Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Enhancement of RAG chatbots performance using various heuristics

Bachelor Thesis

2025
Teo Pazera

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Enhancement of RAG chatbots performance using various heuristics

Bachelor Thesis

Study Programme:	Data Science
Field of Study:	Computer Science and Mathematics
Department:	Department of Computer Science
Supervisor:	Mgr. Michal Uherek

Bratislava, 2025
Teo Pazera

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Teo Pazera
Študijný program: dátová veda (Medziodborové štúdium, bakalársky I. st., denná forma)
Študijné odbory: informatika
matematika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Enhancement of RAG chatbots performance using various heuristics
Vylepšenie výkonu RAG chatbotov za použitia rôznych heuristík

Anotácia: Cieľom práce je ukázať a otestovať rôzne spôsoby, ako zvýšiť pravdepodobnosť správnych odpovedí modelu na danú množinu otázok, ktorých odpovede by sa mali nachádzať buď priamo, alebo kontextuálne v dokumentoch, na ktorých bude postavený model RAG. Použitie len triviálnych metód pri tvorbe RAG nemusí pri zložitejších dokumentoch (štruktúrou alebo obsahom) viesť k vysokej miere úspešnosti odpovedí modelu. Preto chceme implementovať viaceré heuristiky, ktoré pomôžu pri doručovaní informácií potrebných na zodpovedanie otázok väčším jazykovým modelom, ktorý bol nezávisle od našich požiadaviek natrenovaný na všeobecné otázky a odpovede.

Konkrétne môžeme použiť pridanie metadát k častiam surového textu, reranking vyextrahovaných častí, knowledge grafy z častí a iné podobné heuristiky zamerané na lepšie sprostredkovanie informácií modelu. Samozrejme, k týmto heuristikám pridáme aj testy na overenie skutočného zlepšenia nášho RAG modelu.

Vedúci: Michal Uherek
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 16.10.2024

Dátum schválenia: 18.10.2024 doc. Mgr. Tomáš Vinař, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Teo Pazera
Study programme:	Data Science (Joint degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science Mathematics
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Enhancement of RAG chatbots performance using various heuristics

Annotation: The goal of this thesis is to test and showcase various approaches to improving the probability of obtaining correct answers from the model for a given set of questions, whose answers are either directly or contextually contained in the documents used to build the RAG model. Using only trivial methodologies in building RAG may not yield great results with structurally or contextually complex documents. Therefore, we aim to implement several heuristics that could enhance information delivery to the large language model, which was previously independently trained on general question answering.

Specifically, this can include adding metadata to chunks of raw text, reranking the selected chunks, creating knowledge graphs from text chunks, and other similar heuristics focused on better information delivery to the LLM. Additionally, we will conduct benchmark tests to verify the actual improvement of our RAG model with these heuristics.

Supervisor:	Michal Uherek
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	16.10.2024
Approved:	18.10.2024

doc. Mgr. Tomáš Vinař, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: I would like to thank my supervisor, Mgr. Michal Uherek, for his invaluable guidance and support throughout the writing of this thesis. His mentorship provided me with the flexibility to explore a range of fascinating research topics while managing my work commitments, for which I am deeply thankful.

I also extend my appreciation to my colleagues at Zurich Insurance Company Ltd., particularly Tomáš Antal, whose engaging and insightful discussions led to meaningful refinements in this work, and Tara Stefányi, whose assistance was instrumental in clarifying the ideas presented in this thesis.

Last but not least, I thank my family for all of the support during my studies.

Abstrakt

Táto práca, realizovaná v spolupráci so spoločnosťou Zurich Insurance Company Ltd., sa zameriava na hľadanie optimálneho dizajnu RAG modelu pre automatizované vypĺňanie hlásení o zákazníkoch v oblasti obnoviteľnosti. RAG model kombinuje vyhľadávanie relevantných textov s jazykovými modelmi na generovanie odpovedí na špecifické otázky z hlásení. Cieľom bolo otestovať rôzne heuristické prístupy pri vyhľadávaní relevantných informácií, ako aj predspracovanie textu do grafov znalostí či použitie alternatívnych metód generovania odpovedí. Modely boli testované generovaním hlásení a ich porovnaním s ručne vypracovanými verziami. Kvalita hlásení bola hodnotená pomocou metrík informačného vyhľadávania, presnosti a dôveryhodnosti odpovedí jazykového modelu. Zohľadňovala sa aj časová a finančná efektívnosť modelu, ako aj overiteľnosť jeho odpovedí. Na základe analýzy a manuálneho porovnania výsledkov bol identifikovaný vhodný model, ktorý tím v súčasnosti využíva pri tvorbe hodnotení zákazníkov.

Kľúčové slová: Veľké jazykové modely, RAG, generovanie textu s podporou vyhľadávania, reportovanie udržateľnosti, informačné vyhľadávanie, Graph RAG, graf znalostí, NLP, vektorové databázy, HyDE vyhľadávanie, GPT-reranking

Abstract

This thesis, conducted in collaboration with Zurich Insurance Company Ltd., focuses on identifying the optimal design of a Retrieval-Augmented Generation (RAG) model for the automated completion of customer sustainability reports. The RAG model combines retrieval of relevant text chunks with large language models to generate answers to specific questions from the report. The objective of this thesis was to evaluate various heuristic approaches for retrieving relevant information before generating responses, including methods such as converting documents into knowledge graphs or applying alternative answer-generation techniques. The models were tested by generating reports and comparing them with manually created versions. We evaluated the quality of the reports using information retrieval metrics as well as the accuracy and faithfulness of the large language model's responses. Attention was given to the model's time and cost efficiency, as well as the verifiability of its answers. After thorough analysis and manual comparison of the results, we identified a suitable model that is now used by the team to support customer assessments.

Keywords: Large language models, RAG, Retrieval-Augmented Generation, sustainability reporting, information retrieval, Graph RAG, knowledge graph, NLP, vector database, HyDE search, GPT-reranking

Declaration of Originality

I hereby declare that I have written this bachelor's thesis titled "Enhancement of RAG chatbots performance using various heuristics" independently, using the literature listed in the bibliography and generative artificial intelligence tools (such as ChatGPT) for grammar and style correction. I confirm that the use of AI tools complies with applicable legal regulations, academic rights and freedoms, and ethical principles, while maintaining academic integrity. I take full responsibility for the final content and form of this work.

Contents

List of Figures

List of Tables

Introduction

In recent years, large language models (LLMs) have demonstrated remarkable capabilities in automating complex tasks across diverse domains. One particularly powerful approach is Retrieval-Augmented Generation (RAG), which combines a retrieval mechanism with a generative language model to produce accurate, contextually relevant text by drawing on a specific document collection. This hybrid method is ideally suited for applications that require both precise extraction of information and fluent narrative generation, such as the automated creation of structured reports.

This thesis investigates the application of RAG models in the insurance industry, focusing on the automation of generating customer sustainability reports for Zurich Insurance Company Ltd. In this context, sustainability refers to the environmental performance and ESG (environmental, social, governance) alignment of corporate clients. Insurers use these reports to assess ESG risks, inform underwriting and investment decisions, ensure regulatory compliance, and guide clients toward improved practices.

Traditionally, sustainability reports were compiled manually: analysts reviewed publicly available documents, extracted key data, and completed standardized questionnaires. Our RAG-based workflow replaces much of this labor by pre-processing source documents into text chunks, retrieving those chunks relevant to each report question, and prompting an LLM to generate precise answers for the questionnaire.

In collaboration with the Sustainability Team at Zurich Insurance, we explore various heuristic configurations of the RAG pipeline to determine an optimal design. Chapter 1 reviews the fundamentals of RAG architectures, what components does such model include, how they interact with each other and what are the possible limitations of such model. In Chapter 2 we present several enhancements to most of the components to tackle possible deficiencies of such model. These enhancements will be the main topic of discussion as after setting the base parameters for RAG model they are the secondary thing we can improve in RAG to get better assurance that the model will output answer most suitable for given question Chapter 3 describes our implementation of such solution, basically showcasing the insides of the model we built. It shortly describes how basic RAG approach was implemented, focusing mainly on the implementation of heuristic approaches. Finally, Chapter 4 defines the metrics used

to evaluate model performance, presents statistical significance tests comparing different configurations, answers of different models are compared to show the differences of models answers. The costs of such model configurations are as well compared in this chapter. When all of this extensive experimentation and manual comparison with human-generated reports is performed, we identify a RAG configuration that achieves an optimal balance of accuracy, cost-effectiveness, and verifiability.

This solution has been deployed by the Zurich team to streamline customer sustainability assessments, demonstrating both the practical viability and impact of our research.

Chapter 1

Retrieval augmented generation

In this chapter, we will focus on explaining the deficiencies large language models (LLMs) have and how retrieval-augmented generation (RAG) may come in and help to tackle them. Furthermore we will go through the general structure of this question answering model, functionality of each of its component, the flaws of the simplest implementation of RAG and outline potential solutions to these flaws.

1.1 Challenges in Leveraging Large Language Models

1.1.1 Limitations of Large Language Models

We will assume the majority, if not all, of readers have interacted with LLMs, particularly generative pre-trained transformers (GPT), which have been accessible to the general public since the early 2020s. In your experience of using such language models, you have likely encountered instances where questions were answered either completely incorrectly, partially inaccurately, or, in some cases, the model admitted that it lacked sufficient information to respond.

The responses of all LLMs are prone to hallucination, as the task to generate text (answer) that forms a natural continuation from the input text (question) requires the model to hallucinate text [24]. The generation of text that, while coherent and fluent, may not be factually accurate or grounded in reliable sources. This issue arises because language models are typically trained to generate text that forms a natural continuation of the input text but may lack the specific, factual knowledge required for some queries. While some questions you may have asked were from publicly available information (e.g., searchable on the web), others might have required answers based on information that the model could not have been trained on.

To address this, you might consider enriching the model's knowledge base. Fine-tuning LLMs is one approach, but it is costly and requires a sufficiently large dataset of relevant questions and answers. A more practical solution is to integrate an external

information-seeking component to provide the necessary context to the LLM.

1.1.2 The Role of RAG

RAG is designed to address the limitations of LLMs by enabling them to answer questions that may not fall within publicly available knowledge or the model's training data. It achieves this by enhancing the LLM with an external knowledge retrieval component. When a query is posed, the model doesn't rely solely on its pre-trained knowledge; instead, it retrieves relevant documents or text snippets from a large database. These retrieved documents provide a factual foundation for generating responses, effectively grounding the model's answers in verifiable and accurate information.

1.2 Architecture of RAG models

As the goal of this thesis is to find the optimal design of the RAG model, we need to dive into the architecture. We will look at all parts of the model to understand where improvements or multiple approaches may be taken.

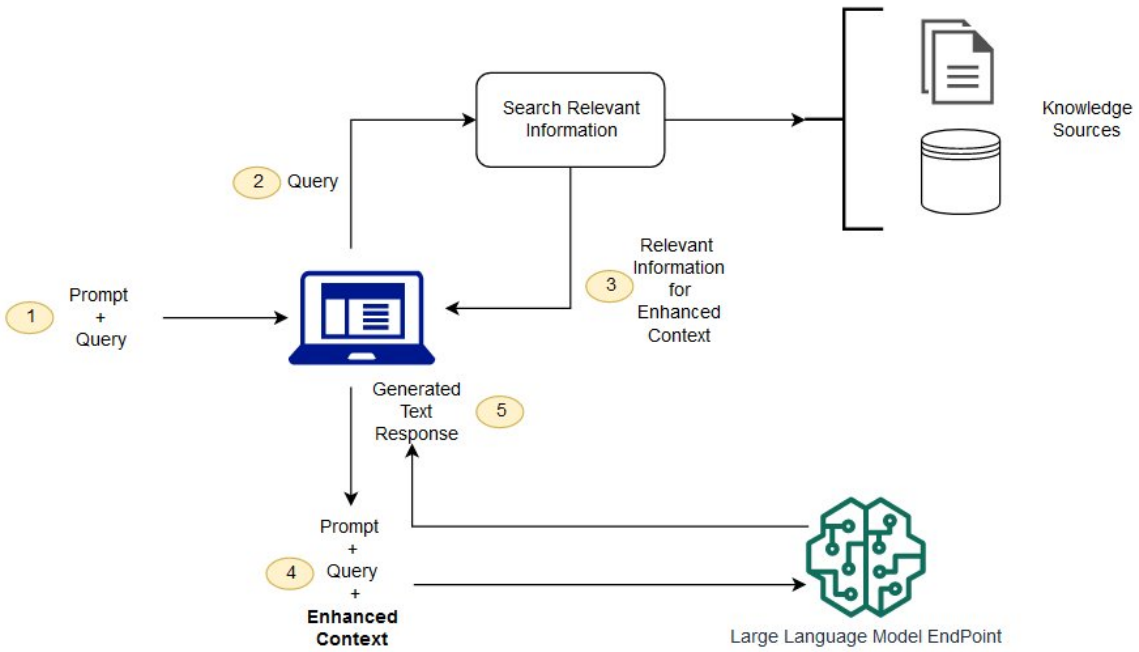

Figure 1.1: Figure from [4], illustrates how RAG model operates when query/question is submitted. Its information retrieval component searches in already preprocessed knowledge sources. The enhanced context found is then send together with the query and base prompt describing the general context of the task to the LLM which generates the answer RAG outputs.

We will go through each part more thoroughly to make sense of what actually happens in figure 1.1. Firstly the LLM component will be quickly introduced, than we will delve into the information seeking component of RAG. Next we will discuss the preparation of data for RAG. Lastly, we will focus on chaining retrieved information with base prompt and query to get a response.

1.2.1 Generative pre-trained transformers

We will only touch the surface on how GPT models operate as it is not usual to try fine-tuning them in RAG implementations. However they are the backbone of the entire RAG and without them RAG models would not exist.

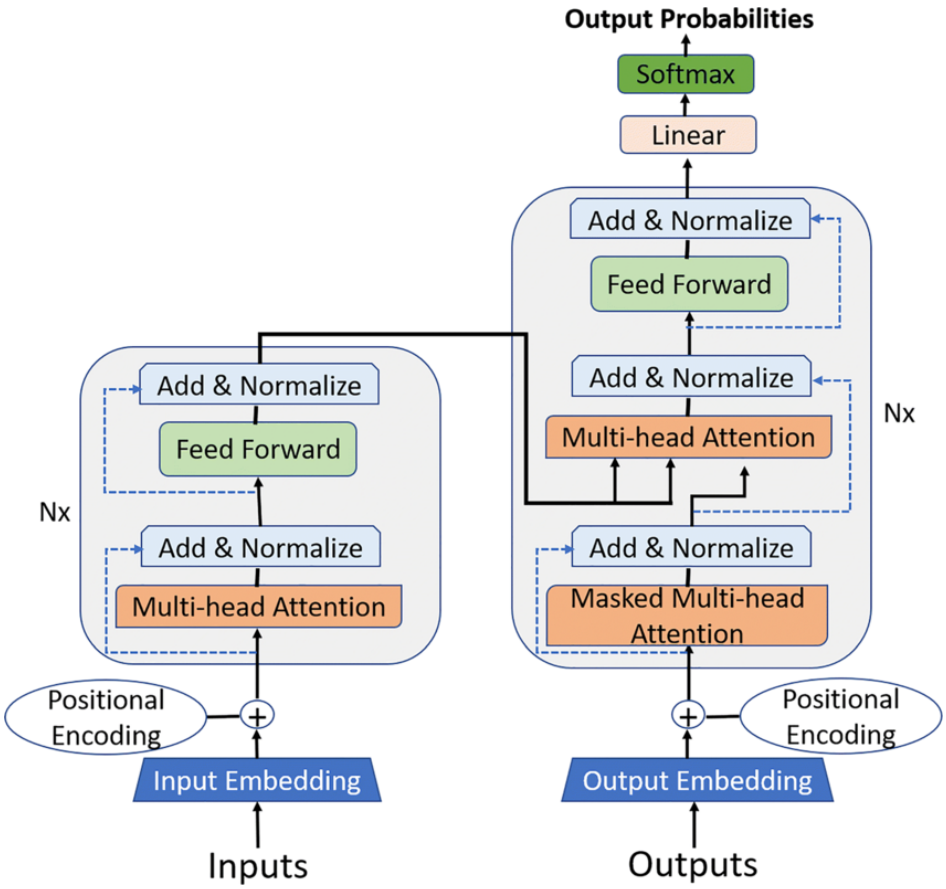

Figure 1.2: A simplified illustration of the Transformer model architecture, highlighting key components relevant to GPTs. This diagram shows the flow of information through the input embeddings, positional encoding, attention mechanisms, feed-forward layers, and the final softmax output. While GPTs utilize a similar architecture, they primarily focus on the decoder portion for text generation, which is on the left side of this figure.

GPTs operate mostly on architecture first published by the Google research team, which showcased that an adding attention mechanism to the previously established architecture of an encoder and decoder significantly improved performance of the model

[36]. While their task focused on translation of text, subsequent research built upon their work led to the first GPTs.

As illustrated in Figure 1.2, we can imagine the architecture of GPTs as a process that begins with the input layer, where each word (or token) in a sentence is transformed into a numerical vector. It operates with vocabulary of set size, this way it can represent each word with one-hot encoding. Which means vector of length of our vocabulary, where only the dimension at index i (the word's value) is 1, and all others are 0 [7]. Operating with vectors of these sizes mostly filled with zeros would be ineffective, therefore an embedding function is used that maps these vectors into a space with smaller dimensions and tries to align words with similar meaning close to each other in the given vector space. We will discuss embeddings in later parts of this thesis as they are crucial in retrieval of relevant documents in RAG models.

These embeddings are then passed through a positional encoding function, which enriches them with information about their position in the given text. Subsequently, attention function tries to predict how much important each part of the text is for the output generation.

Afterwards, the vectors are processed through a feed-forward block, a component of the architecture that applies transformations learned during training through backpropagation. Backpropagation used here works almost just as in multilayer perceptrons [15], adjusting the weights within the network to minimize error.

Lastly, an embedding function is used, which we established earlier to encode one-hot embeddings into smaller vector space, however now the inverse of this encoding has to be used to get back to the representation of each word. The softmax function is applied on the vocabulary vector to obtain a proper probability distribution which represents how likely each word/token will be added to the output. The GPT model then uses sampling technique to select next word/token based on the probability distribution previously obtained. Often though not all of the words from the dictionary are taken into consideration for the next word and only top k words from dictionary have a non zero probability of being sampled [7].

As we stated earlier, this is a vast oversimplification of how the model operates, but it at least gives us some idea on how we should think about GPTs. From now on, we will treat GPT models as a black box—a complex system that takes input text and provides responses without delving into the internal workings of its architecture.

1.2.2 Retrieval of information

In this section, we will discuss different approaches to retrieving relevant information in RAG, comparing traditional keyword-based methods with modern vector-based retrieval techniques. As previously mentioned, in principle, RAG model consists of a

GPT model and an information retrieval mechanism. In our case our knowledge base for RAG consists of text documents, therefore we would like to be able to effectively query and select parts of these documents. Preprocessing of documents, parsing and chunking will be discussed in the following subsections, for now lets assume we already have a database containing chunks of text extracted from various text documents that carry information, which should enrich our RAG model. We want to search through the database to find relevant chunks of text to the query our RAG model is given.

A common method, we might first think of is keyword matching between the chunks of texts stored in the database and the query. It seems logical that texts which contain the same words as the query should be somewhat helpful to answer the question. An Advanced approach to do this is Best Match 25 (BM25), which evaluates text relevance by considering term frequency and document length. Unlike simple keyword matching, BM25 rewards documents with frequent relevant terms while balancing the document length to avoid favoring longer texts excessively [33]. BM25 works well when users are familiar with the terminology and structure of the texts in the database, and their queries closely match the wording of the stored content. However, its effectiveness significantly diminishes when the query uses synonyms or semantically similar phrases that do not directly match the terms in the database. This limitation arises because BM25 is inherently based on lexical matching, failing to capture deeper semantic relationships between words [18].

To address this challenge, vector databases (VecDBs) provide a more robust solution by representing text as dense numerical vectors in a high-dimensional space [14]. These vectors are often referred to as embeddings. Embeddings that are used in VecDBs are generated by machine learning models trained on a large corpus of text, in effect letting them capture complex relationships between texts. These models firstly tokenize the text they are given, next the tokens are encoded using a neural network trained to capture semantical meaning of tokens surrounding them. Finally this encoded model is then put together into single embedding encoding the semantical meaning of the entire text.

Retrieval of information in VecDBs operates on the premise that texts semantically similar to the query are as well relevant to answer the query. We can assume there is some truth to this premise, as VecDBs are widely used in RAG solutions [18]. The first step in utilizing VecDBs is to embed all the text snippets we have prepared to be in our knowledge base. Once all text snippets are embedded, we can search for relevant texts. Given query is embedded using the same function as the one, that was used to build the database. Then we want to find k most similar text snippets to the given query. Cosine similarity is most widely used metric to find these k most similar vectors, however Euclidean distance or other metrics might be used as well [18, 16].

Working with large VecDBs often requires efficiently finding the most similar em-

beddings to a given query. A brute-force search, comparing the query against every stored embedding, can be computationally expensive and time-consuming. To address this challenge, approximate nearest neighbor search algorithms are employed. One such powerful technique is the Hierarchical Navigable Small World (HNSW) graph, which provides an efficient way to estimate the k most similar embeddings. HNSW graphs leverage a multi-layered graph structure to navigate the embedding space quickly [37].

To summarize, effective information retrieval is essential for RAG, as without relevant information being provided for the GPT model, no improvement is expected from the RAG response in comparison to the GPT. VecDBs are the primary option for this task by leveraging embeddings for semantic search, enabling the retrieval of contextually relevant information.

1.2.3 Parsing of documents with relevant information

In this subsection reading of individual text documents and methods used to get as much information out of given document will be clarified. For RAG to be successful in answering questions, we need to ensure we have a proper knowledge base, which has enough information in it for all the possible questions we might want to ask. In my experience, this information primarily came in a form of PDF documents. With that said any form of structured or unstructured text documents can be used to provide context to the question asked.

Lets assume we have been given various PDFs on top of which RAG should be built. We need to extract as much information from these structured, semi-structured PDFs, while preserving their structural relationships [39]. As our RAG model operates in python, we will be parsing the information out of the PDFs using a python program. Many parsers for PDF files have been developed in python for example: PyMuPDF [2], Docling [6], Unstructured [35].

Therefore, instead of developing a new parser and we can try any of the above mentioned libraries. Parsing may seem like an easy task just taking all of the text out of the PDFs. However when we encounter more structurally fragmented files, that contain extensive amounts of tabular data or text structured in an unorthodox manner, the parsing suddenly becomes a lot harder. We may also want to avoid using different representation of our desired outcome of the parsing than just a plain text, but look into more structured ways of representing the text like HTML, markdown or JSON. This gives us opportunity to capture data that is being stored in tables. Or we might consider storing names of chapters/sections more easily, if we wanted our RAG model to cite from where it got the relevant information from.

The last thing we will mention is that our PDFs might be in the form of a scanned PDF/an image-based PDF. Cutting edge parsers can tackle this challenge by perform-

ing layout analysis prior to the text extraction and determine whether there is a need to use optical character recognition (OCR) to access the text in the documents. These parsers might detect tables, mathematical expression or charts and treat them differently to get out as much information as possible [39]. Even though they are still not perfect we can expect the performance of RAG model to improve as we use better parser and get information in a more polished form.

1.2.4 Chunking of text

Chunking is a crucial step in RAG, as it determines how text is segmented before embedding. The choice of chunking strategy can significantly impact retrieval accuracy, citation clarity, and overall system performance. In this section, we will explore different chunking techniques, including fixed-size chunking, recursive character text splitting, contextual chunking, and structure-based approaches. Each method has its strengths and trade-offs depending on the document type and use case.

When chunking text we only consider keeping chunks from the same file in one embedding as doing the opposite brings no advantage. The strategy we use depends on the individual PDF we parsed. When the text from the page only relates to text within the same page, for example in slide presentations we may want to cut off text right at the ends of pages. This gives us precise information about page number the text comes from, which might be useful when we want RAG to cite the information. However, if text is overflowing between pages we must carefully consider, whether splitting it into two chunks could results in the loss of context to answer question accurately.

There are four options to pick from when choosing a chunking strategy, we may use them to chunk text on pages or on the entire text from the file. When using fixed size chunking we set a fixed size of characters/tokens to split the text into and we split right at the end of a word if it exceeds the length. Recursive character text splitting iteratively searches for characters, we pick as separators like newlines and divides the text, when including next separator would exceed the length we choose for the chunks. Contextual strategy focuses on keeping text with similar context together in one chunk using various NLP techniques. Lastly, we may want to try and use the inherent structure from the files that we parsed out, or combine parts of the strategies mentioned above to get the best result on our set of documents [17].

One critical decision is determining the optimal length of the chunks. This decision is influenced by several factors, including the limitations of embedding models, which impose constraints on the maximum length of text that can be embedded. Additionally, the text cannot be too short, as splitting text in the middle of a sentence or thought can lead to loss of information, particularly if only a fragment of the sentence

is retrieved. To ensure consistency and efficiency, it is advisable to maintain chunks of similar lengths. This is particularly important when fitting chunks into the context window of a GPT model, as it allows for better estimation of how much information can be included. While it is not strictly necessary to enforce a fixed length, aiming for chunks of a consistent size helps optimize the system's performance. The optimal chunk length can vary depending on the type of text documents being processed and the nature of the queries. For instance, some queries may require information from multiple parts of a document to be combined effectively, making smaller chunks more suitable. On the contrary, other queries may benefit from longer, coherent paragraphs that provide comprehensive context. Therefore, testing the performance of RAG with different chunk lengths is essential to identify the most effective approach for a given dataset and use case [19].

Chunking is one of the most flexible components of RAG, and we will explore some approaches in the chapter 3 where we implement all heuristics.

1.2.5 Prompt engineering

While working with GPT models, a common optimizing problem shows up, that revolves around asking a GPT model given question in the right wording so the answer the model generates satisfies our needs. Prompt engineering is a discipline revolving around efficiently structuring instructions for the language models, in effect broadening capabilities and extending limitations these models have. Various strategies have been already shown to improve performance [27].

Prompts in RAG typically consist of the following components:

- **Instruction:** A specific task, role which GPT model should take or instructions that direct the model's behavior.
- **Context:** External information or additional details that guide the model towards generating a better response.
- **Input Data:** The input question or data for which we seek a response.
- **Output Indicator:** The expected type or format of the model's output.

Common practice while developing a RAG model is trying a few prompts to figure out which fits the given task best. As well it is good practice to start with simple prompt first and move on to more complicated instructions iteratively. While prompting you have to keep in mind how specific you want your prompt to be if your application needs to answer to only one kind of question being specific about it may improve results drastically [28].

We can also distinguish prompting based on how many times we send input to the GPT model to give us answer. Zero-shot prompting is defined by asking the model question just once to get desired output, on the other hand few-shot prompting allows us to interact with the model many times and specify/correct our needs. Few-shot prompting can be done on the backend without the user interacting with the product generated throughout the process. Well established strategy, chain of though, revolves around decomposition of the task into smaller sub-problems and separately solving each of them in result combining solutions of all of the sub-problems. This method tries to mimic human like reasoning, logically deducing conclusions. Even more advanced technique uses both 'acting' and 'reasoning', decomposing question while also being capable of searching for relevant information in between steps. ReAct as this approach is referred to, is often used outside of RAG applications to enable LLMs with additional capabilities acting as sort of an agent that has various tools while composing an answer. However implementing ReAct in RAG application seems promising as it helps us decompose challenging requests on our RAG into smaller tasks that can be solved more easily by our RAG [26].

Prompt engineering is important for getting as much performance from our model as possible, however it might feel like we are just hoping for better results with differently structured prompts as we are never fully certain, why given prompt improved answer of the model.

1.2.6 Chains

In RAG, the effectiveness of a system does not solely depend on individual components like retrievers, chunking strategies, or embedding models, it also hinges on how these components interact. To orchestrate these interactions, we need chains, which serve as a structured pipeline that connects multiple steps in the retrieval and generation process.

We already discussed some forms of chains without explicitly mentioning that we are talking about them. For example, while few-shot prompting, we are creating a chain where we send multiple requests to the GPT model to receive our final answer. Similarly, when using conversation summary memory, we rely on the GPT model after generating each new answer to prepare a summary of the entire conversation, which can be used for the next question-answering session.

The simplest chain in RAG is the stuff chain. It takes a list of retrieved documents from a retriever, which queried our database with a given question, then inserts them into a predefined prompt alongside the question. This filled prompt is sent to the GPT, and the chain simply returns GPT's answer [34, 21].

A more advanced example is the map-reduce chain, which is useful when we want

the model to process more documents than can fit into one call. This chain uses multiple calls, each processing a distinct subset of documents to generate partial answers. These partial answers are then combined in a final call to GPT to produce the ultimate response. However, the disadvantages of this approach include the potential loss of information during multiple calls and the increased difficulty in debugging errors, as multiple outputs need to be analyzed for a single question [34, 21].

Another way to work with more documents than the token limit allows is the refine chain. In this chain, an answer is first generated using a subset of the documents. This preliminary answer is then iteratively refined by combining it with additional documents in subsequent calls. This process continues until all documents are processed. While refine chains can provide more comprehensive answers, they are prone to error propagation—if an intermediate GPT response contains a mistake, subsequent refinements may carry forward or even amplify this error [34, 21].

Beyond these basic forms, custom chains can also be designed to cater to specific use cases. For example, chains can include conditional logic to handle particular document types differently, or employ additional preprocessing steps such as document summarization, filtering, or embedding-based ranking to optimize the input to the GPT. The flexibility of chain design makes it possible to address domain-specific challenges in RAG systems effectively.

Once again, testing different approaches is essential to determine what best suits our RAG system. The choice of chain depends heavily on the nature of the data, the complexity of the queries, and the requirements for output quality. Experimentation and iteration are the best ways to refine the process and ensure the chosen chain aligns with the desired outcomes.

Chapter 2

Heuristics to improve RAG performance

RAG as a concept was introduced in the first chapter with both practical usage and flaws in some of its components. In this chapter we will look at multiple enhancements that should in theory improve the performance of RAG. Mostly these adjustments will focus on ways to ensure the correct information is retrieved and structured well for the LLM to answer desired questions asked by users.

2.1 Chunks enriched with metadata

The chunks stored in our VecDB contain valuable information, but they consist solely of extracted text without their original context. This can lead to ambiguity, especially when dealing with multiple documents that contain similar content but pertain to different entities.

For example, consider two documents, one discussing insurance policies in Italy and the other covering policies in Switzerland. If the extracted text does not explicitly mention the country, GPT would have no way of distinguishing between the two. This lack of context could lead to inaccurate or misleading responses.

To address this, we should anticipate the types of user queries that require disambiguation and enrich our text chunks with relevant metadata. Useful contextual information—such as the document name, chapter title, or section heading—can often be determined in advance. Including this metadata at the beginning of each chunk can significantly improve retrieval accuracy and the relevance of responses [1, 23].

However, adding metadata comes with trade-offs. It can shift the retrieval process to favor chunks that contain specific metadata rather than those most semantically relevant to the query. Additionally, since all metadata is sent to GPT along with the text, it consumes extra tokens, potentially leading to higher costs and reducing the

available context window for actual content.

2.2 Semantic chunking

In Section 1.2.4, we introduced chunking strategies that group contextually similar text segments. Semantic chunking specifically involves clustering sentences based on their semantic similarity, ensuring each chunk conveys a coherent idea.

The process begins with sentence segmentation, where the raw text is divided into individual sentences. This can be achieved using algorithms like the Punkt Sentence Tokenizer, which is an unsupervised algorithm. It analyzes patterns in punctuation and capitalization to identify sentence boundaries, eliminating the need for predefined rules. This method is effective across various languages and writing styles. Once the text is segmented, each sentence undergoes sentence embedding, where it is converted into a numerical vector representation. We take embedding similarity as our measurement of how close contextually two sentences are. Finally, clustering techniques such as K-means are applied to group semantically similar sentences together. The number of clusters, denoted as k, is predetermined and can be adjusted to achieve the desired chunk length [20].

The aim of this strategy is to maximize the amount of sentences covering the topic close to users query, which are sent to the GPT model to generate answer. One downside of this approach might be that we are no longer able to cite the part of documents from which the answer came as the sentences from various sections of the document are concatenated. We are only able to directly cite the document or the concatenated sentences. Another downside is that it takes more effort to set up semantic chunks too, creating embeddings for each sentence and the clustering take more time than simply splitting based on some specific characters or length of text.

2.3 Hybrid Search: Combination of BM25 and Vector Search

As we established in section 1.2.2 that in most RAG solutions, storing chunks in a VecDB is a common practice as it allows for searching at a deeper level than just word matching. However, there is a possibility to combine both of these approaches, as it may be beneficial to certain queries to prioritize chunks with the most occurrences of certain words. Hybrid search creates an ensemble of BM25 retrieval and vector cosine similarity search [5].

In this context, rank fusion plays a crucial role in combining the results of the two retrieval techniques to optimize the overall retrieval quality. Rank fusion refers to the

process of combining multiple ranked lists from different retrieval methods into a single, unified ranking. It can be broken down into a series of steps. First, each retrieval method (BM25 and vector-based search) independently retrieves a set of relevant documents or chunks based on its specific criteria.

After the retrieval, each method scores the documents it has found. BM25 ranks documents based on term frequency and inverse document frequency (TF-IDF), while vector search ranks documents based on the cosine similarity between the query and the document embeddings. Once these results are scored and ranked independently, the individual rankings are combined into a final ranking, which is a merged ranking that attempts to leverage the strengths of both retrieval methods. This can be achieved using two primary approaches:

1. Reciprocal Rank Fusion (RRF): The RRF formula assigns a score to each document based on its rank in the two systems:

$$\text{RRF Score} = \frac{1}{k + \text{rank}_{\text{BM25}}} + \frac{1}{k + \text{rank}_{\text{Vector}}}$$

- $\text{rank}_{\text{BM25}}$ is the rank of the document in the BM25 retrieval system.
- $\text{rank}_{\text{Vector}}$ is the rank of the document in the vector-based retrieval system.
- k is a damping factor (typically set to 60 to reduce the influence of low-ranked documents).

2. Weighted Rank Fusion: Alternatively, a weighted approach can be used to combine the ranks, where a parameter α controls the contribution of each retrieval method. The formula for weighted rank fusion is:

$$\text{Weighted Score} = \alpha \cdot \frac{1}{\text{rank}_{\text{BM25}}} + (1 - \alpha) \cdot \frac{1}{\text{rank}_{\text{Vector}}}$$

- α is a weighting parameter that ranges from 0 to 1.
- $\text{rank}_{\text{BM25}}$ and $\text{rank}_{\text{Vector}}$ are the ranks of the document in the BM25 and vector-based retrieval systems, respectively.

However, there are downsides to the hybrid search. Running multiple retrieval processes simultaneously demands more computational resources, which can increase costs and slow down response times. This overhead may affect system performance, especially in real-time applications. Additionally, rank fusion might lead to diminishing returns if the individual retrieval methods do not offer significantly different strengths. If both methods perform similarly, combining them could result in a more complex system without providing substantial improvements in performance.

2.4 HyDE-Retrieval

Traditional retrieval methods, such as dense and sparse retrieval techniques, often struggle with vague queries or queries that lack sufficient context. Hypothetical Document Embeddings (HyDE) retrieval is a heuristic approach designed to address this challenge by leveraging the generative capabilities of LLMs, to enhance retrieval performance. Instead of directly searching for relevant documents using the raw query, HyDE first generates a hypothetical document that represents a plausible answer. This synthetic document is then embedded into the same vector space as the stored documents, and retrieval is performed based on its semantic similarity to the database entries.

Building upon this idea, HyDE takes advantage of the LLMs' ability to generate text that aligns with the intent behind the query. The LLM generates a synthetic document that represents a likely response. This generated text does not need to be factually correct; its purpose is to serve as a prototype that captures the semantics of a relevant document. After generation, the synthetic document is embedded into a vector space using the same embedding model that was applied to the stored documents in the VecDB. Once the synthetic document is embedded, retrieval is performed by comparing its vector representation against the stored document embeddings in VecDB, rest of the process stays the same as in the traditional dense vector retrieval [11].

HyDE retrieval offers several advantages over traditional retrieval techniques. By generating a hypothetical answer, it effectively expands the query with semantically richer content, improving recall and reducing the risk of missing relevant documents due to limited keyword overlap. It also improves handling of ambiguous or poorly formulated queries, as the generative process introduces additional context that refines the retrieval process. Furthermore, the approach enhances semantic matching since the generated document is likely to share contextual similarities with relevant stored documents.

2.5 Reranking of retrieved chunks

Traditionally, in RAG models, only a handful of retrieved text chunks are used to generate an answer. These chunks are typically selected using vector similarity search, where the top k most similar chunks to the user's query are retrieved. Increasing k raises the likelihood that all relevant information is retrieved, but it also introduces challenges—namely, the limited context window of LLMs and increased risk of hallucinations when too much information is included [22].

To address this, rerankers modify the way retrieved chunks are used by introducing an additional step: reordering the chunks based on their actual relevance rather

than their precomputed vector similarity. Instead of relying solely on cosine similarity between query and chunk embeddings, rerankers assess each retrieved chunk in the context of the specific query, ensuring that the most relevant information is prioritized. Rerankers are therefore ideal for allowing pure vector search to initially select larger number of chunks, for example 20 and the reduce the number of actually used documents back to 5, with goal of the reranking that now the newly ranked top 5 documents will be the most relevant for the query.

There are multiple approaches to reranking, however we will delve deeper into cross-encoder based rerankers and rerankers using and fine-tuning LLMs for the task of ranking the relevance of documents to a given question. Lastly, the concept of peripheral reranking that utilizes the fact that LLMs are more focused on the information that is at top or bottom of their context windows is also going to be explained in this section too [22].

2.5.1 Cross-encoders

Cross-encoders are a class of models designed to evaluate the relevance of a text chunk in direct relation to a specific query by jointly processing both inputs. Unlike bi-encoders, which independently encode the query and the text chunk into embeddings and then compute their similarity, cross-encoders concatenate the query and the chunk and feed them together into the model. This approach allows the model to capture intricate interactions between the query and the text, often leading to more accurate relevance assessments.

The architecture of a cross-encoder typically involves inputting the concatenated query and text chunk into a transformer-based model, such as BERT. The model processes the combined input and produces a relevance score indicating how well the chunk matches the query. This method enables the model to consider context and nuances that might be overlooked when processing the inputs separately [29].

While cross-encoders often deliver superior performance compared to bi-encoders due to their ability to model complex interactions between queries and text chunks, they are computationally more intensive. This increased computational cost arises because each query-chunk pair must be processed together, making cross-encoders less efficient for large-scale retrieval tasks. Consequently, they are commonly used in a rerankers, where a smaller subset of candidate chunks has already been identified by a faster retrieval method [3].

In summary, cross-encoders play a pivotal role in refining the selection of text chunks in RAG models by jointly processing queries and chunks to accurately assess relevance. Despite their higher computational demands, their ability to capture detailed interactions makes them valuable for tasks requiring precise relevance judgments.

2.5.2 LLM based reranking

Cross-encoders which are covered in previous subsection utilize the option of pair-wise ranking, generating score for the chunk based of the modellooking only at the query and chunk without information of what the other $k-1$ documents look like. It would make a lot of sense to use list-wise ranking, which lets the ranking model look at all the documents at once with the question and then decide what the optimal order looks like. LLMs might be well-suited for this application, as they are capable of consuming large text inputs and adhere to provided instructions they are given.

Downside for using an LLM to rank documents relevance is paying tokens for all of the retrieved chunks and then paying for the select documents on top of it. It has been shown that LLMs have a good enough ability to output decent ranking of the documents [32, 38]. Therefore we can employ it for this task and see, whether for our use case the extra tokens and extra time will be worth the improvement in the performance of the retrieval and answers.

There is a small catch though, we mentioned before that the context window of LLMs we will be using is not as big and the main problem we need to solve when using rerankers in our RAG model is that we can not fit all of the documents into the model at once. Same applies to LLM that would be reranking the chunks for us, however we can adopt a sliding window strategy. This strategy fills the context window with n chunks and ask for the ranking of them, in our next call for ranking we will remove m currently worst ranked chunks from the adepts and fill in m new adepts for ranking [38, 32]. We can repeat this process until we showed the LLM all of the adepts that have been retrieved by simple vector search. The final output of the ranking will be $n-m$ ranked chunks, which we can use to answer the given question.

2.5.3 Peripheral reranking

Peripheral reranking differs from other reranking strategies by not aiming to rank the chunks themselves. Instead, it leverages an existing ranking to reorder the chunks before they are sent in a question-answering LLM call. Some LLMs have been shown to lose attention to text positioned in the middle of their context windows [22]. To mitigate this, peripheral reranking aims to structure the chunk order so that the most important information appears at the beginning and end of the input. By doing so, the least relevant chunks are placed in the middle, where they are more likely to be overlooked.

A straightforward way to implement this is to place the highest-ranked chunk at the start, followed by the second-highest-ranked chunk at the end. The third-ranked chunk is positioned second, the fourth-ranked chunk second to last, and this alternating pattern continues until all chunks are placed. This approach introduces minimal com-

putational overhead while potentially improving the performance of the RAG model. However, the exact impact of this method remains uncertain and requires further evaluation.

2.6 Graph RAG

Graph RAG represents an evolution of traditional RAG models by incorporating a knowledge graph (KG) into the retrieval and generation process. In this section, we explore how a KG constructed from text documents within our knowledge base can enhance our model's ability to answer all sorts of questions.

2.6.1 Knowledge Graph

First of all we need to declare what a KG even is, a knowledge graph is a structured representation of information that captures entities and their interrelations within a specific domain. This structure enables both humans and machines to interpret and reason about the data effectively. Knowledge graphs typically consist of nodes, which represent entities and edges depicting the relationships between these entities, forming a graph-based model that facilitates complex querying and inference.

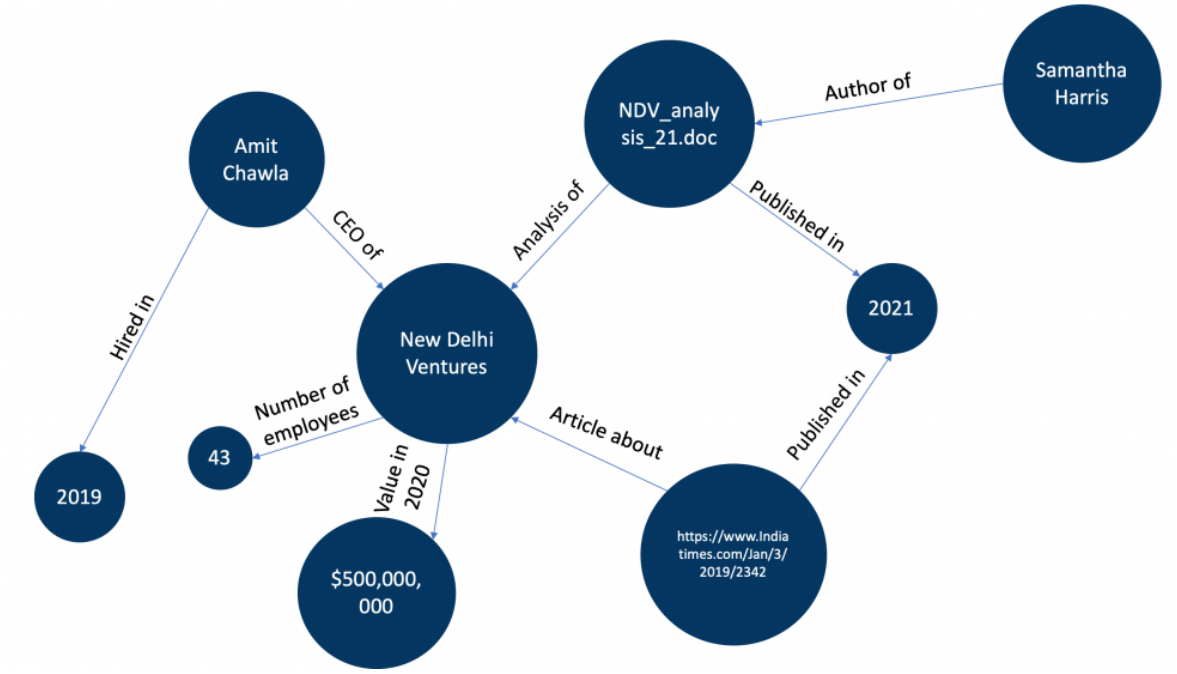

Figure 2.1: An example of a knowledge graph illustrating how entities and their relationships are usually represented and how we can visualize such objects[25].

The concept of knowledge graphs gained significant attention with the introduction of Google's Knowledge Graph in 2012, which enhanced search engine capabilities by

understanding the semantics of user queries. Since then, knowledge graphs have been widely adopted across various domains, including semantic web technologies, data integration, and artificial intelligence applications [9]. In Figure 2.1, we can see a knowledge graph that represents entities and their relationships within a specific business context. The graph illustrates connections between a company, its CEO, financial valuation, employee count, and related documents. By organizing data in this manner, KGs retain only the most essential information, eliminating unnecessary details and redundancy.

In the context of RAG models, integrating a KG enhances the model's ability to retrieve and utilize structured information. Unlike traditional RAG models that rely solely on retrieving text chunks from documents, a KG allows the model to search through structured relationships between entities. This approach improves retrieval, especially when answering complex queries requiring information dispersed across multiple documents. By leveraging a KG, the model can more effectively connect relevant concepts, mitigating one of RAG's key limitations, difficulty in synthesizing knowledge that is distributed throughout an entire knowledge base rather than explicitly stated in a single document.

2.6.2 Answer Generation from Knowledge Graph

We will proceed with the assumption that the KG on top of the text documents in our knowledge base has been constructed according to the implementation proposed by the Microsoft team in their paper [8], where they demonstrate how to create a KG with a LLM for this exact task. The construction of a KG is covered in the chapter on implementation 3.8. Therefore, we can now focus on how the constructed KG will be searched to generate the desired answer. Alright, let's rephrase that with a slightly more academic tone, suitable for a bachelor's thesis:

We employ VecDB as our dense retrieval mechanism, operating on the nodes and their associated textual descriptions within the KG. The retrieved nodes serve as initial anchors for graph traversal. For a given query, VecDB identifies the most semantically relevant nodes. Subsequently, we explore the relational structure of the KG, identifying the top k most prevalent internal and external relationships connected to our retrieved nodes. Internal relationships denote connections between the initially retrieved nodes, whereas external relationships link a retrieved node to a node not initially selected. The significance of these relationships is determined by the LLM during the KG construction phase. This provides a heuristic measure for prioritizing potentially relevant connections and additional nodes. Consequently, we aggregate the retrieved nodes and their associated relationships to form a contextual subgraph. To further enrich the context provided to the generation model, we incorporate community summaries associated with each retrieved node, selecting the most pertinent summary. This contextual

information, comprising relevant text chunks, the retrieved subgraph, and community summaries, is then formulated into a prompt for a LLM to answer.

The proposed Graph RAG framework offers several tunable parameters, including the number of internal and external relationships to retrieve, the initial number of nodes retrieved from VecDB, and the quantity of community summaries incorporated. Furthermore, the strategy for selecting community summaries can also be adapted. By capitalizing on this inherent flexibility and the enriched contextual information derived from the KG pre-processing stage, we aim to enhance the efficacy of answer generation, particularly for complex queries that necessitate reasoning over interconnected information.

Chapter 3

Implementation of heuristics

RAG as a question answering model has been explained in the chapter 1 and all of the heuristics we will be implementing were defined in the chapter 2. This chapter will detail the methodology for implementing the aforementioned heuristics and will also discuss the specific use-case on which we tested the RAG approach and the heuristics.

3.1 Introduction to the Project

The project on which we decided to test and improve the RAG model revolves around the creation of sustainability reports for Zurich Insurance customers, specifically large companies rather than individuals. Zurich Insurance focuses on understanding how much its customers strive to regulate their environmental impact. Previously, this task was performed by the sustainability team, which reviewed sustainability reports published by customers and filled out a couple of predetermined questions to provide Zurich Insurance with sufficient information about the given customers' environmental alignment. The team approached us to help automate at least a part of this task and try to fill out the report automatically using our RAG approach. Given that the project allowed us to have a predetermined set of questions and correct answers provided by the sustainability team, it was perfect for testing which heuristic could most improve performance on the given dataset.

3.1.1 Introduction to Provided Data

We were provided with annual PDF reports from seven companies: ABF, BASF, Bosch, BP, Holcim, Marriott, and RWE. These reports, published yearly, detail each company's future targets, current emissions, sustainability projects, associated investments, and other relevant information. However, the reports often contain extensive information, resulting in documents that can span hundreds of pages. The reports include tabular data, which is critical since emissions values we are searching for are frequently

presented in tables. In addition to the PDF files, we received a dataset containing questions and manually provided answers. The dataset includes 21 identical questions for each of the seven companies, totaling 147 questions, which we will use to evaluate our RAG heuristics.

3.2 Base Model Implementation

Our RAG model implementation is entirely written in Python. This section outlines the approach we adopted for developing the simplest RAG model, covering the process from file parsing to question answering.

3.2.1 Parsing and Chunking Strategy

We begin by parsing the provided PDF files. Based on prior experience, we opted to use a single parsing library, confident in its ability to effectively handle tabular data in text format. In the `parsing_model.py` script, we implemented a `Parser` class that leverages the Python library `docling` for parsing individual files. The `docling` library's `convert` function transforms PDFs into `docling` document objects, which can then be exported as Markdown text and stored in our VecDB. The `convert` function employs page layout models to distinguish between raw text and tabular content, applying the optimal parsing strategy for each section while preserving the logical order of the text.

After converting the entire document, we export each page's text into Markdown, effectively segmenting the text by PDF page. However, the resulting text chunks may still be too long for efficient processing. To address this, we developed a custom Markdown text splitter in the `custom_splitter.py` script. This splitter divides the text into chunks based on a specified length, with special handling for tables. When splitting a table, we identify the table header and prepend it to the new chunk to maintain context. We set a maximum chunk length of 1000 characters and store the chunks in a `ChromaDB` VecDB using OpenAI's `text-embedding-3-large` embedding model, which generates embeddings with a vector dimension of 3072. `ChromaDB` integrates seamlessly with our other libraries and handles document retrieval, eliminating the need to explore alternative VecDB options. Each chunk is saved with metadata, including the page number, source file for citation, and company name, to facilitate targeted searches, as we are only interested in answers specific to individual companies.

3.2.2 Question Answering

Our RAG model is defined in the `rag_model.py` script, where the `RAG` class encapsulates the model's question-answering capabilities. The `RAG` class leverages the

`Langchain` library, which streamlines interactions with GPT models and the `ChromaDB` VecDB. Upon initialization, it creates a `ChatOpenAI` object to abstract calls to the GPT model. Over several months, we evaluated multiple OpenAI GPT models for question answering, including `gpt-3.5-turbo`, which was outperformed by `gpt-4`, and further surpassed by `gpt-4o-mini`. The `gpt-4o-mini` model emerged as the optimal choice for our purposes: it is more cost-effective than `gpt-4`, comparably priced to `gpt-3.5-turbo`, outperforms both on OpenAI's benchmark tests, and offers a larger context window.

After loading the VecDB into memory with a filter applied for the specific company being queried, we utilize the `RAG` class's `answer_query` function, which accepts the arguments `query`, `form_of_answer`, `explanation`, `k`, and `heuristic`. The `query` represents the question, `form_of_answer` is a string specifying the desired answer format (e.g., 'number and appropriate explanation'), `explanation` provides additional context for the question (distinct from VecDB text chunks), `k` determines the number of text chunks retrieved from the VecDB, and `heuristic` is a list specifying the heuristics to apply (e.g., ['basic'] for the basic model). After initial trials and consultation with the sustainability team, we adopted the following prompt:

```
"""You are an advanced AI trained to assist with filling in reports
about sustainability of given corporate.
We will give you relevant documents that should give you context
to answer the question.  At the start of each document, you get
info about which file, page and folder the document is from.
Document might be a markdown table or plain text.
We may give you the optimal form of an answer, stick to it.
If you don't know the answer, just say that you don't know, don't
try to make up an answer.
As well try to keep your answer as concise as possible.
Question:  '{question}'
Form:  '{form_of_answer}'
Context:  '{context}'
Explanation of question:  '{explanation}' """
```

Text chunks are retrieved from the VecDB using the similarity search function provided by `Langchain` and incorporated into the prompt, which is then passed to the GPT model to generate the answer. The function returns a dictionary containing the answer and the relevant text chunks.

3.3 Answer Dataset Generation

After developing a basic RAG model, we created a script to iteratively answer all questions using predefined settings. The answers, along with their corresponding questions and retrieved context, are stored in JSON files. To avoid memory issues due to the large data volume, we do not retain all data in memory during script execution. Additionally, to mitigate the risk of incomplete dataset generation caused by potential API call failures, we save results incrementally, as rerunning the script would be costly.

These operations are handled in the `answer_generator.py` script, which generates JSON files containing pandas DataFrames for a given setting. The script uses the `answer_one_company` and `answer_all_companies` functions to populate a designated folder with JSON files for each company. The `create_combined_dataframe` function then creates JSON files for different companies under the same heuristic settings, mapping manual answers and relevant context chunks to the questions and answers. The resulting JSON file can be used as input for the `testing_framework.py` script to evaluate the performance of the specified setting.

3.4 Semantic Chunking Implementation

We explored semantic chunking as a heuristic in 2.2 to enhance the retrieval of relevant context by grouping the text corpus based on semantic meaning. For instance, text discussing a company's alignment with the Paris Climate Agreement might be grouped into a single chunk. However, we adapted our approach to accommodate the unique characteristics of our data.

In the `semantic_chunking` function of the `parsing_generator.py` script, we defined the chunking process. To preserve the integrity of markdown tables, we excluded them from semantic clusters and embedded them as separate chunks, focusing solely on sentence-level semantic chunking. Including tables in clusters would not improve retrieval, as tables are most useful when retained in their entirety.

Instead of using the Punkt Sentence Tokenizer to split text into sentences, we developed a custom function, `split_markdown_sentences_tables`, to process markdown text generated by the `docling` library from PDF files. This function produces two lists: one containing sentences split using the newline character as a delimiter, and another containing markdown tables. This approach prevents tables from being arbitrarily split, which could occur with the Punkt Sentence Tokenizer, leading to loss of contextual information.

We employed the `all-MiniLM-L6-v2` sentence transformer model to generate embeddings for individual sentences. These embeddings were then clustered using the KMeans algorithm from the `sklearn` library. The number of clusters was determined

using the formula:

$$K = \left\lceil \frac{N \times L}{C} \right\rceil$$

where N is the total number of sentences (excluding tables), L is the average sentence length (in characters), and C is the target chunk length (in characters). Consistent with our baseline approach, we set $C = 1000$ to ensure fair comparisons between chunking methods by minimizing variations in chunk length.

After clustering, we combined the sentences into text chunks and stored them in the VecDB alongside the tables obtained from the `split_markdown_sentences_tables` function, using the same embedding model as in the baseline implementation. To adhere to chunk length limits, we verified that each cluster or table did not exceed the maximum length. If a chunk was too long, we split it using functions that preserve table headers in the resulting table chunks.

3.5 Advanced searches implementation

Extending the RAG model from earlier 3.2, we integrate Hybrid Search and HyDE Search to boost retrieval and answering for Zurich Insurance's sustainability reports. Detailed in sections 2.3, 2.4. We explore their deployment and impact next.

3.5.1 Hybrid Search Implementation

Hybrid Search theoretically combines BM25, a keyword-based retrieval method, and vector-based semantic search. Our implementation in the `HybridRetriever` class, however, introduces a practical variation. It begins by employing a dense vector search via `self.vector_store.as_retriever()`, retrieving the top 90 documents most similar to the query. From these 90 documents, a BM25 retriever is constructed using `BM25Retriever.from_documents(docs)`, configured to return the top k documents. The dense retriever's parameters are then updated to retrieve k documents from the entire corpus, and both retrievers are combined using LangChain's `EnsembleRetriever`, which merges their results, via weighted reciprocal rank fusion, with customizable weights (defaulting to [0.5, 0.5]).

This approach deviates from the standard theory by applying BM25 only to a pre-filtered subset of 90 documents rather than the full corpus. This design choice, reduces computational cost, since BM25 on the entire corpus could be resource-intensive. While this may bias results toward documents favored by the dense retriever, it aligns with the hybrid concept of leveraging both methods, offering a trade-off between efficiency and comprehensive ranking.

3.5.2 HyDE Search Implementation

HyDE Search, as described theoretically, enhances retrieval by generating a hypothetical document with a LLM to represent a plausible answer, then using its embedding for similarity search. The `HyDERetriever` class follows this closely. It initializes with an LLM, a vector store, and a filter. The method `generate_hypothetical_document` employs an `LLMChain` with a custom prompt:

```
"You are an expert in sustainability reports analysis. Based on the following question, generate a detailed, informative and concise document that addresses the question comprehensively. If the question asks about emissions it is likely the concise document should contain some markdown table."
```

This prompt tailors the hypothetical document to the sustainability domain, potentially including tables for emissions queries. The `get_relevant_documents` method generates this document, embeds it, and performs a similarity search in the vector store similar to a base model implementation, with the small change of replacing the query with the hypothetical document.

3.6 Reranking Implementation

Our implementation of reranking algorithms can be found in the script `rerankers.py`. We implemented three functions for the `Rerankers` object.

The `cross_encoder` function takes in a list of documents and a query. It creates pairs of the query and each document, then uses the Sentence Transformer model 'ms-marco-MiniLM-L12-v2' from HuggingFace to score the pairs based on their relevance to each other. The output of the scoring function is normalized using the sigmoid function, so the score ranges from 0 to 1. After reordering the given list of text chunks, it returns the top k ranked chunks.

The `gpt` function reorders the text chunks using an LLM call to the 'gpt-4o-mini' model with the following prompt [38]:

```
I will provide you with {num} passages, each indicated by a numerical
identifier []. Rank the passages based on their relevance to the
search query: '{query}'. 'passages_list' Rank the {num} passages
above based on their relevance to the search query.
All the passages should be included and listed using identifiers,
in descending order of relevance. The output format should be
[]. e.g., [4] > [2] > [3].
Only respond with the ranking results, do not say any word or explain.
```

This function implements the sliding window strategy to handle cases where the tokens exceed the context window capacity of approximately $200,000$ tokens for 'gpt-4o-mini'.

The main focus in GPT reranking is list-wise ranking, which allows ranking with an overview of all the documents simultaneously, therefore as we did not rerank as many text chunks to fill up the whole context window, we did not need to enforce the sliding window strategy.

The `peripheral` function adjusts the order of documents to ensure that the more relevant documents are placed at the periphery, addressing the Lost in the Middle problem.

3.7 Chain of Thought Prompting Implementation

As we mentioned in the section about prompt engineering 1.2.5, there is the possibility of letting the GPT model generate extra reasoning before the final answer is established. This may be of great help as the LLM has to tackle a large amount of context at once and answer a complex question from this content. It may be beneficial to first summarize what the question's goal is, what information it received, and only then answer the given question. We implemented these extra instructions into our original prompt to try and get this behavior from the GPT model:

```
Before answering, think through out loud the following steps:
Understand the question and its requirements.
Analyze the context provided to extract relevant information.
Synthesize the information to form a coherent answer.
Your response should be a JSON object with the following structure:
{ 'Thought': 'Your step-by-step thought process here...',
'Answer': 'Your final answer here...' }
Question: {question}
Form: {form_of_answer}
Context: {context}
Explanation of question: {explanation}
```

We did not change the rest of the basic prompt instructions. Additionally, to try and only give the final answer to the testing model, we set the models parameters to generate json. We want only the final answer, as the testing environment might not correctly grade such an answer with the analysis in the beginning. This works most of the time; however, we do not fully rely on GPT answering correctly all the time, so if there is not proper separation between the answer and thinking, we output the whole answer.

3.8 Creation of knowledge graphs for RAG

We will mostly follow Microsoft Graph RAG paper[8] in our implementation of KG construction to ensure our KG has a proven structure that is widely used in Graph RAG implementations, as illustrated in Figure 3.1 we can see that the light blue-colored sections highlight the components involved in the creation of the KG, which we will enhance with deduplication in our implementation. The framework encompasses stages such as text extraction and chunking from source documents, domain-tailored summarization to generate element instances and summaries, and indexing for efficient query processing. During query time, the framework supports query-focused summarization to produce global and community answers, leveraging the structured knowledge within the graph communities.

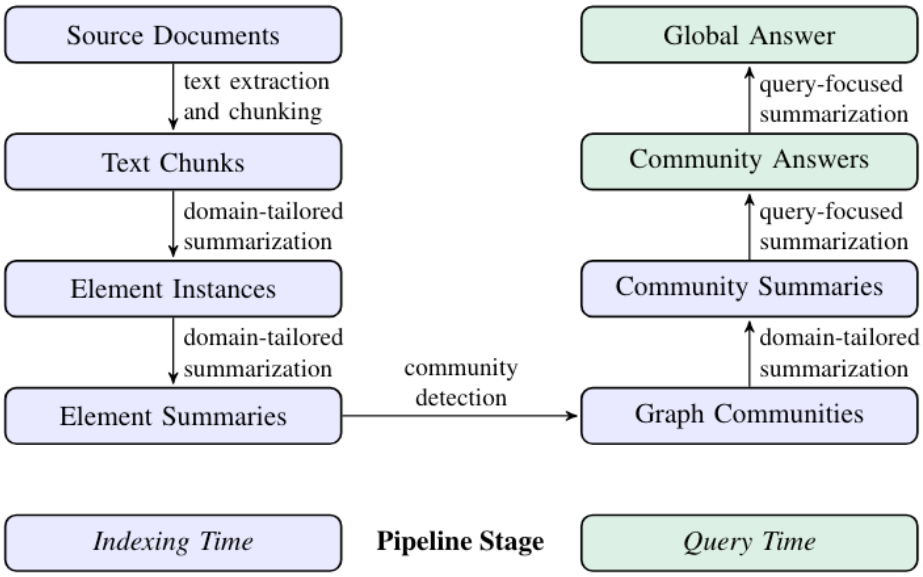

Figure 3.1: Graph RAG Framework proposed by Microsoft [8], illustrating the process of knowledge graph creation and utilization.

3.8.1 Initial Entity Extraction

The pipeline begins with document parsing and chunking using the same approach as our baseline implementation. Following recommendations from [8], we split the text into 1000-characters chunks.

For each chunk, we employ a two-stage LLM prompting strategy, first identification of entity types using a task-specific prompt. Followed by structured entity-relationship extraction with JSON formatting. The JSON schema for entity representation follows:

```
{
    "name": "<entity_name>",
```

```
    "type": "<entity_type>",
    "description": "<entity_description>"
}
```

Relationship extraction produces:

```
{
    "entity1": "<source_entity>",
    "entity2": "<target_entity>",
    "relationship": "<relationship_type>",
    "relationship_strength": <numerical_weight>
}
```

Using asynchronous calls to the Azure OpenAI API we generate such outputs for each text chunk in given file. To ensure we do not overload the API we do this process iteratively over files and not with all of the files at once. We will be saving the outputs in a undirected graph using Networkx library. Additionally in each node we store the name of the file from which given entity originates from, if one entity exists in multiple files we extend the list of files for that entity. Edges are assigned a weight based of the strength of the relationships determined by the LLM.

3.8.2 Deduplication of entities

To address entities that are represented by different names, but essentially represent the same entity. For example "Car Insurance" and "Auto Insurance", both refer to the same type of insurance that covers vehicles, but different companies or regions might use different terminology. To make sure we deduplicate entities we will be employing an embedding model once again. Firstly, we embed the node entity name along with its description and create a similarity matrix, where the value at row i and column j represents the cosine similarity between entity i and entity j. We establish that two entities represent the same entity if their similarity is greater than 0.95, converting the matrix into a binary adjacency matrix with 1s for similar entities and 0s otherwise. The diagonal of this matrix is set to 0 since each entity is inherently connected to itself and does not need to be clustered with itself.

To identify clusters of similar entities, we treat this binary matrix as an adjacency graph, where each node represents an entity, and an edge exists if the similarity threshold is met. Then we apply Breadth-First Search (BFS) to traverse the graph and find connected components. BFS starts from an unvisited node, explores all its directly connected neighbors, and continues expanding outward until all reachable nodes are visited. Each BFS traversal results in a cluster of equivalent entities, allowing us to

effectively merge redundant entities and ensure that relationships between them are properly maintained.

After clustering similar nodes, the KG is restructured by merging these entities into single, consolidated nodes. Each new node retains the combined descriptions and text chunks of the original ones. The graph's relationships are then updated by redirecting all edges from the original nodes to their merged counterparts, ensuring that connections remain intact while eliminating redundancy. If multiple merged entities shared links to the same node, these edges are fused, preserving the most relevant relationship details. Finally, the original nodes are removed, and an integrity check verifies the correctness of the transformation.

3.8.3 Entity summarization

As some entities which have been represented more heavily in text have been extracted by LLM presumably many times they also carry many descriptions stored, some have combination of descriptions from deduplication. Same concept applies to relationships and their descriptions, therefore we will be summarizing both, if the number of tokens their description makes up is greater than 300, we prompt LLM to give concise summary of either the relationship or entity it self. We save up on the amount of data we store while aiming to preserve relevant information [8].

3.8.4 Community detection and community summarization

In large-scale KGs, entities often form communities, groups of nodes that share stronger internal connections compared to their links with the rest of the graph. Identifying these communities helps structure the KG hierarchically, making it more efficient for information retrieval and summarization.

To achieve this, a community detection algorithm is applied to partition the graph into distinct clusters of closely related nodes. One such algorithm, Leiden, is particularly effective because it efficiently identifies hierarchical structures within the graph. The Leiden algorithm works iteratively by optimizing modularity, meaning it seeks to group nodes in a way that maximizes intra-community connections while minimizing external links. It does this by refining partitions through multiple steps: first assigning nodes to provisional communities, then merging and refining these groups based on connectivity strength, and finally stabilizing the structure to ensure high-quality clustering.

Once communities are formed, they are summarized hierarchically. A maximum size is set for leaf-level clusters, ensuring that each cluster remains manageable. Starting from these smallest clusters, summaries are generated by aggregating node descriptions and relationships. This bottom-up summarization continues through multiple levels,

eventually forming high-level summaries that encapsulate broader topics or themes present in the graph.

By structuring information in this way, the KG gains a more interpretable and efficient retrieval mechanism. Instead of navigating individual entities, queries can leverage these community-based summaries to quickly locate relevant information, even when relationships span multiple levels within the graph [8].

3.8.5 Node embeddings

Lastly, the final step in preprocessing data for functional Graph RAG involves generating and storing node embeddings. Similarly to traditional RAG, embeddings are created based on node labels and their descriptions to enable retrieval through cosine similarity with user queries. Embeddings have been stored in Chroma DB vector store for fast retrieval of relevant nodes. We stored all of the properties of node in metadata of each embedding. Finally, the structured KG itself is stored in a JSON format, preserving its relationships and enabling future updates or analysis.

3.8.6 Implementation of Graph-RAG Answering

The `answer_query_kg` method answers queries using a KG following the algorithm we defined in section 2.6.2. Firstly, we select the nodes relevant for given query using basic similarity search. Right after that top internal and external relationships are obtained from the KG for given company. When the graph is traversed and the edges are sorted based on the relationship strength determined by the LLM during the KG creation the newly obtained node metadata is collected. All text chunks which are linked to given nodes are included in the prompt template along with the nodes themselves, their relationships and community summaries. Finally, the GPT model produces an answer as the output.

Chapter 4

Evaluation of heuristics

In this chapter we will look at the results of how much each heuristic we implemented enhanced answers of our model to the prepared set of questions with desired answers and chunks of text that should be retrieved for the model to have the context to answer properly. Question set was prepared partly by the Sustainability team in Zurich Insurance. However as the desired retrieved chunks of text were not in the original dataset provided to us, we had to manually go through the pdf files for each company and cite where the source to the answer the Sustainability team declared to be correct might be found. The performance of the models was naturally not checked manually by a human as it would take tremendous amount of time to compare results for varying models.

Performance of a model can be separated into two parts, one focuses on the retrieval component, how many relevant text chunks the model got, whether the retrieved chunks had the correct chunk within them and so on. The other aspect we rated is the answer itself. We accessed how reliably correct the answer is, how relevant the answer from LLM is to the question and how correct does other LLM call considers the answer based on the manual answer.

We implemented our own testing framework, even though there are publicly available libraries that can be used for this purpose, such as DeepEval and Ragas. We decided it is best to have the insight into how exactly each metric we use is calculated. After short introduction to each metric and how it was implemented, we will look at the process we decided would be the best to choose the final model. Next we will look at how the metrics of each tested model turned out and finally which model was picked by us as optimal for this use-case.

4.1 Evaluation Metrics for RAG Performance Assessment

To comprehensively evaluate the performance of RAG models, we employ a suite of metrics that measure different aspects of the system: *factual consistency*, *semantic alignment*, and *retrieval quality*. These metrics are designed to address the core challenges of RAG systems, including factual accuracy, relevance of generated answers, and the precision of retrieved context. This multi-dimensional approach ensures a robust evaluation framework that aligns with established principles in information retrieval and natural language processing.

4.1.1 Faithfulness: Measuring Factual Consistency

The faithfulness metric evaluates, whether the answers generated by the RAG model are factually supported by the retrieved context. This is crucial for identifying and minimizing "hallucinations", generated content that is not grounded in the provided context. Our implementation follows a systematic approach:

The process begins with statement extraction, where the 'en_core_web_sm' model of `spacy` library is used to break the generated answer into individual statements. Although LLM can be prompted to break up the answer into statements, we choose this approach as it is less time and cost expensive. This step ensures that each factual claim is evaluated independently. Next, in fact-checking with an LLM, each statement is verified against each of the retrieved context chunks separately to ensure the verification is as precise as it can be. The LLM is prompted to classify each statement into one of three categories: "Yes" if the statement is supported by the context, "No" if it contradicts the context, or "Idk" if the context does not provide sufficient information [10]. To shorten the time it takes to evaluate each question we parallelized the calls to the LLM, utilizing the APIs capabilities in combination with `async` python library. Finally, the faithfulness score is calculated as the ratio of supported statements to the total number of statements. This metric is expressed as:

$$\text{Faithfulness} = \frac{\text{Number of supported statements}}{\text{Total number of statements}}$$

This metric provides a clear measure of how well the generated answers aligns with the factual information in the retrieved context.

4.1.2 Answer Relevancy: Assessing Semantic Alignment

Answer relevancy evaluates how well the generated answers align semantically with the original question. This metric is crucial for ensuring that the RAG model produces

responses that directly address the user's query.

The process begins with synthetic question generation, where an LLM generates three potential questions that could have led to the given answer. In addition we provide the LLM with the manual answer as well to ensure that if a given answer of the model is just one word for example 'Yes' the synthetically generated questions would not be generated completely randomly. Next, in semantic similarity calculation, we compute the cosine similarity between the embeddings of the generated questions and the original question [10]. Embeddings are generated using a pre-trained sentence embedding model, such as Sentence-BERT. Finally, the answer relevancy score is determined by averaging the cosine similarity across all generated questions. This metric is expressed as:

$$\text{Answer Relevancy} = \frac{1}{3}\sum_{i=1}^{3} \text{cosine\_similarity}(q_i, q_{\text{original}})$$

where q_i represents the generated questions, and q_{original} is the original question.

This metric ensures that the generated answers are not only factually correct but also semantically aligned with the user's intent.

4.1.3 Context Relevancy: Evaluating Retrieval Precision

Context relevancy assesses the quality of retrieved context chunks by measuring how many sentences within them are actually relevant to the question. Adjusting model to this metric should ensure the model is fed less irrelevant information, that should reduce hallucinations, as well we would be saving tokens by not putting in irrelevant context.

To determine relevancy, the LLM identifies which sentences in the retrieved context chunks are relevant to the question. It either extracts specific relevant sentences or indicates "insufficient information" if none are found [10]. Again calls to the API are parallelized in our implementation, to save up on time. The context relevancy score is calculated as the ratio of relevant sentences to the total number of sentences in the retrieved chunks:

$$\text{Context Relevancy} = \frac{\text{Number of relevant sentences}}{\text{Total number of sentences}}$$

This metric offers valuable insight into the precision of the retrieval mechanism, ensuring that the system retrieves context that is genuinely useful for answering the question.

4.1.4 Correctness of Answer: LLM-Based Scoring

In our evaluation, we have not yet utilized the manually provided answers from the Sustainability team, focusing instead on the text that should be retrieved from the given documents. To measure how much the answer of our model aligns, we decided to employ a GPT model to rate the accuracy of the model's responses. The GPT model scores the correctness of the model's answer on a scale from 0 to 1, based on the manual answer.

The prompt we used was:

```
You are an expert RAG evaluator. Below are two answers for the
same question:
Question: '{question}'
Ground truth: '{manual_answer}'
Model answer: '{model_answer}'
Evaluate the correctness of the model's answer compared to the
ground truth. Consider factual accuracy, relevance, and completeness.
If the ground truth indicates that something is not mentioned,
a negative answer is at least partially correct.
Provide a score from 0 to 1 where 1 is perfectly correct and 0
is completely incorrect. Return only the score.
```

Although this approach is not deterministic, it enables us to utilize the manual answer to some extent. When we tested the model on a sample size of 147 questions, we assumed that the average of this metric would indicate a trend, showing when our model outperforms other models in terms of how its answers align with human-filled reports.

4.1.5 Precision, Recall, and F_1 Score: Traditional Information Retrieval Metrics

In addition to leveraging LLMs to assess answer quality and context relevance, traditional information retrieval (IR) metrics play a crucial role in evaluating the retrieval component of the RAG system. These standardized IR metrics provide a well-established framework for measuring the effectiveness of retrieval, ensuring the system retrieves the most relevant information efficiently [30].

Precision quantifies the accuracy of the retrieved chunks by measuring the proportion of retrieved text that is actually relevant to the given query. High precision indicates that most of the retrieved information is useful, while low precision suggests that many retrieved chunks are irrelevant [30]. It is computed as follows:

$$\text{Precision} = \frac{|\text{Relevant text} \cap \text{Retrieved text}|}{|\text{Retrieved text}|}$$

A high precision score means fewer irrelevant chunks are retrieved, reducing noise in the response generation. However, focusing solely on precision can lead to missing relevant information, which is why it is balanced with recall.

Recall measures the effectiveness of retrieval by evaluating how much of the total relevant information has been successfully retrieved. High recall indicates that most relevant chunks have been retrieved, while low recall suggests that important pieces of information may have been missed [30]. The formula for recall is:

$$\text{Recall} = \frac{|\text{Relevant text} \cap \text{Retrieved text}|}{|\text{Relevant text}|}$$

A high recall score ensures that the system does not overlook crucial information, but optimizing solely for recall may lead to retrieving excessive amounts of irrelevant text, reducing precision.

To balance both precision and recall, the F_1 score is used. It is the harmonic mean of precision and recall, providing a single metric that accounts for both aspects [30].

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

A higher F_1 score indicates a good balance between precision and recall, making it a useful metric for evaluating retrieval performance holistically.

To accurately assess relevance for computing precision and recall, a robust method is needed to determine whether a retrieved text chunk sufficiently aligns with the ground truth text. Given the variability in document parsing and the definition of relevant text in our answer dataset, we employ the Jaccard score, also known as the Jaccard similarity coefficient, as a similarity metric [13]. This metric quantifies the overlap between two sets by calculating the ratio of the size of their intersection to the size of their union.

In our implementation, we enhance granularity by splitting both the retrieved and ground truth texts into sentences and computing the Jaccard score based on matching sentence pairs. This approach accommodates differences in text segmentation while capturing meaningful content overlap. The Jaccard score is defined as:

$$\text{Jaccard Score} = \frac{|\text{Relevant} \cap \text{Retrieved}|}{|\text{Relevant} \cup \text{Retrieved}|}$$

A higher Jaccard score reflects greater similarity between the retrieved and relevant texts. We establish a threshold of 0.7, such that if the Jaccard score between a retrieved chunk and a ground truth chunk exceeds this value, the retrieved chunk is deemed relevant for precision and recall calculations. This threshold ensures a balance

between strictness and flexibility in matching. By integrating the Jaccard score into our evaluation pipeline, we prioritize substantial content overlap over superficial keyword matches, thereby improving the reliability of our retrieval assessment in the RAG system.

4.2 Strategy for finding an optimal model

To determine the optimal model settings for our RAG system, we adopt a systematic methodology that balances performance with practical constraints. This approach ensures the selected model excels in retrieval and response generation while remaining resource-efficient and feasible for implementation.

We optimize the RAG system through an iterative, bottom-up process, where each component is tested and refined sequentially while building upon the evolving base configuration. The testing sequence proceeds as follows: First, we test fundamental parameters, such as the number of documents retrieved, to balance retrieval coverage and relevance in this step. Next, we refine the chunking strategy to identify the most effective method for segmenting documents into retrievable units.

Then we explore all of the various heuristics with the predefined settings we found to be best performing for the basic RAG approach. When we finish comparing the heuristics independently against the base model we try to utilize the flexibility of the system to try combination of best performing heuristics to get even better results in the evaluation metrics. When evaluating the performance of the models we mostly focus on what metric we believe given heuristic should improve for example when testing rerankers we aim for improvements in IR metrics improve against the baseline configuration and when testing CoT approach we expect change only in the answer related metrics.

To ensure the reliability of these enhancements, we will conduct a statistical significance analysis, comparing each enhanced model´s results of the evaluation metrics against a baseline RAG model metrics. We will be measuring whether the change in the given metric is significant according to appropriate tests either paired t-tests if metrics data is normally distributed or if not we use the Wilcoxon signed-rank tests, thus identifying models with meaningful gains in performance.

For models showing significant improvements, we perform a granular analysis at the question level, examining evaluation metrics for individual questions to discern, whether certain models excel on specific question types, potentially informing a mixed-model strategy, where distinct models are deployed based on query characteristics to enhance overall performance.

We assess practicality by comparing costs, like token usage and report generation

time, to ensure advanced methods with small benefits do not use too many resources, balancing effectiveness and efficiency.

To complement these quantitative findings, we conduct a qualitative assessment of top-performing models, manually reviewing responses to examine differences in answer quality not captured by metrics, verifying that the selected model delivers coherent, contextually relevant, and high-quality outputs in practice.

Upon completing these steps, we form a decision based on these results to select the optimal model, which demonstrates robust performance across evaluation metrics, exhibits statistically significant improvements, and maintains a practical balance between efficacy and resource demands, making it suitable for usage by the Sustainability team to generate customer reports efficiently and effectively.

4.3 Quantitative Performance Evaluation of Models

In this section we will go through the results of the evaluation metrics for each of the tested model following the procedure we established in the previous section 4.2

4.3.1 Number of Documents Retrieved Evaluation

We compare performance metrics for different numbers of retrieved text chunks (k) to determine the optimal value for our RAG system.

# Chunks Retrieved	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
4	0.6010	0.7800	0.5980	0.1768	0.0489	0.1041	0.0607
6	0.6217	0.7946	0.5880	0.1789	0.0473	0.1532	0.0672
8	0.6392	0.8087	0.5849	0.1565	0.0409	0.1715	0.0619
10	0.6307	0.8100	0.5863	0.1552	0.0369	0.2007	0.0589
12	0.6278	0.8090	0.5758	0.1415	0.0338	0.2156	0.0553
30	0.6644	0.8506	0.5646	0.1080	0.0179	0.3116	0.0331

Table 4.1: Averages of metrics for different numbers of text chunks retrieved from the database, with chunk length set to 1000 characters and basic similarity search performed in the VecDB.

Analysis of Table 4.1 shows that Correctness and Recall increase with higher k, reflecting broader information coverage, while Precision declines due to more irrelevant chunks. The F_1 score peaks at $k = 6$, offering an effective balance. We selected $k = 6$ for its strong performance and practicality, as 30 documents are significantly harder for the Sustainability team to fact-check when compiling reports. However, $k = 30$ sets a benchmark with the highest Correctness and Faithfulness scores.

4.3.2 Chunking Strategy Evaluation

We tested three chunking strategies for our RAG system: basic chunking (1000-character limit or markdown table segmentation), enriched chunking with metadata, and semantic chunking with clustered sentences and separated tables. Responses were generated by retrieving six documents using basic similarity search. Averages of performance metrics are shown in Table 4.2.

Chunking Strategy	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
Enriched-6	0.6459	0.8189	0.5612	0.1780	0.0474	0.1594	0.0691
Semantic-6	0.6134	0.7951	0.6076	0.1495	0.0251	0.1534	0.0407
Basic-6	0.6236	0.7977	0.5905	0.1787	0.0471	0.1526	0.0670

Table 4.2: Averages of evaluation metrics for different chunking strategies, with six text chunks retrieved using basic similarity search on each of the VecDBs.

Enriched chunking outperformed basic chunking slightly in Correctness, Faithfulness, and F_1 score, while semantic chunking lagged except for a small gain in answer relevancy. We chose enriched chunking for its balanced performance.

4.3.3 Advanced Searches Evaluation

We evaluated advanced search strategies using a VecDB with metadata-enriched chunks, retrieving six documents per query. Tested techniques included HyDE, hybrid search with equal weights (0.5 for similarity and BM25), and hybrid search with BM25 weighted at 1.0, compared to basic similarity search. Performance metrics are shown in Table 4.3.

Search Technique	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
Basic-6	0.6459	0.8189	0.5612	0.1780	0.0474	0.1594	0.0691
Hybrid05-6	0.6712	0.8191	0.5807	0.1395	0.0363	0.1410	0.0541
Hybrid1-6	0.6381	0.7267	0.5755	0.0950	0.0334	0.1172	0.0461
HyDE-6	0.6425	0.8120	0.5622	0.1638	0.0446	0.1730	0.0665

Table 4.3: Averages of evaluation metrics for different search strategies, with six text chunks retrieved using each search technique on metadata-enriched text chunks.

Hybrid search with equal weights slightly improved Correctness (0.6712), while HyDE increased Recall (0.1730) but reduced Precision (0.0446). Faithfulness peaked with basic similarity search (0.8189), and F_1 Scores were slightly lower for advanced methods. Given these mixed results, we will proceed with basic similarity search for further heuristic testing and later explore interactions between search techniques and heuristics.

4.3.4 Reranking Evaluation

We evaluated three reranking methods—Cross-Encoder, GPT, and peripheral—using a VecDB with metadata-enriched chunks. For Cross-Encoder and GPT, we retrieved 30 chunks via similarity search and selected the top 6 after reranking. The peripheral method retrieved only 6 chunks initially. Performance metrics are presented in Table 4.4.

Reranking Method	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
Basic-6	0.6459	0.8189	0.5612	0.1780	0.0474	0.1594	0.0691
Peripheral-6	0.6408	0.8075	0.5649	0.1868	0.0474	0.1594	0.0691
Cross-Encoder-6	0.6569	0.8468	0.5538	0.1626	0.0550	0.1906	0.0793
RerankGPT-6	0.6932	0.8184	0.5666	0.2152	0.0585	0.2253	0.0882

Table 4.4: Average evaluation metrics for reranking heuristics. For Cross-Encoder and GPT rerankers, 30 text chunks were initially retrieved using basic similarity search augmented with metadata-enriched text chunks. The top 6 ranked chunks were used to generate responses. In the peripheral heuristic, only 6 text chunks were retrieved initially.

The peripheral method showed no significant improvement over the baseline. The Cross-Encoder slightly enhanced Faithfulness (0.8468) and Correctness (0.6569) but reduced Context Relevancy. The GPT reranker outperformed others in both the IR metrics which we value the most, however its dominance showed up in Answer related metrics too, achieving the highest Correctness (0.6932) and F_1 Score (0.0882), indicating superior relevance detection.

4.3.5 Chain of Thought Prompting Evaluation

We assessed Chain of Thought (CoT) prompting using a similarity search on an enriched VecDB, retrieving six text chunks, to evaluate its impact on response quality compared to a basic approach. Metrics are shown in Table 4.5.

Prompt Used	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
CoT-6	0.6841	0.8061	0.5307	0.1826	0.0472	0.1594	0.0689
Basic-6	0.6459	0.8189	0.5612	0.1780	0.0474	0.1594	0.0691

Table 4.5: Average evaluation metrics for Chain of Thought prompting. Six text chunks, enriched with metadata, were retrieved via similarity search and used to generate responses.

CoT improved Correctness (0.6841 vs. 0.6459) but slightly lowered Answer Relevancy and Faithfulness, possibly due to thought process inclusion. Precision, Recall, and F_1 Score were similar. Manual response comparison will further explore quality improvements. The increase in Correctness highlights CoT's ability to enhance answer

accuracy. However, the trade-off in Relevancy and Faithfulness suggests a need for refining the reasoning steps included.

4.3.6 Graph RAG Evaluation

We compare the Graph RAG approach (KG-8 and KG-12) against the baseline RAG model (Basic-6) to assess whether knowledge graphs enhance response quality. In this methodology, a preconstructed KG was utilized for each company to generate sustainability reports. Key parameters were defined, including the initial number of nodes to retrieve and the number of internal and external relationships to explore. Specifically, we tested the KG approach by retrieving 12 nodes initially, supplemented by 12 internal and 12 external relationships.

Heuristic	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
Basic-6	0.6459	0.8189	0.5612	0.1780	0.0474	0.1594	0.0691
KG-8	0.7202	0.8218	0.5683	0.0391	0.0156	0.3152	0.0288
KG-12	0.7120	0.8008	0.5432	0.0295	0.0128	0.3620	0.0241

Table 4.6: Average evaluation metrics for the Graph RAG heuristic. For KG-8 and KG-12, 8 and 12 nodes, respectively, were initially retrieved via similarity search, augmented with 12 internal and 12 external relationships, including associated nodes, text chunks, and community summaries. These results are compared to the baseline model with 6 retrieved text chunks.

Table 4.6 shows that KG-8 and KG-12 improve Correctness (0.7202 for KG-8) and Recall (0.3620 for KG-12) but reduce Precision (0.0128 for KG-12) and Context Relevancy (0.0295 for KG-12), leading to lower F_1 scores (0.0241 for KG-12) compared to Basic-6 (0.0691). The larger context of KG-12 is harder for the Sustainability team to fact-check when compiling reports. Since heuristics like CoT prompting achieve similar gains without extensive contexts, we prioritize those for further evaluation.

4.3.7 Interactions Between Heuristics Evaluation

Following the independent evaluation of each heuristic, we sought to investigate the potential synergistic effects of combining multiple heuristics to determine whether such combinations could yield enhanced performance. The hypothesis was that certain heuristic pairings might complement one another, resulting in greater improvements in response quality.

Chain of Thought Prompting with GPT Reranker Evaluation

We evaluate the combination of CoT mprompting and GPT reranking to assess its performance. This is compared against their independent uses to determine if synergy

enhances response quality.

Heuristic	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
RerankGPT-6	0.6932	0.8184	0.5666	0.2152	0.0585	0.2253	0.0882
CoT-6	0.6841	0.8061	0.5307	0.1826	0.0472	0.1594	0.0689
CoT-RerankGPT-6	0.6629	0.7846	0.5552	0.2211	0.0595	0.2286	0.0901

Table 4.7: Averages of evaluation metrics for the combination of Chain of Thought prompting and RerankGPT. Six text chunks, enriched with metadata, were retrieved via similarity search and used to generate responses.

Table 4.7 indicates that combining CoT and GPT reranking (CoT-RerankGPT-6) yields lower Correctness (0.6629) and Faithfulness (0.7846) than RerankGPT-6 (0.6932, 0.8184) or CoT-6 (0.6841, 0.8061), suggesting no synergistic benefit. For the IR metrics we see that Precision (0.0595) and Recall (0.2286) improve slightly, which we believe is purely random as the retrieval process was the same for both of the compared models.

Chain of Thought Prompting with Graph RAG Evaluation

Aiming to weave the strengths of structured reasoning into the knowledge graphs, we paired CoT prompting with Graph RAG. This combination is pitted against standalone KG configurations to uncover any leaps in response quality.

Heuristic	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
KG-8	0.7202	0.8218	0.5683	0.0391	0.0156	0.3152	0.0288
KG-12	0.7120	0.8008	0.5432	0.0295	0.0128	0.3620	0.0241
CoT-KG-8	0.6707	0.7369	0.5705	0.0387	0.0154	0.3195	0.0285
CoT-KG-12	0.7156	0.7548	0.5786	0.0410	0.0123	0.3535	0.0232

Table 4.8: Averages of evaluation metrics for Chain of Thought prompting combined with Graph RAG heuristics. For CoT-KG-8 and CoT-KG-12, 8 and 12 nodes, respectively, were initially retrieved, each supplemented with 12 internal and 12 external relationships in the final prompt.

Table 4.8 reveals that blending CoT with Graph RAG does not perform as well as expected. Correctness for CoT-KG-12 (0.7156) improves sligthly past KG-12 (0.7120) but falls shy of KG-8's peak (0.7202), while Faithfulness tumbles for CoT-KG-8 (0.7369) and CoT-KG-12 (0.7548) against KG-8 (0.8218) and KG-12 (0.8008). A spark of promise shines in Answer Relevancy, climbing to 0.5705 for CoT-KG-8 and 0.5786 for CoT-KG-12. The impact of CoT prompting appears to differ between basic similarity search and KG-based approaches, likely due to the overwhelming volume of context in KGs, which may challenge the efficacy of the CoT strategy. These findings suggest that the integration of CoT with Graph RAG does not yield the anticipated synergistic benefits.

GPT Reranking with Different Search Strategies Evaluation

Seeking to boost GPT reranking's prowess, we paired it with Hybrid and HyDE search strategies. These combinations are weighed against their standalone performances and the baseline RerankGPT-6 to gauge any uplift in response quality.

Heuristic	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
RerankGPT-6	0.6932	0.8184	0.5666	0.2152	0.0585	0.2253	0.0882
Hybrid05-6	0.6712	0.8191	0.5807	0.1395	0.0363	0.1410	0.0541
Hybrid05-RerankGPT-6	0.6993	0.7976	0.5826	0.2015	0.0568	0.2215	0.0858
HyDE-6	0.6425	0.8120	0.5622	0.1638	0.0446	0.1730	0.0665
HyDE-RerankGPT-6	0.6939	0.8183	0.5821	0.2203	0.0549	0.2229	0.0828

Table 4.9: Average evaluation metrics for the interactions of Hybrid search with RerankGPT and HyDE search with RerankGPT, compared to the independent application of these heuristics.

Table 4.9 shows that blending GPT reranking with Hybrid and HyDE search slightly improves IR metrics forward, with Hybrid05-RerankGPT-6 hitting Precision (0.0568) and Recall (0.2215), and HyDE-RerankGPT-6 reaching Precision (0.0549) and Recall (0.2229), outpacing their independent runs. Yet, these pairings barely outshine the baseline RerankGPT-6, which holds strong with Correctness (0.6932), Faithfulness (0.8184), and F_1 (0.0882). The integration of GPT reranking with alternative search strategies thus produced only marginal differences, suggesting that traditional similarity search remains a robust foundation for GPT reranking.

Evaluation of GPT Reranking with KG

To refine the rich but hefty contexts of Graph RAG, we paired GPT reranking with KG-8. This combination is compared to standalone KG-8 to assess if reducing the number of text chunks boosts response quality.

Heuristic	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
KG-RerankGPT-8	0.7277	0.8298	0.5642	0.0472	0.0180	0.2650	0.0328
KG-8	0.7202	0.8218	0.5683	0.0391	0.0156	0.3152	0.0288

Table 4.10: Average evaluation metrics for the interaction between Graph RAG with 8 initially retrieved nodes and GPT reranking, which selects the top 12 text chunks for inclusion in the prompt.

Table 4.10 shows that GPT reranking sharpens KG-8's edge, lifting Correctness to 0.7277 from 0.7202 and Faithfulness to 0.8298 from 0.8218, though Answer Relevancy dips slightly to 0.5642 from 0.5683. Recall holds steady at 0.2650 despite fewer chunks, proving the potency of 12 well-chosen texts, yet the F_1 score rises modestly to 0.0328.

Evaluation of Chain of Thought Prompting with GPT Reranking and Different Search Strategies

Despite the suboptimal performance of CoT prompting when combined solely with GPT reranking, we sought to further investigate its efficacy by integrating it with advanced retrieval strategies. This evaluation aimed to determine whether CoT prompting could enhance performance when paired with Hybrid search and HyDE search in conjunction with GPT reranking.

Heuristic	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
Hybrid05-RerankGPT-6	0.6993	0.7976	0.5826	0.2015	0.0568	0.2215	0.0858
CoT-RerankGPT-Hybrid05-6	0.6427	0.7557	0.5554	0.1992	0.0575	0.2234	0.0870
HyDE-RerankGPT-6	0.6939	0.8183	0.5821	0.2203	0.0549	0.2229	0.0828
CoT-HyDE-RerankGPT-6	0.7027	0.8466	0.5526	0.2160	0.0569	0.2345	0.0866

Table 4.11: Averages of evaluation metrics for the interactions of advanced retrieval strategies (Hybrid and HyDE) with RerankGPT and CoT prompting, using 6 retrieved text chunks, compared to the same strategies without CoT prompting.

The results, as presented in Table 4.11, reveal divergent outcomes for the two combinations. The integration of CoT prompting with Hybrid search and GPT reranking (CoT-RerankGPT-Hybrid05-6) resulted in an overall decline in performance, with reductions in Correctness (0.6427 versus 0.6993), Faithfulness (0.7557 versus 0.7976), and Answer Relevancy (0.5554 versus 0.5826) compared to Hybrid05-RerankGPT-6. In contrast, the combination with HyDE search (CoT-HyDE-RerankGPT-6) yielded slight improvements across several metrics, including Correctness (0.7027 versus 0.6939), Faithfulness (0.8466 versus 0.8183), though Answer Relevancy saw a minor decrease (0.5526 versus 0.5821). These findings, detailed in Table 4.11, suggest that while the addition of CoT prompting to Hybrid search did not enhance performance, its integration with HyDE search and GPT reranking produced marginal gains. This indicates that the inclusion of a third heuristic can, in some cases, positively influence outcomes, though inherent variability in the metrics may also contribute to these results.

4.4 Statistical Significance Analysis of Model Improvements

According to the strategy outlined in Section 4.2, we evaluate the statistical significance of the results for the model identified as the "best performing" based on the average of selected metrics. This analysis is necessary due to the volatility in our testing outcomes, primarily caused by the incomplete context dataset and the probabilistic nature of LLM-based metrics, which introduce variability and challenge confidence in observed improvements.

The PDF files employed frequently distribute answers to a single question across multiple sections, making it challenging to encompass all relevant sources within the dataset. Moreover, some desired responses are not explicitly articulated in the text, needing deductive reasoning by the model. For example, consider the question:

```
Are the company's targets assessed by Climate Action 100?
```

In the absence of relevant context, the model must deduce a negative response, implying that the company is not assessed. However, such questions lack identifiable correct context within the dataset, thereby worsening performance on IR metrics. Additionally, for certain queries, retrieving a single text chunk is enough, negating the need to access all available chunks in the dataset. Another limitation lies in the Correctness metric's inherent variability, as the LLM assigns a correctness rating to responses. We mentioned in Section 1.2.1, LLMs predict subsequent tokens probabilistically, which, while effective to a degree, lacks a clear line between an answer proclaimed as 80% correct versus one at 70% correct. This undermines confidence in the metric's ability to reflect genuine improvements in answer quality. Now our objective is to assess the statistical significance of the results for the model identified as the "best performing" based on the average of selected metrics.

Despite a dataset comprising 147 records—typically adequate for metric evaluation—the elevated standard deviations observed show a need for more rigorous statistical analysis. To this end, we perform a two-sample t-test across all metrics to compare the basic similarity search model, utilizing 6 text chunks, against the Graph RAG model, which initially retrieves 8 nodes. The Graph RAG model exhibited enhanced performance in answer-related metrics and recall relative to the basic similarity search approach, positioning these metrics as prime candidates for significant differences.

Table 4.12 displays the average values for the evaluation metrics of both models, for which we will be testing the significance of improvements of given metrics.

Heuristic	Correctness	Faithfulness	Answer Relevancy	Context Relevancy	Precision	Recall	F_1 Score
basic-6 averages	0.6459	0.8189	0.5612	0.1780	0.0474	0.1594	0.0691
KG-8 averages	0.7202	0.8218	0.5683	0.0391	0.0156	0.3152	0.0288

Table 4.12: Averages of evaluation metrics for the basic similarity search model (using 6 text chunks) and the Graph RAG model (initially retrieving 8 nodes).

We first conducted a normality test for each of the metrics using the Kolmogorov-Smirnov (KS) test from the `scipy.stats` module [12], which results can be seen in 4.13. Our analysis revealed that only the Answer relevancy metric was normally distributed. The enhanced table below presents these results with improved readability:

Since only the 'answer relevancy' metric was found to be normally distributed, we employed the Wilcoxon signed-rank test for the remaining metrics to compare the per-

Metric	Statistic (basic-6)	p-value (basic-6)	Distribution (basic-6)	Statistic (KG-8)	p-value (KG-8)	Distribution (KG-8)
Correctness	0.1698	0.0004	Not Normal	0.2057	<0.0001	Not Normal
Faithfulness	0.4341	<0.0001	Not Normal	0.4453	<0.0001	Not Normal
Answer Relevancy	0.0686	0.4731	Normal	0.0640	0.5616	Normal
Context Relevancy	0.1725	0.0003	Not Normal	0.1654	0.0006	Not Normal
Precision	0.3766	<0.0001	Not Normal	0.3185	<0.0001	Not Normal
Recall	0.3886	<0.0001	Not Normal	0.3476	<0.0001	Not Normal
F_1 Score	0.3877	<0.0001	Not Normal	0.3379	<0.0001	Not Normal

Table 4.13: Normality test results for the basic similarity search model (basic-6) and the Graph RAG model (KG-8). The distribution is classified as 'Normal' if the p-value from the KS test exceeds 0.05; otherwise, it is 'Not Normal'.

formance of the two models. The Wilcoxon signed-rank test serves as a non-parametric alternative to the paired t-test, appropriate when data do not satisfy the normality assumption required by the t-test. This test evaluates whether the median difference between paired observations from the two models is zero, making it well-suited for our metrics, which are either ordinal or lack normal distribution [31]. It operates by ranking the absolute differences between paired observations, assigning signs based on the direction of the difference, and calculating a W-statistic as the sum of the ranks for positive or negative differences, whichever is smaller [31]. For samples with more than 10 paired observations, as in our dataset of 147 records, the W-statistic approximates a normal distribution, enabling the computation of p-values to assess statistical significance.

	Correctness	**Faithfulness**	**Answer Relevancy**	**Recall**
W-Statistic	824.5	803.0	5896.0	222.0
p-value	0.0003	0.5163	0.8116	<0.0001
Reject Null Hypothesis	Yes	No	No	Yes

Table 4.14: One sided Wilcoxon signed-rank test results comparing the basic similarity search model (basic-6) and the Graph RAG model (KG-8) across various metrics. The null hypothesis, stating no improvement of Graph RAG against basic model, it is rejected if the p-value is less than 0.0125.

The one-sided Wilcoxon signed-rank test results in 4.14 reveal significant differences for certain metrics on level of significance $\alpha = \frac{0.05}{4}$ as we test with a correction and only 4 of the metrics have a possibility to be significantly greater for the Graph RAG approach. For Correctness, the W-statistic is 824.5 with a p-value of 0.0003, below the 0.05 threshold, leading us to reject the null hypothesis. This confirms that the Graph RAG model (KG-8) significantly outperforms the basic similarity search model (basic-6) in generating more correct responses (averages: 0.7202 vs. 0.6459). Similarly, for Recall, the W-statistic of 222.0 and a p-value less than 0.0001 indicate a significant difference, with KG-8 retrieving a higher proportion of relevant context (averages:

0.3152 vs. 0.1594).

For Faithfulness, Answer Relevancy, the p-values exceed 0.05 indicating no statistically significant differences in the direction tested. Despite some variation in average values, these high p-values, coupled with substantial standard deviations, suggest that any observed differences are likely due to random variation rather than consistent model superiority in the hypothesized direction.

We extended the same Wilcoxon framework to other configurations. The independent reranking with LLM significantly outperformed the base model in Correctness, Context Relevancy, Precision, Recall, and F_1 Score. The CoT approach improved Correctness significantly ($p = 0.0228$), while `CoT_hyde_rerankGPT` enhanced Correctness, Context Relevancy, Recall, and F_1 Score ($p < 0.05$) compared to the base model. Lastly we tested whether the Graph RAG approach with 8 initially retrieved nodes significantly outperforms the `CoT_hyde_rerankGPT` model for all of the metrics on the level of significance of $\alpha = \frac{0.05}{7}$ and it only improved the Recall significantly, as the Graph RAG uses way more context in answering it was expected. The conclusion for us is that the Graph RAG might offer better responses than the base model but it is not significantly better overall in comparison to other heuristics.

4.5 Granular Analysis of Model Performance by Question

Following the strategy outlined in Section 4.2, we now examine the performance of models on individual questions to identify specific strengths and weaknesses. This analysis is motivated by the possibility that certain questions may be better suited to specific heuristics and the observed variability in aggregate metrics, which may mask question-level differences.

4.5.1 Analysis of GPT Reranking Against Basic Model Across Questions

Among the comparisons conducted, the most notable performance enhancements were observed when evaluating the rerankGPT model against the basic model. Let us now examine the metric comparisons between these two models to identify the specific areas where rerankGPT achieved its improvements.

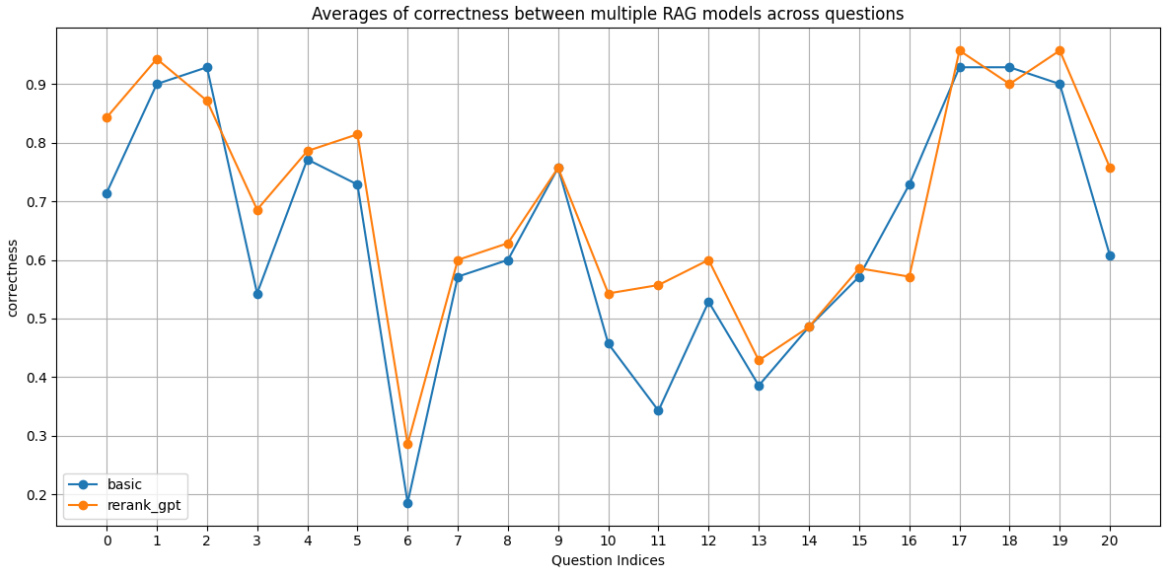

Figure 4.1: Average of Correctness metric across different questions for the GPT reranking model and the basic model.

Figure 4.1 illustrates the fluctuations in the average Correctness metric across various questions. Both models consistently achieve high accuracy for questions 1, 2, 17, 18, and 19, which may be classified as relatively straightforward. These questions typically pertain to information readily available in sustainability reports, such as Scope 1 and Scope 2 emissions targets and their validation by the Science Based Target initiative.

Significant improvements for the GPT reranker are evident in questions 3, 11, and 20. Conversely, the basic model outperforms the GPT reranker in question 16, exhibiting a substantial difference in the Correctness metric. Question 3 addresses Scope 3 targets, which are less frequently detailed in reports. The retrieval process may prioritize text chunks discussing Scope 1 and Scope 2 targets due to their repeated mention, leading to a mismatch. This confusion arises when the model incorporates context related to similar yet distinct targets, resulting in responses that address different emission scopes. The GPT reranker mitigates this issue by recognizing the greater relevance of chunks pertaining to Scope 3 targets.

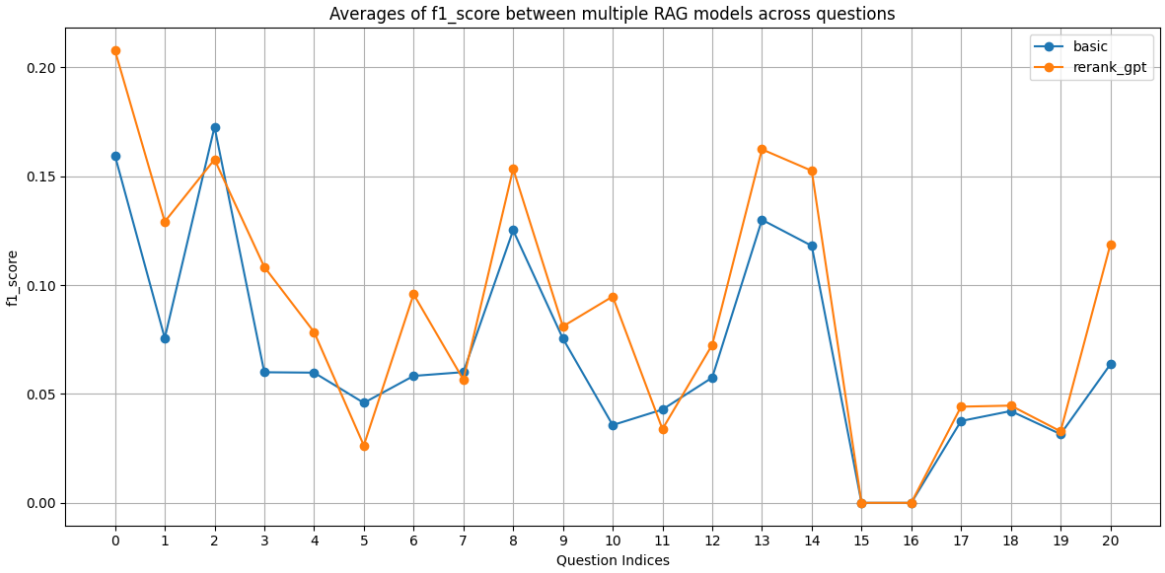

Figure 4.2: Average of F_1 score metric across different questions for the GPT reranking model and the basic model.

The superiority of the GPT reranker extends beyond the Correctness metric to include IR metrics. Figure 4.2 highlights the effectiveness of the GPT reranker, which surpasses the basic model in 19 out of 21 questions. The questions previously identified as exhibiting the most substantial gains in Correctness—namely questions 3 and 20—also demonstrate higher average F_1 scores with the GPT reranker. However, this trend does not consistently apply to question 11, where the improvement in F_1 score is less pronounced. This discrepancy may be attributed to the enhanced value of the context retrieved, which likely aids the model in formulating more accurate responses.

4.5.2 Analysis of Chain of Thought Against Basic Model Across Questions

The CoT prompting approach demonstrates notable differences in answer-related metrics when compared to the basic similarity search model. Since the CoT model employs the same retrieval method as the basic model, the IR metrics remain unchanged. Therefore, our analysis focuses on answer quality, specifically the Correctness metric, as illustrated in Figure 4.3.

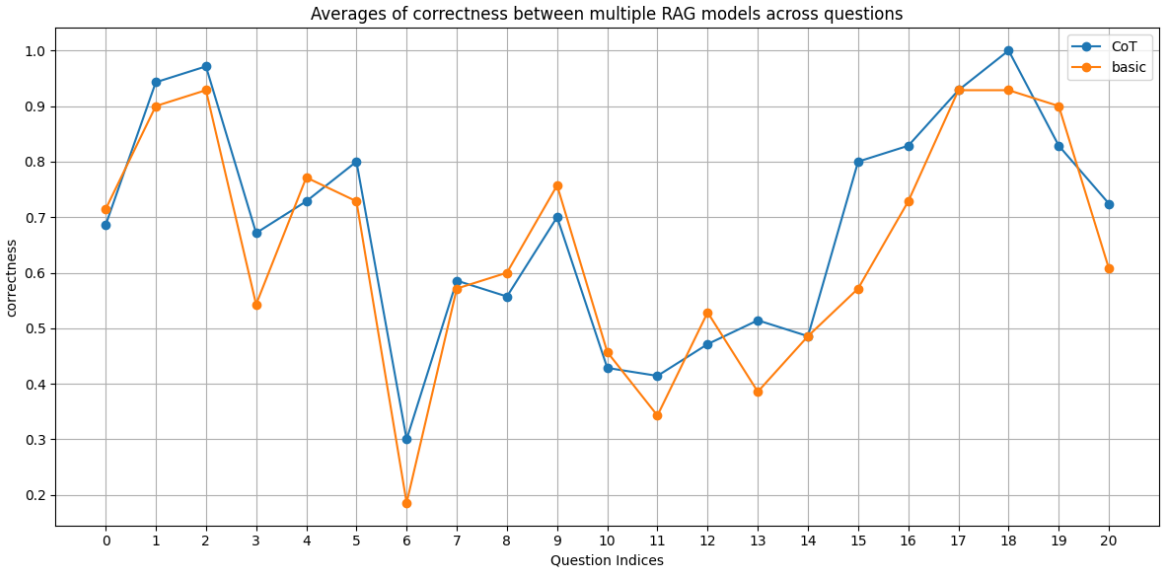

Figure 4.3: Average of Correctness metric across different questions for the CoT model and the basic model.

Figure 4.3 reveals that the basic model outperforms the CoT model in seven questions based on average Correctness scores. However, the differences in these scores are typically less than 0.05, suggesting that the basic model does not necessarily provide correct answers where the CoT model fails. This minor discrepancy may stem from the CoT model's tendency to embed much of its reasoning within the thought process itself. Consequently, while the CoT model's final answer remains accurate, the evaluating LLM may assign a lower Correctness score due to the perceived lack of explicit justification in the answer.

Significant improvements for the CoT model are observed in questions 3, 15, and 20. For question 3, which pertains to Scope 3 emission targets, the CoT model likely benefits from a better differentiation between Scope 1, Scope 2, and Scope 3 targets, leading to more accurate responses. Question 15 addresses whether given company´s targets are accessed by Climate Action 100, sometimes the answer to this question must be deduced by interpreting that no mention of CA100 means it does not access the targets of the company. The CoT model's structured reasoning process, can help the LLM understand this fact contributing to its superior performance. Similarly, question 20, which queries the global organizations tracking emissions with which a company aligns, benefits from the CoT approach. With approximately four such organizations mentioned, the CoT model's step-by-step reasoning aids in identifying relevant mentions in the context and making a more informed decision.

Although the CoT heuristic exhibits a slightly lower average Correctness score for some questions, its substantial improvements in others—particularly questions 3, 11, and 20—demonstrate its ability to deliver higher-quality answers using the same con-

text. This underscores the positive impact of the CoT approach on enhancing answer quality in RAG chatbots.

4.6 Practicality and Token-usage Analysis

Per Section 4.2, we evaluate token usage alongside performance metrics to identify cost-effective models for sustainability reports, as some heuristics' resource demands may outweigh their benefits. This analysis informs decisions on the most efficient model. For this demonstration, we focus on input tokens, which are less costly, without separating input and output tokens. While constructing the KG, we did not precisely track token expenditure, but rough estimates suggest about 5 million tokens for a single KG construction. For a 21-question report, this adds roughly 230,095 tokens per question, included in our estimates. Detailed calculations are in the Excel file `Token_usage.xlsx`, recording token usage per heuristic, question, and report based on model interactions.

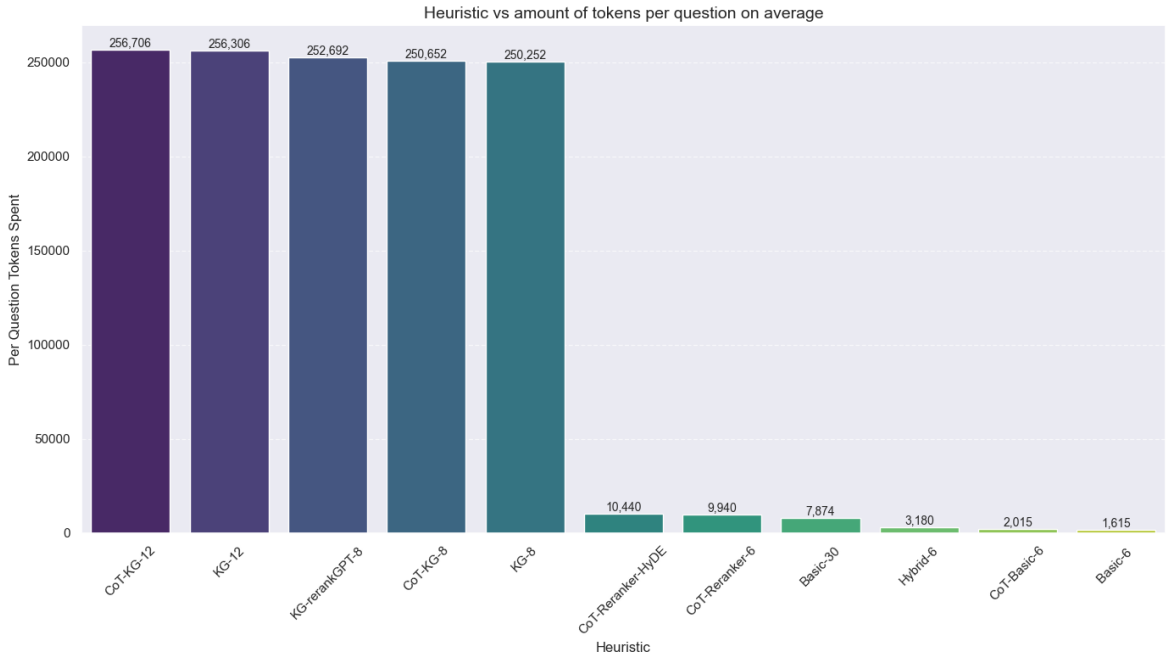

Figure 4.4: Approximation of tokens spent per question on average for various RAG models, highlighting the significant disparity in token usage when employing Graph RAG models.

Figure 4.4 illustrates a stark contrast in token expenditure when utilizing Graph RAG models compared to other heuristics. The majority of tokens consumed by Graph RAG models are attributed to the preprocessing phase, compounded by the larger context windows required for individual calls. In our specific use case—where the model is tasked with completing a sustainability report for a single instance, without

subsequent reuse for the same company—this raises a pertinent question: Are the marginal performance improvements (as measured by the tracked metrics) sufficient to justify the significant time and cost investments associated with Graph RAG models?

4.7 Manual Assessment of Answer Quality

Following the qualitative assessment strategy in Section 4.2, we examine responses from various models to evaluate if quantitative metrics reflect true improvements in answer quality. We compare the Graph RAG model (8 nodes), the CoT model with HyDE search and rerankGPT, and a basic model (6 chunks), documenting selected responses and their ratings.

All three models correctly addressed the first three questions pertaining to company ABF; however, none achieved a perfect score of 1 across all responses. For instance, the Graph RAG model's response to the question `Does the company have net zero targets for scope 1 and scope 2 emissions?` was:

> `Yes, the company has net zero targets for Scope 1 and Scope 2 emissions. Associated British Foods (ABF) aims to achieve a 30% absolute reduction in Scope 1 and Scope 2 emissions by 2030, using a 2018 baseline. Additionally, ABF is committed to reaching net zero greenhouse gas emissions by 2050, which includes cutting both Scope 1 and Scope 2 emissions as part of its broader climate strategy.`

The manual answer provided by the team stated:

> `Net-zero by 2050 (depending on availability of renewable energy and decarbonization of vehicle fleets etc.). Businesses are all committed to cutting their Scope 1 and Scope 2 greenhouse gas (GHG) emissions`

While the response is accurate, it includes an additional reference to a 30% reduction goal by 2030, which, though not incorrect, is extraneous. Ideally, such responses should not be penalized to better reflect their overall accuracy.

A subsequent, more challenging question for company ABF followed up, whether the company has a formal net zero Scope 3 target, distinct from interim goals. Notably, the company lacks a company-wide net zero target for Scope 3; however, certain individual businesses within the company have established such targets.

The Graph RAG model responded as follows:

> `No, the company does not have a formal net zero Scope 3 target distinct from interim goals. While Primark has set an interim`

```
target to halve its absolute carbon footprint across all three
scopes by 2030, there is no specific formal target for net zero
Scope 3 emissions mentioned in the provided context.
```

This response is entirely accurate and appropriately infers the absence of a company-wide target.

The CoT model's response was:

```
 Answer:  Yes, Primark has a formal net zero Scope 3 target as
part of its commitment to reduce emissions by 50% by 2030.  However,
AB Sugar is still measuring its Scope 3 emissions and does not
yet have a formal target.
```

This response is also correct but fails to address the company-wide context, focusing instead on individual business units, which does not fully align with the question's intent.

The basic model's response was:

```
 Yes, the company has a formal net zero scope 3 target, distinct
from interim goals.  They aim for a 50% absolute reduction in Scope
1, 2, and 3 emissions by 2030 from a 2018 baseline, with specific
commitments to reduce emissions across their value chain.
```

This response is inaccurate, as it incorrectly attributes a company-wide target based solely on Primark's goals.

The assigned Correctness scores were 0.8 for the Graph RAG model, 0.5 for the CoT model, and 0.2 for the basic model. All models received a Faithfulness score of 0.5, and none achieved success in IR metrics, all scoring zero. The Correctness metric partially reflects the factual accuracy in this instance; however, an optimal evaluation would assign a score of 0 to the basic model's response and 1 to the Graph RAG model's response to better capture the degree of accuracy.

The next question we will inspect more closely relates to scenarios where no content needs to be retrieved, as the documents lack mention of the given subject, requiring the model to deduce the answer independently. The question was:

```
Are the company's targets assessed by Climate Action 100 (CA100)?
```

Here, we aim to highlight the advantage of CoT prompting over basic models in handling such cases. For the firm RWE, the correct answer is that the company's targets are not assessed by CA100. The response from the combined CoT, HyDE, and rerankGPT model was:

```
1. The question asks whether RWE's targets are assessed by Climate
Action 100 (CA100) and requires a yes/no answer along with any
gaps in criteria mentioned by CA100.
2. The context from the sustainability strategy report shows RWE
has ambitious climate targets, including climate-neutrality by
2040 and interim targets for 2025 and 2030, validated by the Science
Based Targets initiative (SBTi). However, there is no mention
of CA100 or its assessment of these targets.
3. Since no reference to CA100 exists in the provided documents,
I cannot confirm that RWE's targets are assessed by CA100.
No, the company's targets are not assessed by Climate Action 100
(CA100). Gaps in criteria mentioned by CA100 are not specified
in the provided context.
```

This detailed reasoning justifies the negative answer and provides a thorough assessment, unlike the basic model's response:

```
No, the company's targets are not explicitly assessed by Climate
Action 100 (CA100). Gaps in criteria mentioned by CA100 include
the lack of direct reference to CA100's engagement process and
specific alignment with CA100's expectations for corporate climate
accountability.
```

While correct in its conclusion, this response introduces unsupported references to CA100's engagement process and expectations, which could mislead readers.

The Graph RAG model's response was:

```
No, the company's targets are not mentioned as being assessed by
Climate Action 100 (CA100). There are no specific criteria or
assessments related to CA100 provided in the context.
```

The CoT model excels by offering clear, step-by-step reasoning, enhancing transparency in deducing the absence of CA100 assessment. The Graph RAG model delivers a concise, accurate answer without extraneous details. The basic model, though correct, risks confusion by adding unsupported information. This comparison underscores the value of reasoning clarity and relevance alongside factual correctness.

4.8 Selecting the Optimal Model for Implementation

After the entire evaluation process which included comparing performance metrics, the statistical significance of their changes, token usage breakdown, analysis of metrics by

question and lastly manual answer inspection by us and the Sustainability team, we narrowed down the model options for future sustainability report creation to three: the Chain of Thought model with HyDE retrieval and GPT reranking, the pure GPT reranking model, and the pure Chain of Thought model. All of these models showed improvements in metrics they were supposed to enhance. While CoT model gained mainly in Correctness of the answer, GPT reranking model gained in IR metrics as well as the correctness as the quality information helped model answer accordingly to the question. The model which combined all of these approaches with the HyDE retrieval improved in all respects the most.

We excluded the Graph RAG approach as it required time-consuming preprocessing to build the knowledge graph, which added delays without significantly improving correctness compared to other methods. The high token cost and the challenge of fact-checking with extensive context also made it impractical.

The baseline model performed well enough too, showing that our core components—like parsing, prompting, and chunking—were effective. This meant that improvements in more advanced models were small but still meaningful.

We chose the combined approach of Chain of Thought with HyDE retrieval and GPT reranking because it achieved the highest average scores across evaluation metrics (excluding Graph RAG). Statistical tests confirmed significant improvements on level of significance $\alpha = \frac{0.05}{7}$ in Correctness, Precision, Recall, and F_1 Score compared to the baseline model. The Sustainability team can further improve the HyDE retrieval prompt for each question to better target relevant text chunks. The Chain of Thought method excels at deducing answers that require analysis beyond direct text citations. As requested, the model's thought process will be included in a separate column in the report for transparency. Additionally, the reranker consistently selected the most relevant chunks, improving answer accuracy.

While individual heuristics performed well on their own, combining their strengths in this unified approach maximizes their benefits. After reviewing these advantages with the Sustainability team, they expressed satisfaction with the model's capabilities. As of the writing of this thesis, the team is already using the report-making chatbot to assess Zurich Insurance's customers.

Conclusion

In this thesis, we investigated a range of methods to enhance the quality of responses generated for sustainability reports through Retrieval-Augmented Generation models. After a comprehensive evaluation of various heuristic configurations, we arrived at a final model that effectively balances simplicity and performance, adhering to the principle of Occam's razor. Although the Graph RAG approach exhibited potential in newly found relationships among entities within the documents—offering valuable insights for answer generation, however the marginal performance gains did not justify the significant increase in computational resources and time required for report generation. Consequently, we adopted a less resource-intensive model that delivers comparable outcomes, aligning with practical constraints and efficiency considerations.

The workflow of the final model is both systematic and efficient. Upon receiving a query, the HyDE retriever generates a hypothetical text chunk encapsulating a potential answer. This synthetic chunk is embedded into a semantic vector space using the *text-embedding-large-3* model, enabling the retrieval of the 30 most similar text chunks from the VecDB. Unlike traditional methods that embed the query directly, this approach enhances retrieval precision by aligning the embedding with the target answer context. Subsequently, the GPT reranker, powered by the *gpt-4o-mini* LLM, evaluates these chunks and selects the top six most relevant ones. This reduction in chunk volume facilitates efficient verification by the sustainability team, ensuring transparency and reliability in the final output. The selected chunks, alongside the original query, are then processed by the *gpt-4o-mini* model, which employs chain-of-thought prompting to analyze, reason, and formulate a coherent and accurate response.

Our analysis revealed that the performance differences between the heuristic approaches and the baseline RAG model were subtle, with average evaluation metrics exhibiting only modest variations. Initially, these improvements appeared insignificant or small enough to not take into account, however a Wilcoxon signed-rank test confirmed their statistical significance, accounting for the non-normal distribution of the metrics. This finding was particularly pertinent given the presence of outliers—such as instances where no relevant text existed in the documents for certain queries—highlighting the limitations of our evaluation framework. These insights prompted a recalibration of our expectations, encouraging a more nuanced appreciation of the model's achievements

within the context of imperfect metrics.

The consistency of our results may be partly attributed to the robustness of the parsing algorithms and chunking strategies employed. By leveraging state-of-the-art parsing techniques, we ensured the production of high-quality text chunks, which likely played a pivotal role in stabilizing model performance—a factor we did not explicitly quantify against simpler alternatives. Furthermore, advancements in LLMs, exemplified by *gpt-4o-mini*, have diminished the necessity for extensive preprocessing compared to earlier models like *gpt-3.5-turbo*. This evolution underscores a shift in RAG design, where sophisticated LLMs mitigate the need for intricate peripheral optimizations.

In summary, within the allocated time and budgetary constraints, we exhaustively explored optimization strategies for RAG-based sustainability report generation. The resulting solution has been successfully implemented by the sustainability team at Zurich Insurance Company Ltd., streamlining their assessment processes for future clients. This deployment not only validates the efficacy of our approach but also demonstrates its impact on operational efficiency.

Looking ahead, a promising avenue for further research is the development of an agentic RAG framework, in which multiple specialized retriever–generator agents collaborate to answer complex, multi-step queries. Such a multi-agent system could route different aspects of a question to the most appropriate model or data source, improving both accuracy and efficiency. Having first established a simple, transparent RAG setup that meets our practical requirements, we now plan to build on these results by prototyping and evaluating this more sophisticated, agent-based approach in future work.

Bibliography

[1] Step Up AI. Improving the retrieval step of rag systems. https://stepup.ai/rag_improving_retrieval/, 2023. Accessed: 2025-03-02.

[2] Artifex Software, Inc. *PyMuPDF Documentation*. URL https://pymupdf.readthedocs.io/en/latest/. Accessed: 2024-11-29.

[3] Ross Ashman. The art of rag part 3: Reranking with cross encoders, 2023. URL https://medium.com/@rossashman/the-art-of-rag-part-3-reranking-with-cross-encoders-688a16b64669.

[4] AWS. Retrieval-augmented generation, 2023. URL https://docs.aws.amazon.com/sagemaker/latest/dg/jumpstart-foundation-models-customize-rag.html. Accessed: 2025-01-31.

[5] Ashish Bansal. Optimizing rag with hybrid search and contextual chunking. *Journal of Engineering and Applied Sciences Technology*, 2023. doi: 10.47363/jeast/2023(5)e114.

[6] Data Science for Software Development (DS4SD). *Docling: Document Language Models for Industry*. URL https://ds4sd.github.io/docling/. Accessed: 2024-11-29.

[7] Daniel Dugas. Gpt architecture - a visual exploration, 2023. URL https://dugas.ch/artificial_curiosity/GPT_architecture.html. Accessed: 2024-11-10.

[8] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization, 2024. URL https://arxiv.org/abs/2404.16130v1. Version 1, No published version found;.

[9] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. In *Joint Proceedings of the Posters and Demos Track of 12th International Conference on Semantic Systems - SEMANTiCS2016 and 1st International*

Workshop on Semantic Change & Evolving Semantics (SuCCESS16), pages 1–4, 2016. URL https://www.researchgate.net/publication/323316736_Towards_a_Definition_of_Knowledge_Graphs.

[10] Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of EACL 2024: System Demonstrations*, pages 150–158, 2024. URL https://aclanthology.org/2024.eacl-demo.16/.

[11] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of ACL 2023 (Volume 1: Long Papers)*, pages 1762–1777, 2023. doi: 10.18653/v1/2023.acl-long.99. URL https://aclanthology.org/2023.acl-long.99/.

[12] GeeksforGeeks. Kolmogorov-smirnov test (ks test), 2024. URL https://www.geeksforgeeks.org/kolmogorov-smirnov-test-ks-test/. Accessed: 2025-05-13.

[13] GeeksforGeeks. How to calculate jaccard similarity in python, 2025. URL https://www.geeksforgeeks.org/how-to-calculate-jaccard-similarity-in-python/. Accessed: 2025-05-12.

[14] Siladitya Ghosh. A deep dive into openai's text-embedding ada-002: The power of semantic understanding, 2024. URL https://medium.com/@siladityaghosh/a-deep-dive-into-openais-text-embedding-ada-002-the-power-of-semantic-understanding Accessed: 2024-12-02.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[16] Harrison Hoffman. Embeddings and vector databases with chromadb. URL https://realpython.com/chromadb-vector-database/#text-embeddings. Accessed: 2024-12-14.

[17] Antonio Jimeno Yepes, Yao You, Jan Milczek, Sebastian Laverde, and Renyu Li. Financial report chunking for effective retrieval augmented generation, 2024. No published version found;.

[18] Zhi Jing, Yongye Su, and Yikun Han. When large language models meet vector databases: A survey, 2024. No published version found;.

[19] Kush Juvekar and A. Purwar. Introducing a new hyper-parameter for rag: Context window utilization. *ArXiv*, abs/2407.19794, 2024. doi: 10.48550/arXiv.2407.19794. No published version found.

[20] Aadit Kshirsagar. Enhancing rag performance through chunking and text splitting techniques. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2024. doi: 10.32628/cseit2410593.

[21] LangChain. Chains - langchain documentation, 2025. URL `https://python.langchain.com/v0.1/docs/modules/chains/`. Accessed: 2025-01-02.

[22] 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. *Transactions of the Association for Computational Linguistics*, 12: 157–173, 2024. doi: 10.1162/tacl_a_00638. URL `https://aclanthology.org/2024.tacl-1.9/`.

[23] Akash Mathur. Advanced rag: Optimizing retrieval with additional context metadata using llamaindex. *Medium*, 2023. URL `https://akash-mathur.medium.com/advanced-rag-optimizing-retrieval-with-additional-context-metadata-using-lla` Accessed: 2025-03-02.

[24] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.173. URL `https://aclanthology.org/2020.acl-main.173`.

[25] Ontotext. The power of knowledge graphs within the financial industry, 2024. URL `https://www.ontotext.com/blog/the-power-of-knowledge-graphs-within-the-financial-industry/`.

[26] Ioannis Papadimitriou, Ilias Gialampoukidis, Stefanos Vrochidis, and Ioannis Kompatsiaris. Rag playground: A framework for systematic evaluation of retrieval strategies and prompt engineering in rag systems, 2024. No published version found;.

[27] Prompting Guide Contributors. Prompting guide, 2024. URL `https://www.promptingguide.ai/`. Accessed: 2024-12-31.

[28] Prompting Guide Contributors. Prompting guide, 2024. URL `https://www.promptingguide.ai/introduction/tips`. Accessed: 2024-12-31.

[29] Nils Reimers and Iryna Gurevych. Cross-encoders, 2020. URL https://www.sbert.net/examples/applications/cross-encoder/README.html.

[30] Samia Sahin. Evaluating rag systems: Metrics and best practices, 2025. URL https://medium.com/@sahin.samia/evaluating-rag-systems-metrics-and-best-practices-906a2c209bb5. Accessed: 2025-03-06.

[31] Statistics Solutions. How to conduct the wilcox sign test, 2010. URL https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/how-to-conduct-the-wilcox-sign-test/. Accessed: 2025-05-13.

[32] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is chatgpt good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14918–14937. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.923/.

[33] MyScale Team. Components of okapi bm25: Unraveling term frequency and document length, 2024. URL https://myscale.com/blog/beginners-guide-okapi-bm25-unraveling/#components-of-okapi-bm25. Accessed: 2024-12-02.

[34] Vinu Sebastian Thomas. Document chains in langchain, December 2023. URL https://medium.com/@vinusebastianthomas/document-chains-in-langchain-d33c4bdbabd8. Accessed: 2025-01-02.

[35] Unstructured-IO. *Unstructured: Preprocessing Text Data for NLP*. URL https://github.com/Unstructured-IO/unstructured. Accessed: 2024-11-29.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[37] Wikipedia contributors. Hierarchical navigable small world, 2024. URL https://en.wikipedia.org/wiki/Hierarchical_navigable_small_world. Accessed: 2024-12-15.

[38] Crystina Zhang, Sebastian Hofstätter, Patrick Lewis, Raphael Tang, and Jimmy Lin. Rank-without-gpt: Building gpt-independent listwise rerankers on open-source large language models. In *Advances in Information Retrieval*, pages 233–

247. Springer, 2025. doi: 10.1007/978-3-031-88711-6_15. URL `https://link.springer.com/chapter/10.1007/978-3-031-88711-6_15`.

[39] Qintong Zhang, Victor Shea-Jay Huang, Bin Wang, et al. Document parsing unveiled: Techniques, challenges, and prospects for structured information extraction, 2024. No published version found;.

Appendix A: Electronic Appendices

The electronically submitted version of this thesis includes all code scripts used in the project, along with a `requirements.txt` file listing the necessary Python libraries. These scripts are compatible with Python 3.10.10; however, I cannot guarantee that the dependencies will function correctly with other Python versions. Additionally, to run the RAG program, you will need your own Azure OpenAI key, as Zurich Insurance Company LTD. does not permit the sharing of internal keys.

If you have access to Azure OpenAI services, please create a `.env` file to input your credentials. Once this is done, the code should run smoothly. For further details, please refer to the submitted README markdown file.

Additionally, due to the data generated from this project exceeding the available space in the AIS submission system and in compliance with the privacy policy of Zurich Insurance Company LTD., the remaining data is submitted on a USB drive. This USB contains all output JSON files from testing each file, along with Excel summary sheets and the original PDFs that were parsed for information. Furthermore, it includes VecDBs and knowledge graphs saved as JSON files, which can be easily converted into NetworkX graphs for use in the graph RAG heuristic approach.