlap between generated and reference sentences. ROUGE measures the recall between output and reference text. ROUGE gives higher scores when the candidate and reference text are similar in content but not in word order. Lastly, BERTScore leverages a pre-trained embedding model to calculate the contextual relevancy of candidate and reference texts.
The results in Table 3 show the results for our retrieval-augmented generation pipeline when the tickets are not summarized. The setting of k denote the different amount of tickets we add to the context of what we stream to Geitje. The k equals zero setting means that we only give the user question to Geitje without any additional context.
In Table 4 we have the results for RAG using summarized tickets as context. The results are similar to the results in Table 3. We do not see a gain in performance when we increase the amount of tickets.
In order to investigate further why these metrics do not seem to improve when adding extra tickets we employ two more evaluation metrics.

5.2.2 G-Eval

There are multiple metrics we can choose to evaluate using G-Eval but the most relevant metric is relevance. We want to know whether the generated text is relevant compared to the ground truth. The ground truth is every message that a helpdesk employee has send in a ticket.

Table 3: Retrieval-augmented generation results based on the unchanged ticket. In this example we do not change the content of the ticket, we only remove the identifiable noise such as time, dates and helpdesk employees names

Dataset	k	Rouge1	Rouge2	RougeL	Bleu	BertP	BertR	BertF1
Tickets	0	0.17	0.02	0.10	0.02	0.61	0.62	0.61
Tickets	1	0.18	0.03	0.10	0.02	0.61	0.64	0.62
Tickets	3	0.19	0.03	0.11	0.01	0.60	0.64	0.62
Tickets	5	0.17	0.02	0.10	0.02	0.61	0.64	0.62

Table 4: Retrieval-augmented generation results based on summary of the ticket

Dataset	k	Rouge1	Rouge2	RougeL	Bleu	BertP	BertR	BertF1
Tickets	1	0.16	0.02	0.09	0.02	0.61	0.63	0.62
Tickets	3	0.17	0.02	0.10	0.02	0.61	0.63	0.62
Tickets	5	0.17	0.02	0.09	0.02	0.60	0.63	0.62

We compare the ground truth to the generated text and instruct the model to give us a score based on its perceived relevance.
It is the models' responsibility to find the corresponding statements in the generated text compared to the ground truth. This is not an easy task. It is not easy since relevance is a nuanced concept. Also, Geitje and Llama have no latent knowledge on the IT tasks that are administrative in nature so the generated text is almost never relevant for questions of that nature.
The model that we use to evaluate the relevance is Llama 2 13B[6]. We have chosen this model since it is larger than Geitje so it should be able to reason better. There is also a 70B model available but due to computational constraints we can not use that model here. We prompt the model similarly as in the G-Eval paper. We give it a role, a task, a metric for evaluation and what the different scores mean. The meaning of the scores we want Llama to assign to generated texts are in Table 5.

Table 5: The meaning we give to the scores that G-Eval gives. We use these texts in our system prompt that we use to instruct Llama.

Score	Meaning
0	The generated answer is not relevant to the ground truth
1	The generated answer is slightly relevant to the ground truth
2	The generated answer is completely relevant to the ground truth.

As shown in Table 6 the highest G-Eval score can be found for a value of k is 5. That shows that added context in the prompt increases our relevance score. This can be explained by

[6] `https://huggingface.co/meta-llama/Llama-2-13b-hf`

Table 6: G-Eval evaluation on retrieval-augmented generation output.

Dataset	k	Average G-Eval score	Standard deviation
Tickets	1	1.20	0.42
Tickets	3	1.26	0.46
Tickets	5	1.34	0.49

the higher probability that a correct answer is found in the context so this is reflected in the generated text. Our thesis is build on this assumption that when the correct answer is present in the context that Geitje is smart enough to recognize this and that it is reflected in the output generation. However, we can not say this is the case without inspecting the generations manually.
When delving deeper into the results produced by G-Eval we notice something interesting. G-Eval almost never gives a score of 0. G-Eval almost never deems an answer to be irrelevant. When looking through the answers this is due to every answer being somewhat plausible if the system has no specific domain knowledge. Most questions could have appeared on an open forum somewhere and ended up in Geitje's training data.

5.2.3 Manual evaluation

The manual evaluation is done on the best performing setting. G-Eval on k equals 5 gives a score of 1.34 so this is the setting that we manually evaluate. Also, we compare the differences between the settings by showing a few examples. The score in Table 7 is calculated after manually reviewing 100 randomly chosen samples.

Table 7: Manual evaluations of generated text and ground truth. The first column denotes the similarity between the ground truth and the generated text. Kendall Tau correlation is between G-eval and manual evaluation. Lastly, we manually evaluate the actual utility of the generated text for a helpdesk employee

Dataset	k	Similarity to ground truth	Kendall Tau correlation	Utility
Tickets	5	0.86	0.05	1.02

There is a very weak correlation between the set of data evaluated by G-Eval and the set of manually evaluations. This is caused by the fact that G-Eval often gives a high score of 2 even though it's actually a 1. This shows that G-Eval does not understand when an answer is completely relevant in this domain.
In order to inspect the manual evaluations we show a few examples. For the first ticket we look at a problem where an employee's software center wasn't loaded with any software. Software center is the place where employee's can download managed software provided by the university. This problem arises on laptops or computers when it has had a fresh reinstall and hasn't been connected to the university's network for long enough. Here is the employee's original question:

> *Software center is op locatie leeg.*

Here is what Geitje said based on 5 previous tickets that were retrieved and summarized:

> *Om het probleem van een leeg software center op locatie aan te pakken, raad ik aan om de volgende stappen te volgen:*
> *1. Controleer of de software center correct is ingesteld op uw netwerkverbinding door te controleren of de netwerkinstellingen correct zijn ingesteld op 'VUW.leidenuniv.nl'.*
> *2. Als de netwerkinstellingen correct zijn, controleer dan of de software center zelf up-to-date is en of er geen updates of patches nodig zijn.*
> *3. Als de software center nog steeds leeg is, probeer dan de volgende commando's uit te voeren in de command line interface (CLI) van uw systeem: - 'gpupdate /force' om software te updaten die niet standaard is geïnstalleerd. - 'run now' gevolgd door de naam van de software die u wilt installeren, om de installatie te starten.*
> *4. Als de software center nog steeds leeg is, neem dan contact op met de IT helpdesk*

This ticket probably came in through phone and so the helpdesk employee summarized the question that Geitje is trying to answer. The helpdesks' response was that the employee has a lot of updates in software center and that they showed the employee how to install these from their home.
G-Eval evaluated this response with a 1. That implies that the response is slightly relevant to the ground truth. We do not agree since none of these options are relevant based on the ground truth.
To understand why Geitje generated this response we look at the given context from which it had to generate an answer. The context given is quite interesting since together it does form a complete picture of why the software is not available. In one ticket there is mention of the fact that the user has to be connected to the 'NUWD' network. This is true, in order to install university software one has to be connected to this specific network. An interesting tidbit is that Geitje hallucinates what the abbreviation means and generates: 'Network Use Data (NUWD)'. A different ticket mentions the use of configuration managers ability to run a specific program that can help solve the issue. This is the correct answer in most cases. Geitje generated this in point 3 but doesn't understand the context from where a user has to use 'run now'.
In this case the retrieval was excellent and the generated answer lackluster. The provided summaries do offer enough information for a helpdesk employee to solve the question but Geitje was not quite able to deduce the relevant answers. This could point to the fact that Geitje does not understand which answers are most relevant and sees every answer as probable therefore incorporating each ticket in the answer.

Our second question involves a Citrix problem. Citrix is virtualization software that allows employees to access their desktop environment or applications from home. These tickets were quite relevant in the period of our data set because of the pandemic. The question from the employee was:

> *Hangende Citrix sessie* [7]

The proposed respone by Geitje is:

[7] Slightly paraphrased since the original message also includes the employees username.

Om de hangende Citrix-sessie succesvol af te sluiten, adviseer ik de volgende stappen:
1. Controleer of de Citrix-applicatie nog actief is door te kijken of het Citrix-icoon in de taakbalk of systeemvak aanwezig is.
2. Als het icoon aanwezig is, klik erop om de applicatie te sluiten.
3. Als de applicatie niet direct sluit, kunt u proberen de applicatie geforceerd te sluiten door de toetsencombinatie `Ctrl + Alt + Del` te gebruiken.
4. Als de applicatie nog steeds niet sluit, probeer dan de applicatie te sluiten via de Taakbeheer (Task Manager) in Windows.
5. Als u de applicatie niet kunt vinden of sluiten, kunt u proberen de applicatie te verwijderen uit de lijst met actieve programma's (Task Manager) en deze opnieuw te installeren.

G-Eval gives the answer a score of 2. The ground truth in this case is only that the helpdesk employees closed the citrix session so we would give it a score of 0. the correct answer would be that the helpdesk employee has to end the Citrix session using a specific program. This procedure is described in the tickets given as context.
So for this example Geitje did not come up with the correct answer even though the correct answer was present in the provided context. Furthermore, each ticket in the context was relatively similar so we expect that Geitje would pick up on this and realize that when the answers are repeated it must be the correct answer.
In our third example we will show an example of a correctly generated answer. The question here is about SAP Selfservice. A tool employees can use for HR related tasks. This tool has to be opened in a remote Citrix environment and can cause issues. The question is:

Mevrouw kan SAP Self Service niet openen op haar computer. Geadviseerd om de cookies en cache te verwijderen volgens de bijlage.

In this example the second sentence gives the answer to the question. The answer should not have been added to this field since it is reserved for the user question. This could intefere with Geitje's use of the context even though the proposed solution is correct. The helpdesk also sent the customer a PDF file with instructions on how to clear the cache and cookies.
Here is the generated response by Geitje:

Om Mevrouw te helpen bij het openen van SAP Self Service op haar computer, is het belangrijk om te controleren of de cookies en cache correct zijn verwijderd. Dit kan vaak helpen bij het oplossen van problemen met de toegang tot de applicatie. Hier zijn de stappen die u kunt adviseren:
*1. **Controleer de browserinstellingen**: Ga naar de instellingen van de browser die Mevrouw gebruikt (meestal te vinden onder 'Instellingen' of 'Opties' in de browser). Zoek naar 'Privacy' of 'Cookies en andere sitegegevens' en kies voor 'Alle cookies verwijderen' of 'Alle sitegegevens verwijderen'.*
*2. **Cache wissen**: Ga naar de instellingen van de browser en zoek naar 'Cache' of 'Browsegegevens'. Kies voor 'Cache wissen' of 'Browsegegevens wissen'.*
*3. **Herstart de browser**: Sluit alle openstaande SAP Selfservice*

Geitje cuts off its response in item 3. We have tried to instruct the LLM to only give the needed steps in order to solve the problem but sometimes it will summarize the issue first which takes

up tokens. In the given context there was one other solution posed to the problem and that is to use the Edge browser instead of Google Chrome. This is not present in the above mentioned solutions but could have been added had Geitje been instructed to generate more tokens.
The other documents in the context mention the clearing of cache and cookies to solve the problem. There are also some things in the context that Geitje doesn't include in the answer. For instance there are two mentions of going to SAP Selfservice through the University's website and not going to the Citrix environment.

6 Discussion

The results indicate that Geitje is able to output plausible answers but its performance does not fluctuate much based on the size of the context. The retrieval works decently and is supported by the redundancy in the data set.
The results suggest that Geitje is somewhat able to answer helpdesk questions using previously resolved tickets. We notice that Geitje focuses too much on the actual question when generating an answer. The last example in the result section shows that well. In that case an answer was given in the question section of the ticket and Geitje only reflected this answer in its output. However, the context also had a plausible solution.
Furthermore, it seems that Geitje ignores most of the given context even though the context is relevant. We think this is due to the fact that Geitje is unable to understand the correct solutions given in previous tickets. Geitje should make the comparison between the question and every ticket that is given in the context but since we first supply the context and then the ticket the model might get too focused on just the question.
Although, we do have examples where Geitje uses the context to answer questions so we can not claim that the context is never used.
Contrary to the hypothesized assumption the generated answers can answer a few simple questions but struggles when it does not understand the given context. This is different compared to open domain question answering [15]. We think this is due to the complexity of the question and possible amount of answers. In open domain question answering there is usually a single correct answer. In our case there can be multiple solutions that have to be tried out before arriving at the correct solution. This is why knowing what previous helpdesk employees have done is so valuable because you get a range of options.

6.1 Recommendation for Topdesk

Topdesk is trying to find a natural path for AI tools to be used in their customers' workflow. As of this moment they have a few tools that can help the helpdesk employees by summarizing a ticket, translating a knowledge item to English and generating an email. These functions make the life of a helpdesk employee easier. However, the end goal should be to solve tickets faster. For this to happen some sort of RAG has to be implemented. We noticed that the KB contained only a few hundred items was not complete enough to answer every question users submit. The problem lies in obscure questions and retrieving the correct context for such questions.
Even though our implementation lacks precision, we still think it is valuable to implement some parts of RAG. The first step of RAG is retrieval as of right now the retrieval in Topdesk is imprecise. It would be a good first step to increase retrieval performance. Once this is done the

top 5 retrieved tickets can be summarized and shown to a helpdesk employee. The employee can judge for themself if there is a relevant answer in these previous tickets. This method filters out the hallucinations of a model.
So, we think the techniques are valuable but not deployable as of right now.

7 Conclusion

While the importance of chatbots becomes ever increasing, the possible applications are not simple to implement. This paper explores the use of RAG in solving new IT helpdesk questions. We have implemented a RAG system where we index old queries so when an new query comes in we search for similar queries among the old indexed questions. Afterwards, we pre-process these tickets by summarizing them in order to remove noise. These tickets become the context that gets appended to the new query and is given to Geitje. The LLM then generates an output based on a system prompt, the new question, and the proposed context.
We notice that the automatic evaluation does not seem to change much based on the amount of context given to the model. We do see a change in evaluations when we use G-Eval with Llama13B. In that case, adding more context to the prompt improves performance. We also manually evaluate 100 of generated outputs based on the best performing G-Eval setting. These manual evaluations provide most of the insight to the performance of the model. We notice two things. The model focuses too much on the questions and the model does not incorporate the context correctly. If this is due to the power of the model is hard to say. We think it is due to the complexity of the question. Another reason is that the model does not understand the environment in which it is operating so does not understand when relevant answers are actually correct and should be repeated in the output.
To solve these problems we can think of a few recommendations. Firstly, use a more powerful model. We used Geitje because it is a Dutch trained LLM, but also to prove the concept. The problem is that it does not always seem to recognize the right solution in the context. This is a problem that other models might not have since bigger is usually better [12]. We could create a prompt that would force the model to consider each individual ticket more closely. In our current method we append the new question after the context. There could be a method that forces the model to consider the context more directly. Lastly, focusing on retrieving quality tickets instead and re-ranking tickets we seem as more informative could increase performance.

References

[1] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. Mining duplicate questions in stack overflow. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, page 402–412, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Martin Balfroid, Benoit Vanderose, and Xavier Devroey. Towards llm-generated code tours for onboarding. In *2024 ACM/IEEE International Workshop on NL-based Software Engineering (NLBSE '24)*, United States, April 2024. ACM Press. 3rd Intl. Workshop on NL-based Software Engineering, NLBSE '24 ; Conference date: 20-04-2024 Through 20-04-2024.

[3] Erik Brynjolfsson, Danielle Li, and Lindsey R Raymond. Generative ai at work. Working Paper 31161, National Bureau of Economic Research, April 2023.

[4] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3), mar 2024.

[5] Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. Unleashing the potential of prompt engineering in large language models: a comprehensive review, 2024.

[6] Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. Search-based neural structured learning for sequential question answering. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[7] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[8] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation, 2023.

[9] Hanlei Jin, Yang Zhang, Dan Meng, Jun Wang, and Jinghua Tan. A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods, 2024.

[10] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[11] Ehsan Kamalloo, Nandan Thakur, Carlos Lassance, Xueguang Ma, Jheng-Hong Yang, and Jimmy Lin. Resources for brewing beir: Reproducible reference models and an official leaderboard, 2023.

[12] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

[13] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering, 2020.

[14] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in*

Neural Information Processing Systems, volume 33, pages 9459–9474. Curran Associates, Inc., 2020.

[16] Shuo Li, Sangdon Park, Insup Lee, and Osbert Bastani. Trac: Trustworthy retrieval augmented chatbot, 2023.

[17] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

[18] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[19] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment, 2023.

[20] Craig Macdonald and Nicola Tonellotto. Declarative experimentation ininformation retrieval using pyterrier. In *Proceedings of ICTIR 2020*, 2020.

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[22] Edwin Rijgersberg and Bob Lucassen. Geitje: een groot open nederlands taalmodel, December 2023.

[23] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[24] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation, 2021.

[25] Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering. *Transactions of the Association for Computational Linguistics*, 11:1–17, 01 2023.

[26] Calvin Wang. Potential for gpt technology to optimize future clinical decision-making using retrieval-augmented generation — springerlink. `https://link.springer.com/article/10.1007/s10439-023-03327-6`, 2023. (Accessed on 09/18/2023).

[27] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*, 2024.

System prompt

You are an assistance system for an employee at the IT helpdesk of the Leiden University.
You will get to see 1 new question and k conversations between a user and a helpdesk employee. These conversations resemble the question that you will have to answer. Try to retrieve the relevant information from the conversations. Not every dialogue is relevant to the question.
Your job is to answer the new question with the help of the information in the relevant k conversations.
these are the steps you have to take:

1. Read the new question
2. Read the old k conversations
3. Determine whether there is relevant information in the old conversations
4. Generate an answer to the new question

Figure 10: Example of the RAG pipeline system prompt.

[28] Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, and Zheng Li. Retrieval-augmented generation with knowledge graphs for customer service question answering. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2905–2909, New York, NY, USA, 2024. Association for Computing Machinery.

[29] Kai Zhang, Wei Wu, Haocheng Wu, Zhoujun Li, and Ming Zhou. Question retrieval with high quality answers in community question answering. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, page 371–380, New York, NY, USA, 2014. Association for Computing Machinery.

[30] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020.

[31] Pengfei Zhu, Zhuosheng Zhang, Jiangtong Li, Yafang Huang, and Hai Zhao. Lingke: A fine-grained multi-turn chatbot for customer service, 2018.

A Prompts

System prompt

You are an assistance system for an employee at the IT helpdesk of the Leiden University. Helpdesk employees have to help users to answer their IT related questions. It is your job to help the employees answer the users' question. Don't directly supply an answer for the question but provide the steps a helpdesk employee has to take in order to figure out what is going wrong. You first get to see some context that could help you to answer the question. Read this context well it could contain the answer that the user is looking for.
You will get to see two pieces of text. The first is a summary of previously solved tickets that are similar to the new question. These summaries might be relevant.
The second thing is the new user question. Try to answer this question using the relevant summaries that we provide you with. If none of the summaries are relevant then try to give a plausible answer yourself.
Instructions for the formatting of your answer:

- Ignore irrelevant tickets
- Do not give a general overview of the problem
- Directly provide the helpdesk employee with the steps necessary to answer the users' question.

Figure 11: Example of the RAG pipeline system prompt where we substitute the ticket for a summarization of the ticket.

System prompt

You are a language model that is going to evaluate another language model its generated answers. You are going to see 3 pieces of text: A user question, the ground truth and a generated answer. It's your job to determine whether the generated text is relevant to the user question based on the ground truth. Give a relevance score between 0-2. Only give integers so 0, 1 or 2. 0 means that the generated text is not relevant to the ground truth. 1 means that the generated text is somewhat relevant to the ground truth. 2 means that generated text is completely in line with the ground truth. Give the output is JSON format. An example is: r̈elevance: s̈core: Only give this output and no further reasoning.

Figure 12: G-Eval system prompt.