Faculty of Science & Technology

Department of Computer Science

Enhancing Investigative Journalism: Leveraging Large Language Models and Vector Databases

Muhammad Nauman Ali

Master's thesis in Computer Science – INF-3990 – May 2024

Submitted on May 15, 2024

Supervisor – Lar Ailo Bongo Professor, Department of Computer Science,
The Arctic University of Norway

Co-Supervisor – Benjamin Ricaud Associate Professor, Department of Physics and Technology,
The Arctic University of Norway

Co-Supervisor – Nicoali Bakkeli Utviklingssjef, Mediehuset iTromsø

We cannot solve problems with the kind of thinking we employed when we came up with them. – Albert Einstein

Declaration of the Usage of AI tools

For my Master Thesis work, I acknowledge the use of Generative Artificial Intelligence ChatGPT for the following purposes:

- Improving text and removing grammatical mistakes.
- To understand the different concepts, doing the summarization.

Acknowledgements

I would like to express my deepest gratitude to my supervisors Lars Ailo Bongo, Benjamin Ricaud and Nicolai Bakkeli for their invaluable guidance and support throughout the project. Their suggestions, insights and expertise have been instrumental in shaping this thesis. I extend my appreciation to Health Data Lab, IFI, UiT and iTromsø for providing the resources and facilities necessary for conducting this project.

This thesis would not have been possible without the contributions and support of all these individuals, and for that, I am truly grateful.

Tromsø, May 2024
Muhammad Nauman Ali

Abstract

The advancement in the field of Artificial Intelligence (AI) has brought revolution in almost every field of life, and Journalism is also one of them. Which includes prospective use in Investigating reports and uncovering information. This project explores the avenue of integrating technologies such as Large Language Model (LLM) with the Vector Databases. At the same time, the motive is to address two avenues: Information Retrieval and LLM for summarization and finding information of interest to the journalists.

We begin the study with an overview of related concepts/literature. Then, we proposed a system based on the literature in the methodology. The proposed system is based on Retrieval Augmented Generation (RAG) architecture employing Vector Database and the integration of LLM. The vector database was employed to efficiently retrieve relevant documents, and LLM for putting the information in concise form and also identifying any irregularities in the cases. A series of queries and prompts were presented by iTromsø, and the system was tested. The results, both documents retrieved and the prompt answers were evaluated by iTromsø.

The results for documents retrieval, had varied varied degree of accuracy, with some queries giving the most relevant and some completely fail to retrieve the document intended. The quality of answers from also showed variance as expected and ChatGPT4 outperforming ChatGPT 3.5 turbo and GPT4All in answering the prompt with high accuracy.

The duplication of documents and also the presence of special characters and void spaces in the text effected the results for documents retrieval by not able to retrieve most desired document in most cases. Except ChatGPT 4, ChatGPT 3.5 turbo and GPT4All response was also effected due to special characters and white spaces.

While the proposed system showing advantage in assisting journalists with investigative process both in term of scalability and efficiency when compared to traditional approaches. But the limitations in accurate document retrieval must be addressed by cleaning the text data.

Contents

CHAPTER 1

Introduction

1.1 Background

Data is generated at an exponential speed every day, whether it is health, financial, or any other sector of life. Since the age of digitization, the use of data has increased a lot. This digital transformation has opened various avenues for searching for solutions to the problems never thought of before. Now a single click on a link presents data which can be further analyzed to identify the behavior of the user, his interests, inshort a complete user profile can be computed based on his clicks on the different links during web surfing.

In the past decade data was called the next black gold after petroleum. We are seeing the use of data in almost every sector of life and has changed the way we used to deal with problems. Advancements in Artificial Intelligence and Machine Learning have made it possible to process and analyze this enormous amount of data in a very short period of time with higher efficiency.

There has been a lot of research and development in the area of LLM in recent years. LLM which has opened various avenues for integrating the capabilities of LLM to different time consuming tasks. For instance, Transformers, a deep learning architecture proposed in 2017 has significantly changed the whole scenario of natural language processing. Since then many large language models like GPT-2, GPT-3, GPT-4 and BERT has demonstrated the ability of transformers in tasks such as:

- Question Answering
- Paraphrasing
- Sentiment Analysis
- Language modeling

A fundamental element of Transformers is self-attention, where each token's 'ego-centric' focus within a corpus is evaluated. This self-attention mechanism gauges the relevance of neighboring words to the token at hand. LLM's excel in their intended purpose: generating coherent and contextually plausible text in response to input. Furthermore, they are demonstrating proficiency in additional tasks such as summarization, question answering, and text classification, often referred to as emergent capabilities.

Although LLM are mostly used for text generation. But they have the capabilities beyond just text generation. For instance, LLM have recently found application in creating sentiment detectors, toxicity classifiers, image descriptions,and recommender systems etc.

1.2 Motivation

To get the information about what is happening around these days we use some form of media either it is newspaper, radio, television or even social media. News became very important in our daily life from the past century because of different events political, business. And more specifically the events happening locally are also very much important in the sense of new development projects, investigating local living conditions. To cover all those developments and present it as a news story, this is the job of a Journalist. A journalist's job is to collect information, which involves reading press releases, reading documents, verifying statements and facts.

The Journalist goes through different documents, stats and facts, to investigate the local living conditions and disparities in social and economic aspects. This process is a time consuming process because of going through a variable number of documents individually almost every day. Then looking into each document to get an overview of it and find the disparities and consider it as a news story. But sometimes this could get a hectic process, which might lead to missing some important data in those different documents which might be of interest to the journalists. For a small number of documents this wont make any difference but what if there is a high number of documents being generated every day from multiple authorities.

Now with recent development in Deep Learning. LLM has shown some promising results in tasks related to question answering, summarization. So they are capable and there is a potential to leverage vector stores and integrate with LLM to solve the problem faced by journalists.

1.3 Objectives

The main objective we want to achieve is to make the job of a journalist easier in a way that involves designing a system for efficient and fast retrieval of documents based on the keywords search from vector stores.

In this project, our focus is on building an application that aids journalists in investigating local living conditions and disparities in social and economic aspects. We leverage a large dataset comprising various types of documents, including text. The additional objective of our application is to classify documents as newsworthy for journalists. Additionally, the system can recommend related documents from different sources to address user queries.

1.4 Thesis Structure and Organization

This thesis provides a comprehensive overview of the LLM based application for the Journalists. It is structured in a way that following chapters provides brief overview of concepts, methodology, implementations, experimental results, discussions and conclusion and future recommendations.

CHAPTER 2
Background and Terminology

2.1 Internet Searching

Since the introduction of the internet and search engines, it has completely changed the way we look for information in today's modern age. It has become an integral part of our daily lives. We now have a term called Google for searching on the internet because of the very high usage of the Google search engine in the whole world for searching anything on the internet. In one way, It has brought ease in having access to open information and data. But if we look at it the other way around, at the same time, there is a very high volume of information that makes it sometimes quite difficult to get the most desired information.

While there are quite sophisticated search engines that retrieve the highly desired information, there is a chance that one might miss the content of the retrieved data because of the few lines displayed on the search engine page. In some scenarios, let's say there is a relevant result displayed, and when you open the link, the information is not in the interests of the user because of the very inadequate amount of info displayed from a search result.

An estimate from 2022 suggests that data on the internet has reached 175 trillion gigabytes of information or data, which can and in some way might hide the information that might be of interest to 1 percent or less than that of the user's queries. In a way, it can also contribute to hiding useful results from users because of the search engine mechanism.

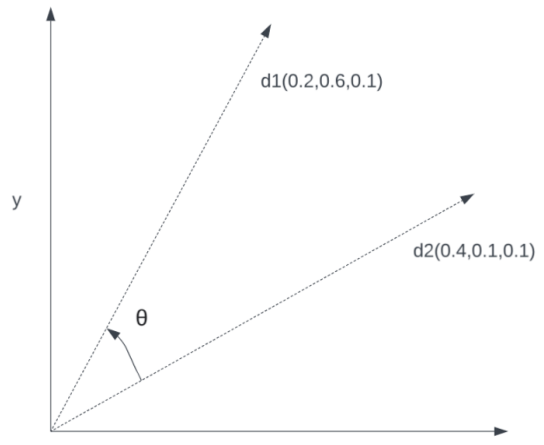


Figure 2.1: A figure representing two document vectors.

2.2 Vector Space Model

Representing documents in the high dimensional vector space concept was first introduced in 1975 by Salton, G., Wong, A., & Yang, C. S. in the paper "A Vector Space Model for Automatic Indexing" Communications of the ACM[1]. A method for representing documents and queries in a high-dimensional vector space was proposed. It describes a term or a definition in vocabulary corresponding to unique dimensions. A term presence or absence in a document is represented as binary values. The similarity between documents and queries has to be computed using vector-based techniques. Those vector-based techniques involve cosine similarity. Vector Space Model involves components such as:

- Term Frequency: Term Frequency can be defined as the frequency of each term or query in a document, representing its importance.
- Inverse Document Frequency: Inverse document frequency or IDF is calculated by the log of an inverse fraction of documents containing the term. It is the measure of the document and how much information it provides.
- Weighting Scheme: The weighting scheme is by combining term frequency TF and inverse document frequency IDF in order to assign weights to a term. It basically captures the relevance.

2.3 Word Embeddings

The traditional method of word representation lacks the capability of capturing semantics, i,e. The meaning of the word and their similarity with others words. Although the traditional methods were simple and were easy to implement but lacked one of the very powerful features.

Word embeddings can be defined as a representation of words for text analysis[2], typically in the form of a real-valued vector, encoding the meaning of the word as a number. In simple words, we can define word embeddings as numerical representations of words, capturing semantic syntactic relationships between words based on their context in a text corpus, which is given as an input.

Because of word embeddings algorithms can understand the text better by inferring the semantics together with syntactics. Which in turns enables algorithms to better process natural language (NLP) tasks. The words are encoded in continuous vector space, in which similar words are located closely and show relativity, in terms of semantics. By capturing the word semantics it has several advantages and applications.

- **Semantic Similarity:** In vector space words and concepts that share similarities in terms of their meaning are located close, which allows algorithms to calculate the similarity by their differences in vectors.
- **Contextual Information:** In word embeddings, not only the meaning of the word is captured, but also the context in which it is used is also captured.
- **Dimensionality Reduction:** Embeddings have much lower dimensions as compared to the original document.It involves the capturing or encoding in high dimensional space.
- **Feature Representation:** The vector space representation of words or documents can be used as an input features for machine learning models for various tasks such as text classification, information retrieval, sentiment analysis etc.

2.3.1 Neural Network Based Embedding models

Various embedding models based on neural networks have been developed over the past years e,g ada-002, Bidirectional Encoder Representations from Transformers (BERT). However, the concept gained attention after a Google researcher presented the paper Efficient Estimation of Word Representations in Vector Space in 2013. In that paper, they present two efficient architectures, both based on Neural Networks, for learning high-quality word-embeddings[3]. The following are the two architectures:

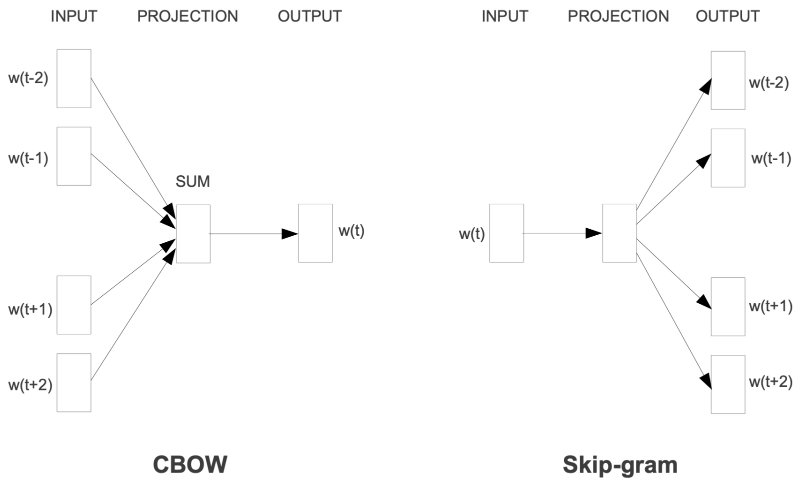


Figure 2.2: CBOW and Skip Gram models architecture[3].

- Continuous Bag of Words Model
- Continuous Skip Gram Model

Continuous Bag of Words Model

In the Continous Bag of Words (CBOW) Model architecture, all the words are projected to the same position, and their vectors are averaged. And the output is to correctly classify or predict the middle word. This architecture uses the continuous distribution of the context[3].

Continuous Skip Gram Model

The Continuous Skip Gram Model architecture is quite similar to CBOW. A log-linear classifier with a projection layer takes a word as an input, and the output is the before and after words in a given range[3].

It was reported that when trained on a large amount of data, the generated or resulting vectors can identify semantic relationships very accurately. This could make a breakthrough in various NLP applications[3].

2.4 Hierarchical Navigable Small World (HNSW) Graph

K-nearest neighbor has been widely used for information searches. However, advancements in computing have led to new methods of storing data and information. The volume of data is continuously increasing at an exponential speed. This constantly growing amount of data has caused us to have scalable and efficient similarity search data structures. So In 2018, Yu. A. Malkov, D. A. Yashunin presented a new approach using HNSW graphs in the paper "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs" [4]. In which the objective was to address the challenges associated with the nearest neighbor searches in high-dimensional space. The main issues were scalability and retrieval quality.

The HNSW graph was introduced as a novel approach for approximate neighbor search, which resulted in improved retrieval quality and efficient retrieval. The paper presents the HNSW graph construction algorithm. The algorithm partitions the data space recursively into smaller regions and, based on their similarity, connects the data points from each region. The paper also describes a search algorithm that finds the approximate nearest neighbor efficiently within the graph structure.

The results from various tests on high dimensional data sets for experimental evaluation of Hierarchical Neighbor Small World had shown promising results with significant improvements in retrieval quality and scalability[4].

2.5 Large Language Model (LLM)

LLM is an artificial neural network/deep learning algorithm capable of performing a variety of natural language processing tasks. The main purpose of LLM is to understand and generate human-like text based on its knowledge of being trained on vast amounts of data. The most interesting feature of LLM is the processing and comprehension of natural languages with very remarkable accuracy and efficiency.

2.5.1 Architecture

The base for LLM is the groundbreaking transformer[5] architecture, which has brought the revolution in the field of NLP. Transformers work with multiple layers of self-

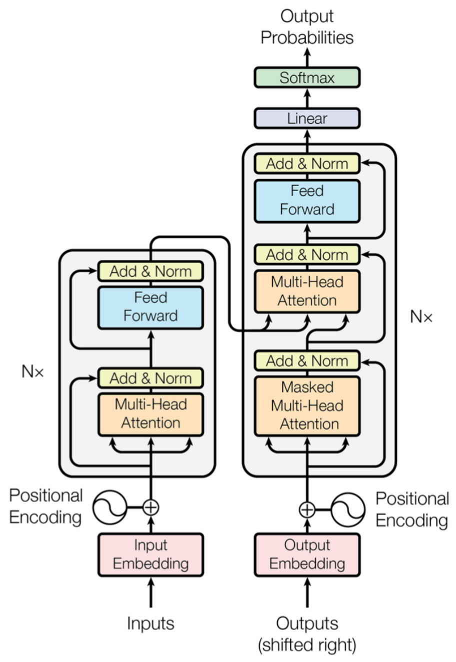


Figure 2.3: A figure showing transformer architecture[5].

attention mechanisms combined with feed-forward neural networks, which makes the model learn things more quickly than other neural network-based models.

Transformer follows an encoder-decoder structure[5]. Where the encoder maps an input sequence to a sequence of continuous representation z. The decoder then generates an output sequence. At each step, the model consumes the previously generated symbol as an additional input[5].

2.5.2 Encoder and Decoder Stack

The encoder consists of multiple identical N layers stack, where N=6[2]. Every layer is further divided into two sub-layers. The first one is a multi-head self-attention mechanism. And the second one is position-wise fully connected feed-forward network[5].

The decoder also consists of the same N=6 identical layers, with the addition of two sub-layers in each layer having another third sub-layer for performing multi-head attention from the output of the encoder[5].

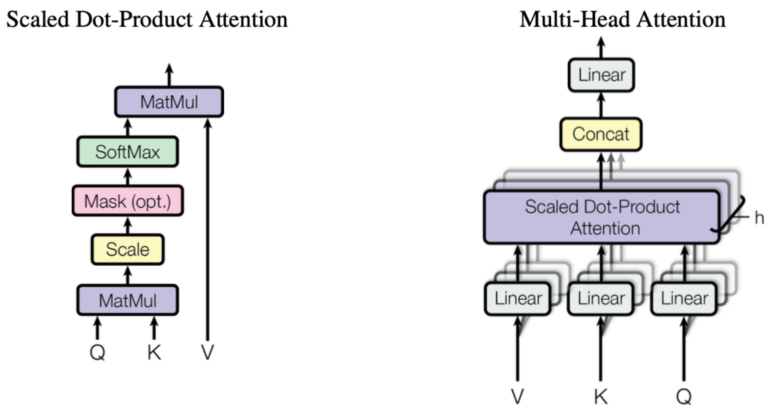


Figure 2.4: Scaled Dot-Product Attention and Multi-Head Attention[5].

2.5.3 Attention

In the attention function, a query and set of key-value pairs are mapped, where all of them, including query, key-value pairs, and output, are all vectors. A compatibility function of the query computes the weight of each value. The output is computed as a weighted sum of the values[5]. The importance of different words in a sequence is weighted while generating the text while also allowing the model to understand the text.

Scaled Dot Product Attention

The input for the particular scaled product attention is queries of dimension d and keys with dimension d. The dot product of the query with all keys is divided by d_k, and then the softmax function is applied to get the weights[5].

Multi-Head Attention

In multi-head attention query, keys and value are linearly projected h times to learned linear projections to d_k, d_k, and d_v dimensions. The attention function is performed in parallel on the projected version of queries, keys, and values, which results in output values of d_v dimensional. These output values are concatenated and projected again, resulting in the final values[5].

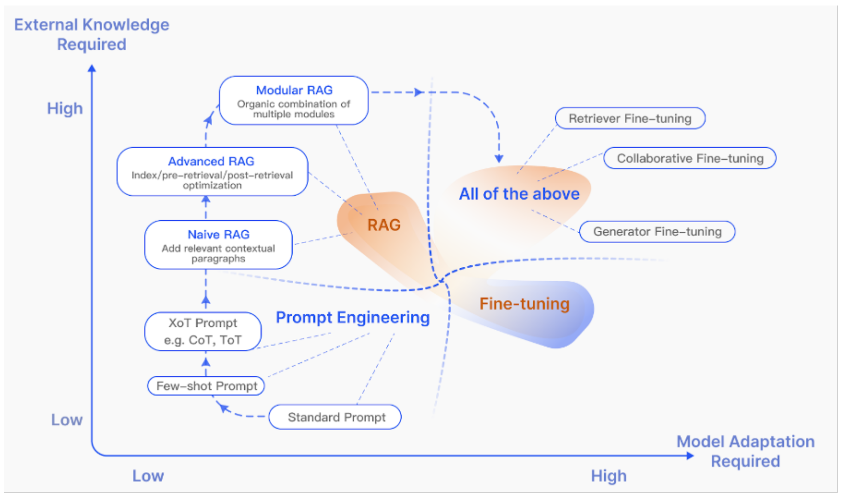

External Knowledge Required
High
Low
Modular RAG
Organic combination of multiple modules
Advanced RAG
Index/pre-retrieval/post-retrieval optimization
Naive RAG
Add relevant contextual paragraphs
RAG
XoT Prompt
e.g. CoT, ToT
Prompt Engineering
Few-shot Prompt
Standard Prompt
Retriever Fine-tuning
Collaborative Fine-tuning
All of the above
Generator Fine-tuning
Fine-tuning
Model Adaptation Required
Low
High

Figure 2.5: Optimization methods for enhancing LLM capabilities[6].

2.5.4 Position-wise Feed-forward Networks

Each of the layers in the encoder and decoder also contains one fully connected feed-forward network. It consists of two linear transformations with ReLU activation in between[5].

Embeddings and softmax

Learned embeddings is used to convert the input tokens and output tokens to vectors of dmodeldimension. The decoder output is converted to predicted next-token using the learned linear transformations and softmax function[5].

Positional Encoding

Positional encoding of the dimension, same as d_{model} of embeddings, is added to the input embeddings at the end of the encoder and decoder stack. The positional encoding is actually information about the absolute or relative position in the sequence. The positional information of the word is incorporated into the embeddings. This way, the model understands the order of words in a sentence[5].

2.6 Retrieval Augmented Generation (RAG)

Large language models have shown some promising results in NLP tasks, but they also come with some problems related to text generation, question answering, etc. Some of the problems so far encountered are Hallucination, Outdated Knowledge, and an untraceable reasoning process[6]. Retrieval Augmented Generation, or RAG, an advanced natural language processing technique, has come into play to solve this issue and has presented a promising solution by providing the capability of incorporating knowledge from external sources.

In this way, the generated text content has greater credibility in the context of knowledge-intensive tasks and domain-specific knowledge. The generated text can also be cross-checked because RAG also provides the source of information, which comes in handy when verifying the information. The positive aspect of RAG is to keep the LLM up-to-date with domain-specific knowledge[6].

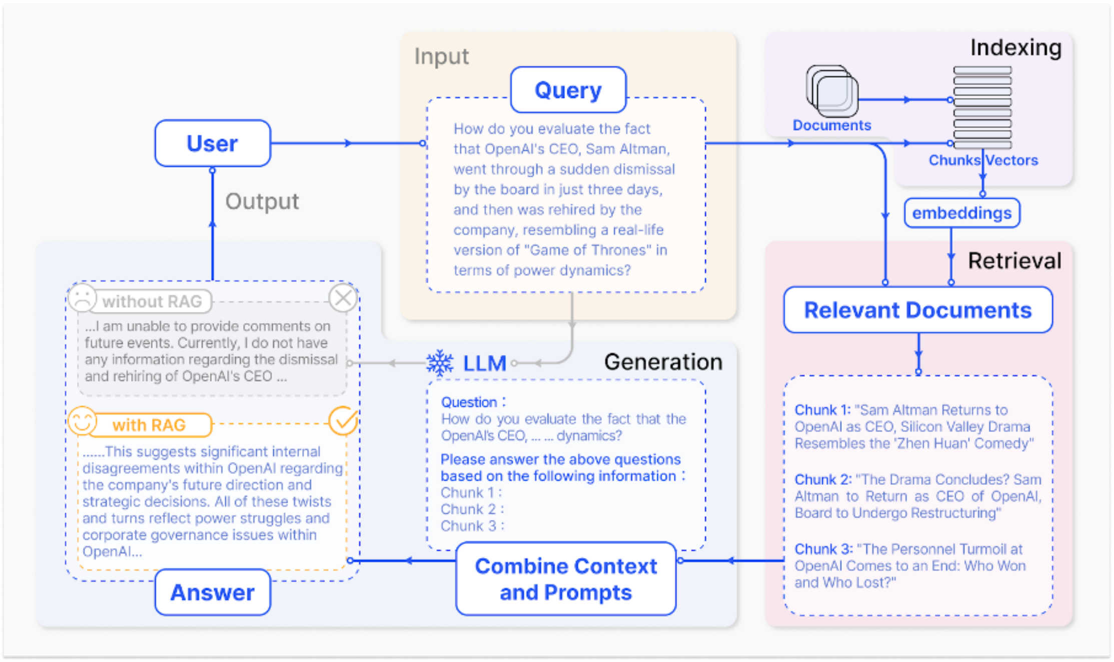
In terms of optimization techniques for enhancing the response from LLM, there are various techniques, such as:

- Prompt Engineering
- Fine Tuning
- RAG
- Combination of all of above

As shown in Fig. 2.5, the RAG has an advantage over other optimization techniques in that it gives real-time knowledge and information. The choice between fine-tuning, RAG, and prompt engineering depends on the problem to be addressed. But the combination of all these approaches can lead to the most optimal performance[6].

The RAG concept was presented by researchers from Facebook (now Meta) back in 2020 in the paper "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." The approach is simple yet powerful in generating knowledge. A pre-trained, parametric-memory generation model is concatenated with non-parametric memory through general-purpose fine-tuning and is referred to as Retrieval Augmented Generation. Parametric memory is a pre-trained transformer, and non-parametric memory is sparse vector index[7].

The RAG model works on a simple approach of combining elements of retrieval-based and generation-based approaches together for a more detailed and verifiable response for knowledge-intensive tasks[6]. The following below is the breakdown:



How do you evaluate the fact that OpenAI's CEO, Sam Altman, went through a sudden dismissal by the board in just three days, and then was rehired by the company, resembling a real-life version of "Game of Thrones" in terms of power dynamics?

...I am unable to provide comments on future events. Currently, I do not have any information regarding the dismissal and rehiring of OpenAI's CEO ...

......This suggests significant internal disagreements within OpenAI regarding the company's future direction and strategic decisions. All of these twists and turns reflect power struggles and corporate governance issues within OpenAI...

Question :
How do you evaluate the fact that the OpenAI's CEO, ... dynamics?
Please answer the above questions based on the following information :
Chunk 1 :
Chunk 2 :
Chunk 3 :

Chunk 1: "Sam Altman Returns to OpenAI as CEO, Silicon Valley Drama Resembles the 'Zhen Huan' Comedy"

Chunk 2: "The Drama Concludes? Sam Altman to Return as CEO of OpenAI, Board to Undergo Restructuring"

Chunk 3: "The Personnel Turmoil at OpenAI Comes to an End: Who Won and Who Lost?"

Figure 2.6: An example workflow in RAG[6].

Retrieval Module

This is the first step in the RAG sequence, where knowledge is gathered to form a large corpus of text or knowledge base[6].

Generation Module

In the RAG sequence, the second or next step is the Generation-based model, which is an LLM, general purpose pre-trained model capable of generating text from scratch[6].

So, what basically happens is the data is retrieved from the knowledge base and is given to a generation-based model to produce the text as a result. That knowledge is the serve as the context for the generation of response form the LLM.

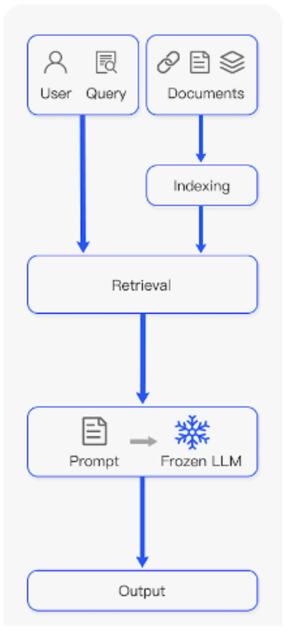


Figure 2.7: RAG dataflow diagram[6]

2.7 Big Data

The term Big Data is associated with huge and massive amounts of data. Dealing with such a large amount of data has its own variable challenges. However, with continuous advancement and development, new solutions are being developed to address those different challenges. With such advancements in the solutions, the availability of massive amounts of data on the internet has drawn huge attention from researchers in the area of decision-making, governance, and business[8].

Big data can be defined by 5V's, i.e., volume, velocity, variety, veracity, and value.

- Volume: Volume of the data is the size of the data or information.
- Velocity: Velocity is the speed, or intervals at which the data is generated.
- Variety: The different types of data being generated, e,g. structured, unstructured etc.
- Veracity: Veracity corresponds to the accuracy of the data.
- Value: It is the most V's of all, which is the worth of the data, the potential of the data in any process, decision.

Dealing with such kinds of data has profound implications and can significantly influence the process of decision-making, governance, etc. The vast amount of data can give actionable insights, which can lead to data-driven decisions. In business, it can identify market trends, preferences of any group, and future opportunities.

In governance, this vast amount of data can be used to efficiently use the resources, improve the processes, and allocate resources to some fields. These, in turn, contribute to the enhancement of public services. For example, Government agencies can use data to address problems in society, such as health, education, and societal challenges. These things in themselves are huge, diverse areas to discover, to learn how to take effect of the data that is available and address those problems that need to be solved.

CHAPTER 3

Methodology

This chapter provides a comprehensive overview of our data, approach, design, and methodologies for the problem we aim to solve, serving as a blueprint for the proposed system, serving as a blueprint for the proposed system.

3.1 Data

The data available in this project is:

- Unstructured text data from housing permits cases.

This data was made available via file transfer. Around 200,000 files in Text File Document (TXT) format were obtained from different cases, with each case containing at least two text files. The original format of those was Portable Document Format (PDF) and were converted to TXT format after scraping.

The data from Byggesak has been made available in this project by iTromsø Mediehuset. Byggesak Tromsø Kommune is the authority responsible for building and construction-related cases. The data is scraped from *"https://innsyn.tromso.kommune.no /byggsak"* for the duration of the last ten years till June 2022. The

The metadata contains some essential information about the document, such as document ID, case serial number, URL, etc. Where each document is related to a property or building case. The cases are applications, complaints from the public, or from the Byggesak to the landlord about any issue or complaints. Then, there is a reply document from Byggesak or the landlord with the same case serial number as the applicant or complainee. Examples of metadata attributes are:

"document id", "case serial number", "document recorded timestamp", "document recorded date", "property address", "title", "to form", " case responsible", "case the legal basis for restriction", "document unit", "document type", "document responsible", "source", "document type code", "document category".

The data available is open information for the public and can be accessed by anyone from anywhere. So there is no big implication of any potential GDPR rule violations or breach of data while processing it on Azure OpenAI and storing it on Azure cloud.

3.2 Proposed Approach

The LLM based application for journalists we design and develop is a kind of system, where we leverage the capabilities of an LLM with our own data. The data which includes unstructured text documents.

To get to our goal, the steps would be:

- Maintaining a vector database
- Interface for querying
- Retrieval of the nearest document in vector store based on query
- Processing the retrieved documents by an LLM based on a prompt

The desired goal for this thesis is to explore the possibilities of LLMs in utilizing tasks related to journalism, which include scraping information from a corpus of documents efficiently and correctly. So far, the problem that journalists face every day is that they go through a large number of documents to scan for potential news stories related to their interests. This time consuming process can lead to missing some information in documents. Since there are multiple documents.

Here, we want to leverage the capabilities of LLM and vector stores. The vector store index documents in vector stores, which store data as high dimensional vectors. Vectors are mathematical representations of features created by applying an embedding function on text, image, audio, and video data. Then, indexed data in the vector store is retrieved and fed to an LLM for summarization and presentation in a concise form for the journalist. Compared to traditional databases, which have predefined criteria for exact matches, vector stores allow for searching the most relevant documents in terms of semantics and context, which makes it ideal for a wide range of applications.

The main component for our system would be User Interface (UI) for the user which is the journalists at iTromsø, data storage component, data retrieval component, and LLM. These are the main components of the system we have aimed for. By this approach we want to solve both problems for precise and efficient retrieval and also with analyzing documents through an LLM for summarization, checking for violations of rules and regulations.

Given the proposed approach to solving our problem, we would need to gather all those open, unstructured text documents. The data should be stored in a data storage solution. However, the challenge here lies in the selection of storage solutions for this enormous amount of data. We have to keep in mind the functionality of the system and the issue that needs to be solved. So, In this case, we have to consider all available options in view of both system functionality and the problems we want to address.

We chose to employ a vector database or vector store as a backbone of the system because it handles high-dimensional data efficiently. Azure Vector Store is highly scalable and optimized for handling vector data. The reason for moving forward with vector databases is they provide a robust framework for querying and managing unstructured data based on similarity metrics. By structuring our data in vector format, we not only streamline the storage but also do the groundwork for more nuanced analysis and insights by an LLM.

Given that we have problems involving the retrieval of documents, So it entails tasks such as similarity tasks, and having this functionality is critical for the efficient and precise retrieval of documents. By employing vector-based solutions, similarity can be efficiently and effortlessly calculated.

After deciding with the storage solution for our approach, the next step involved will be the choice of LLM for the solution of the problems related to hallucination, summarization, and question answering over a document. Here we have multiple possibilities of having an LLM based locally or cloud, and the selection is strictly based on the type of data we want to use, performance and results.

In order to be able to upload the document data to the vector store, we must convert it into its vector embeddings. For that purpose, the whole data should be converted, and the new incoming documents, generated day by day, should also be treated as such. The embeddings for text data can be generated by the text embeddings model. The choice of selecting such a model for generating embedding depends on the type of LLM we will use for the system.

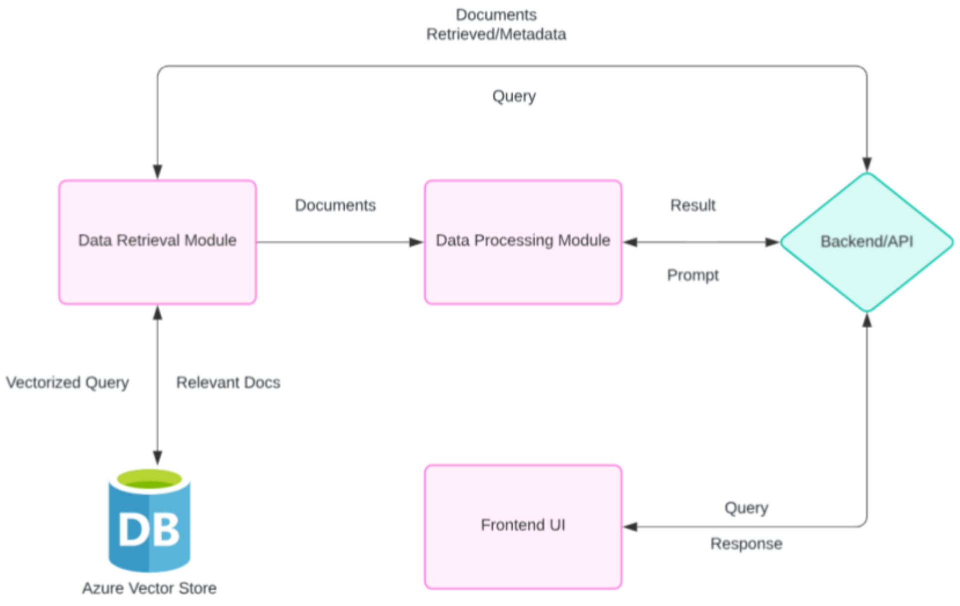


Figure 3.1: The figure represents the architecture of our proposed system.

3.3 Architecture

This section describes the overall architecture of the system, with different services on which it is based and uses for its functionality.

The proposed system is based upon RAG(Retrieval Augmented Generation) architecture. The retrieval of documents is based on RAG. RAG incorporates retrieved documents into a generative model. In which the retrieval component retrieves the document or information and then passes it on to the generative model for response generation. In 2020, researchers from Facebook(now Meta) presented a framework called Retrieval Augmented Generation in order to give the LLM access to data and information on which it hasn't been trained and make it able to answer questions in a more accurate and precise way.

The following are the components of the systems we intend to implement:

- Frontend UI
- Backend/API

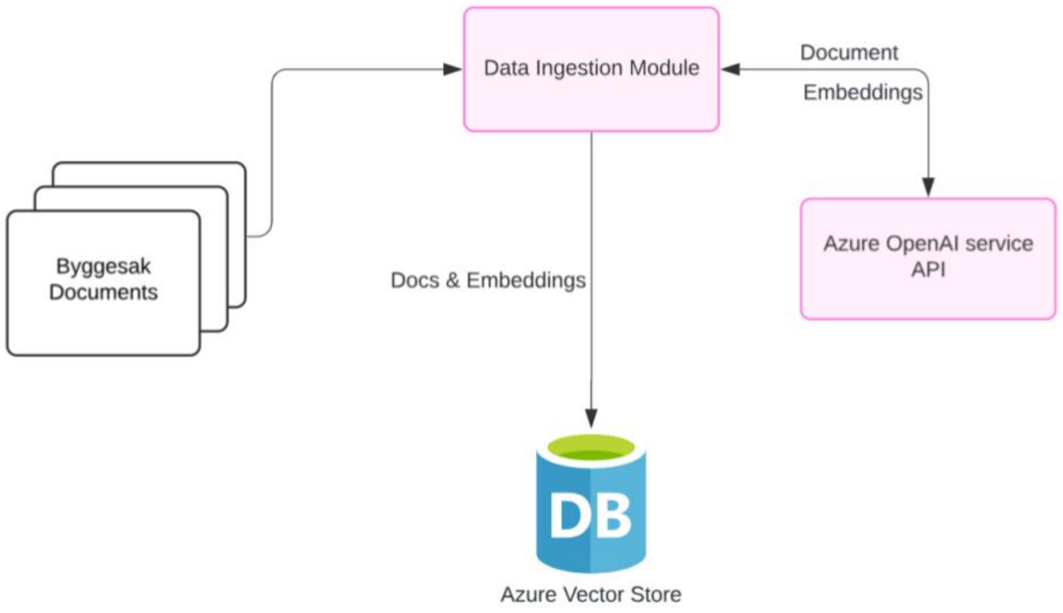


Figure 3.2: Architecture diagram showing the working of Data Ingestion Module

- Data Retrieval Module
- Data Processing Module
- Data Ingestion Module

3.3.1 Frontend UI

The client-side component represents the main UI or the frontend part of the application. This is a simple interface with different text boxes and different filter dropdowns for search and chat functionality for a user to interact with documents like question and answering and summarization tasks. The client-side interacts with the backend through Application Programming Interface (API). Basically, the client side mainly takes the different arguments from the user and sends them to the backend for processing based on the data.

3.3.2 Backend/API

The backend/API layer is responsible for receiving queries from the front end, which is the data entered by the user via an API request. It is responsible for routing the requests to the responsible modules and serves as the intermediary between the frontend UI and the rest of the modules.

3.3.3 Data Retrieval module

The data retrieval module is responsible for retrieving the data for the requests, from the storage, the vector store. The retrieval is based on the requests from the frontend UI. It serves as a bridge for retrieving the relevant document and providing it to the front end UI.

3.3.4 Data Processing Module

So, after the relevant data/documents are retrieved from the vector store, this module processes them according to the requirements of the queries, e.g., prompts from the user. It involves prompting the LLM based on the context provided by the retrieved documents.

3.3.5 Data Ingestion Module

The purpose of the data ingestion module is to collect new documents from the various sources, might be more than one source. It loads the document, creates the embeddings for the documents, and finally uploads it to the vector store, together with the original text document.

CHAPTER 4

Implementation

This chapter describes the overall implementation process and the tools and technologies used for the development of the system. The implementation gives us the ground for a system that can be designed and developed to assist journalists.

4.1 Tools & Technologies

This sction gives an overview on the tools and technologies e,g languages and frameworks, employed to implement our solution.

4.1.1 Frontend UI

The following tools and technologies are used to implement the frontend or client-side application.

4.1.2 Frontend UI Application Content

In the development of the client side application content is based on the HTML (Hyper Text Markup Language). Hyper Text Markup Language (HTML) provides the foundation for creating the overall structure of web pages, allowing for the arrangement of text, links and other elements in a structured manner.

4.1.3 Frontend UI Application Layout

In order to enhance the visual representation and keep the UI principles in mind, the layout is based on the Cascading Style Sheet (CSS). The appearance of the HTML elements are customized by CSS, which include color, font etc.

4.1.4 Frontend UI Application Logic

For implementing the logic of the front end application JavaScript is used. Javascript plays an important role in the implementation of the logic of frontend applications. Interactivity and dynamic behavior of pages is made possible by Javascript.

4.1.5 Backend/API

Python and Flask Framework

The backend/API layer serves as the crucial component The whole application logic for the server side is implemented in Python and Flask framework. Python serves the primary purpose of our choice, i.e., processing data and uploading it into storage, retrieving it, handling incoming requests, and generating responses. The reason for choosing Python over the other languages is because of its simplicity and functionalities and the presence of a wide range of libraries compared to other languages, thus making it well-suited for implementing complex server-side logic.

In the Backend/API, we used Flask which is a lightweight Python framework for the web. It establishes a robust foundation for handling API requests from the Frontend UI We defined endpoints for handling the requests through Flask'sFlask's routing system. The different kinds of requests from the Frontend UI involve investigating cases or simply prompting LLM.

Error Handling Mechanism

Also for reliability and usability of the API, error handling mechanism is implemented for ensuring the smooth working of the system and handling of exceptions and providing appropriate response.

4.1.6 Data Modules

Several modules were developed to streamline data retrieval, processing and ingestion with each one serving specific function in the system. We have made use of Lanchain, an open-source Python library for developing modules. It has provided us with a modular and extensible framework for building our application.

Data Retrieval Module

The data retrieval module is also implemented in using Python, a sophisticated system is implemented for the interaction with Azure Vector Store API and retrieving relevant documents from the store. It also uses Azure OpenAI API for converting the query to vector representations through ada-002 embeddings model. This module serves as the bridge between the backend and Azure Vector Store.

Data Processing Module

The data processing module is also implemented in Python to interact with Azure OpenAI API for GPT 3.5 turbo and GPT 4. This module extracts meaningful insights from the retrieved documents and enhances the system's capabilities.

Data Ingestion Module

The module for ingesting the documents from external sources, i.e., Tromsø Kommune, is also implemented in Python. This module consumes both Azure OpenAI for the ada-002 embeddings model and Azure Vector Store API as it both generates embeddings and uploads them together with documents to the vector store.

4.1.7 Database

The database infrastructure plays an important role in not only storing and managing but also in retrieving system data efficiently:

Vector Store

The database we are using is Vector Store or Vector Database. In which we are storing the documents with their associated metadata and embeddings. The reason for choosing vector storage is because of our requirements to have an efficient retrieval of documents. This aspect is quite crucial for the seamless performance of the solution, ensuring quick access to relevant data.

4.2 Deployment

In our deployment, the whole application, including the database and LLM, was deployed and hosted on an Azure cloud platform. This setup includes Azure Vector Store, which is Azure AI search, private instances of GPT 3.5 turbo, GPT 4, and ada-002. Additionally, we have tested a local machine-based LLM, such as the GPT4All.

4.2.1 Azure AI search

The purpose of Azure AI search is to store all of our data, including the embeddings for them together. After the ingestion of documents from various sources, the embeddings are generated and uploaded to the Azure AI search. This integration streamlines the whole process of retrieval and analysis of data, which in turn enhances the overall functionality of our system.

4.2.2 Azure OpenAI service

Within the Azure environment, the Azure Open A.I. service is used to deploy various versions of GPT, including GPT 3.5 turbo, GPT 4 and ada-002 the text embedding model, for generation of embeddings for the documents.

4.3 Implementation Details

This section shed light on overall working of the system, how different modules and components interact to process the queries.

4.3.1 Front-end

In the frontend/client-side application, an interface for search queries is designed and implemented. In which there is a simple HTML form for queries together with a prompt. The form also includes optional factors such as option for filtering based on specific metadata, and number of results to be included. This form serves as the basis for search functionality, by providing users with a simple straightforward mean for queries and prompts input.

The optional features such as number of results and metadata filtering are incorporated for enhancing the customization and search precision. This kind of feature gives the users greater control over search query by enabling them to efficiently navigate and manage the information retrieved.

In the HTML form, except the query, all the fields are kept optional and have some default values if the user does not choose or enter a specific argument. The entered data in the form is collected and sent to the backend through API calls to the backend via Hypertext Transfer Protocol (HTTP) protocol. But before making any API call, the

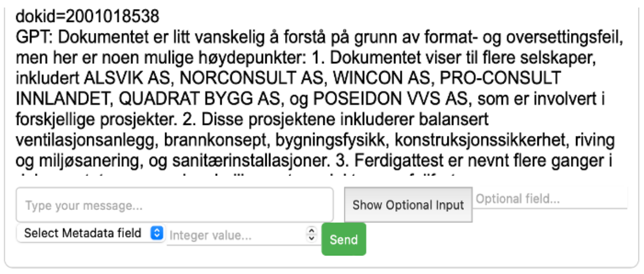

dokid=2001018538
GPT: Dokumentet er litt vanskelig å forstå på grunn av format- og oversettingsfeil, men her er noen mulige høydepunkter: 1. Dokumentet viser til flere selskaper, inkludert ALSVIK AS, NORCONSULT AS, WINCON AS, PRO-CONSULT INNLANDET, QUADRAT BYGG AS, og POSEIDON VVS AS, som er involvert i forskjellige prosjekter. 2. Disse prosjektene inkluderer balansert ventilasjonsanlegg, brannkonsept, bygningsfysikk, konstruksjonssikkerhet, riving og miljøsanering, og sanitærinstallasjoner. 3. Ferdigattest er nevnt flere ganger i

Figure 4.1: A Frontend screenshot of the system.

data format is checked for all the values in order to ensure the type and minimum and maximum limits are met if there are any for any of the values.

In our frontend implementation, simplicity, functionality, and interaction are prioritized. The front end serves as the seamless gateway for accessing and getting the most out of the capabilities of the application.

4.3.2 Backend/API

In the backend implementations, one single file is designated to handle the requests from the client side. The received requests from the front end are processed accordingly, i.e., routing to different endpoints based on the operation. The first step is to start executing the business logic i,e from the incoming query; Vector Store is queried through Azure Vector Store API for retrieval of specific documents, which is a matter of interest to the user. If the user has specified additional inputs, such as metadata filters and a number of documents, the documents, together with their associated metadata, are retrieved within those constraints.

When a query is submitted, the query is first converted to vector by using the same algorithm, in our case, ada-002, that was used for generating the embeddings for the documents in the store previously ingested. The data in the vector store is structured by Hierarchical Navigable Small World Graph (HNSW) to facilitate fast and efficient retrieval. The search algorithm, which has been set in the vector store when creating the index profile, starts for similarity search.

The retrieval of documents is strictly based on similarity search. In the context of vectors and natural language processing (NLP), similarity can be defined as how two different vectors are closely related in terms of direction. For similarity, the two must be in vectors, i,e document, and query. The formula for calculating the cosine similarity is given below for a document D and a query Q:

$$cosinesimilarity(D,Q) = D.Q/||D||||Q||$$

The Q.D. is the dot product, and D.Q. is the Euclidean length of vectors D and Q. The range of cosine similarity is from -1 to 1.

- 1 is perfectly similar.
- 0 is no similarity.
- -1 is exactly opposite to each other.

In the next step, after the documents are retrieved from the store, the resulting documents are passed to the Data Processing Module and given to an LLM for performing operations based on the prompt from the front end. This seems fairly a simple process in which LLM takes in the document or text and the prompt. However, the generation of response from an LLM is completely based on the context provided. In this way, the response is highly optimized, meeting the needs of the client, which any other way LLM couldn't have processed the prompt, resulting in an irrelevant response or failure to process the request.

This is the phase where LLM's potential for investigating the document or case can be used. The potential to investigate any regularity or any potential violation of the rules and also for considering it as a new story. The prompt to the LLM from the user is optional, meaning it should have one prompt as a default set from before. So, by default, it will evaluate the document based on it.

Also, on the server side, we also have the module for ingesting a document and creating embeddings for that ingested document, the Data Ingestion Module. Which will look for a new document after a certain time interval; if there is some new entry, it will start loading it, create the embeddings through Azure OpenAI API, and upload it to the Vector Store via Azure Vector Store API. In case there is some entry, it will load the document and create the embeddings through ada-002, a model for creating embeddings to be consumed by the GPT. The created embeddings, together with the text, are uploaded to the Vector Store in JavaScript Object Notation (JSON) format.

Database

The database we used was Azure AI search, which was deemed the most convenient option for implementing this system. We must keep in mind our prerequisites to have efficient and precise storage and retrieval. In our case, both can be comprehended and fulfilled by Azure AI search. Our goal is to set up semantic retrieval of documents in query execution. It adds language understanding for the processing of search results. So, in turn, promoting the most similar results to the top.

To implement the vector store index for our solution, various configurations, and settings have been carefully selected, including the BM25 similarity algorithm for search accuracy. The search efficiency is optimized with HNSW algorithm, which is used for creating the index structure. Moreover, the search-ability through metadata is also enabled to filter documents based on specific metadata arguments.

CHAPTER 5
Results & Discussion

In this chapter we presented the methodology for evaluation which is being adapted to assess the performance and accuracy of the system and the results are discussed.

5.1 Evaluation

For a system to be able to judge its applicability in the field, the very common practice is to evaluate its performance based on some factors.

5.1.1 Methodology

The methodology for evaluation chosen for testing the effectiveness of this system is that a team of Journalists at iTromsø will present a list of questions and queries, which requires access to the documents. So, the retrieval of documents and the response from the LLM will be evaluated by the iTromsø. We have tested our solution with multiple LLMs, which are GPT4All, GPT 3.5 turbo, and GPT 4. The reason for doing testing with GPT4all is that we wanted to test local LLM, so if we include both private and confidential data in the future, it won't be an issue. GPT 3.5 and GPT 4 are easily available in the cloud on Azure and are some of the most advanced solutions with very promising results. The goal was to identify the most suitable solution for the task.

Evaluation Metrics

- Accuracy: The accuracy of the solution is measured as the ability to retrieve the relevant docs is graded by the journalists at iTromsø.

- User Feedback: The response from the LLMs is evaluated by journalists, assessing and answering the prompt correctly or making sense of the documents provided.

5.2 Results

This section explains the tests we performed, and the gradings of those results from the iTromsø team. A list of prompts presented by iTromsø are given below:

- Propmt1: *Du tar rollen som kommunal saksbehandler av plan og byggesak for kommunen og skal se over følgende dokument. Oppsummer dokumentet og konkretiser hvilke utfordringer som eksisterer for å kunne godkjenne søknaden.*
- Prompt2: *Du tar rollen som kommunal saksbehandler av plan og byggesak for kommunen og skal se over følgende dokument. Hvilken dispensasjon bes det om, hva er det som normalt er stridig med loverk som utbygger ber kommunen om untagelse fra. Hvilke konsekvenser kan dette ha?*
- Propmt3: *Du tar rollen som lokal journalist og skal se finne hvilke nyhetsverdige poeng som kommer frem. Oppsummer kort korrespondansen og påpek hva klagen dreier seg om og på hvilket grunnlag klagen avises.*
- Propmt4: *Oppsummer og kom med vedtak som benyttes. Hvile poeng har betydning for befolkningen i nærområdet?*

5.2.1 Original Queries Results:

The figure 5.1 depicts the results from the query as the documents retrieved. In figure 5.1There each color represents a separate query. On the x-axis we have the documents (D1, D2, D3) and on the y-axis is the score in terms of relevancy. One thing that should be noted is the same score for different documents for the same query represents the duplication of document. The list of queries are given below:

- Q1: *Mangler ved søknad om ferdigattest CONSEPT EIENDOM AS*
- Q2: *Dispensasjon Juldagan Skir Peab Bjørn Bygg*
- Q3: *Kræmer Eiendom klage avises*

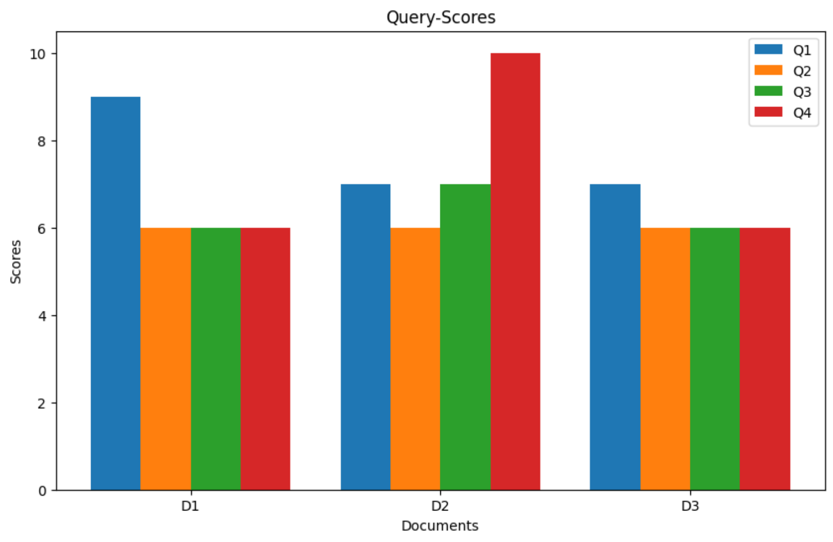


Figure 5.1: The retrieved documents score for each query from the original list of queries.

- Q4: *Rammetillatelse Quadrat Eiendom AS*

The analysis of the original queries (**Q1, Q2, Q3, Q4**) revealed insightful patterns regarding the relevance of retrieved documents. Here's a summary of the findings:

- Query 1 (Q1): All documents received relatively high scores, indicating general relevance to the query.
- Query 2 (Q2): Consistently high scores but duplication of documents in all three.
- Query 3 (Q3): Document 2 exhibited the highest score, while Document 1 & 3 got the same score.
- Query 4 (Q4): Again, Document 2 has the highest score of all, suggesting irregular relevance.

The analysis of query-score has provided insights according to the relevance and effectiveness of our proposed system in retrieving documents based on journalist queries. So from the figure 5.1 we see that most of the queries scores depicts out of order retrieval, and also shows that in only two cases the most intended document has been retrieved. Which cannot be seen as a great result to be expected. The expected result we were hoping for is the maximum relevance of the document to the query. The analysis of query-score has provided insights according to the relevance and effectiveness of our proposed system in retrieving documents based on journalist queries.

It is observed that except for the **Q1** all other queries have mixed varied results. The retrieved documents not showing relevance as supposed to be in similarity search in decreasing order, the relevance score graded by journalists. We have also tested the queries with multiple different prompts, and the results were satisfactory according to the journalists as the LLM answered reasonably.

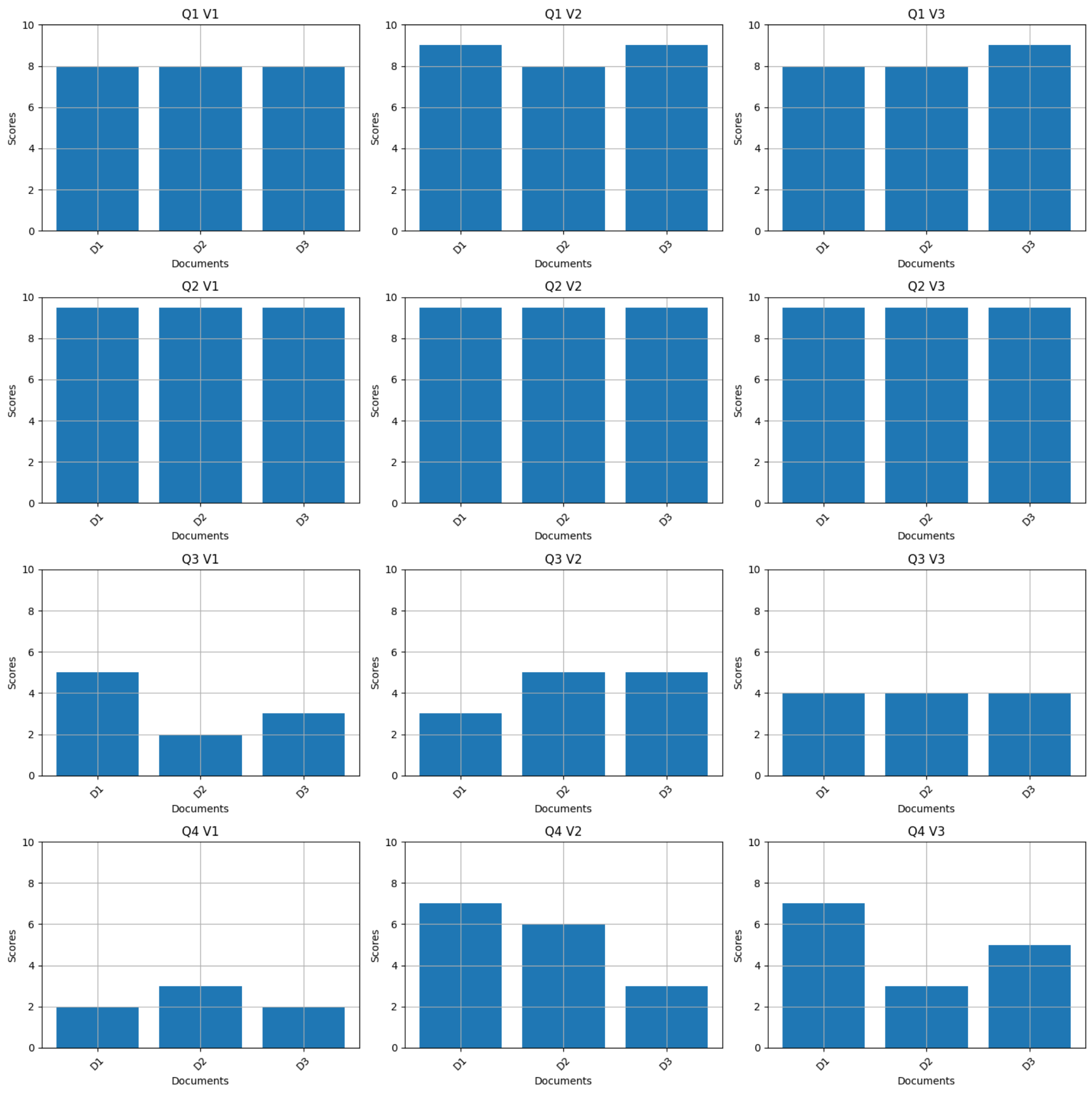


Figure 5.2: The results of the queries generated from original list of queries with ChatGPT.

5.2.2 Query Variations Results:

After testing the original list of queries, a list of queries from each of the original queries was generated through ChatGPT 3.5 turbo. The generated queries were semantically similar to the original query. The generated semantically similar queries are given below:

Query1 Variations:

- *Mangler dokumentasjon ved søknad om ferdigattest CONSEPT EIENDOM AS*
- *Behov for ytterligere informasjon ved søknad om ferdigattest CONSEPT EIENDOM AS*
- *Ufullstendig søknad om ferdigattest CONSEPT EIENDOM AS*

Query2 Variations:

- *Søknad om dispensasjon Juldagan Skir Peab Bjørn Bygg*
- *Spørsmål vedrørende dispensasjon Juldagan Skir Peab Bjørn Bygg*
- *Informasjon angående dispensasjon Juldagan Skir Peab Bjørn Bygg*

Query3 Variations:

- *Kræmer Eiendom klage avises uten grunn*
- *Avvist klage fra Kræmer Eiendom uten begrunnelse*
- *Ugyldig avslag på klage fra Kræmer Eiendom*

Query4 Variations:

- *Rammetillatelse nektet for Quadrat*
- *Eiendom AS Avslag på rammetillatelse for Quadrat Eiendom AS*
- *Manglende rammetillatelse for Quadrat Eiendom AS*

Some of the key observations from query variations (**V1, V2, V3**) from figure 5.2 are explained below.

- Variations of Query 1 (Q1 V1, Q1 V2, Q1 V3): Consistently high scores across all documents indicate that these variations maintained relevance to the original query, showing the same trend as the original query retrieved documents.

- Variations of Query 2 (Q2 V1, Q2 V2, Q2 V3): Similar to Query 1 variations, these variations also showed high scores across all documents, suggesting consistent relevance, but here, the problem is all the retrieved documents are the same, duplicates. Again, the results were the same as the original query retrieved documents, which were also the same but, this time, more relevant.
- Variations of Query 3 (Q3 V1, Q3 V2, Q3 V3): Variations yielded lower scores for documents, indicating varied impacts of semantic variations on relevance. Also, the V3 yielded the same duplicate documents.
- Variations of Query 4 (Q4 V1, Q4 V2, Q4 V3): Mixed results were observed, with some variations performing better for certain documents while others performed worse for different documents.

5.3 Discussion

The testing and analysis of the system, both with the original list of queries and the semantically similar queries generated from original queries, has shed light on the effectiveness of the retrieval system and the impact of semantics variation on the relevance. Below, we first discuss the testing of retrieved documents from Vector Store and then LLM's responses.

5.3.1 Documents Retrieval

The accuracy of the documents retrieved is not quite good, although the documents retrieved from the 1st query and its different variations were quite comfortably the most relevant ones; duplication is the issue seen in its variations. Similarly, the variations of the queries also result in duplication. And for the rest of the original queries and the different versions of them, the results weren't very much of interest to the journalists by not finding the most relevant document.

After testing the system on a list of semantically similar queries generated from an original query, it was also observed that the documents retrieved were not the same as those in the original query. This might be because of the following reasons:

- Semantic Ambiguity: Similar terms may still have differences in meaning even if they are most semantically relevant because natural language is complex, and even if there is a slight change in words, it leads to a complete change in the meaning.

- Vectorization: The process of converting queries to vectors may not always capture the specific intent behind the query and are not close enough together to retrieve the same document.

The duplication is seen as one of the major problems in retrieving documents. This is due to the uploading and presence of multiple copies of the same documents in the data corpus. As in similarity search, the document is retrieved based on similarity, so if it sees one of the duplicate documents as the most relevant, the next most similar would also be the duplicate of the first one because both of them would have the same relevancy to the query.

We tried to solve the problem of duplication through identifying the documents through *"document id"* and other attributes but that also didn't help because there majority of documents have missing metadata values.

The uncleaned text data seem to be the cause behind the inaccurate retrieval of documents from vector store as those special characters and large white spaces were also taken into account during the creation of embedding. The embedding algorithm takes into account them also which results in not able to truly capture the document vector. So in turn it effected the document vector representation and for that reason the document most closer to the query is not retrieved.

By applying metadata filters the retrieved documents were most relevant and are of interests to journalists. For example when specifying the date range, or case id or any sort of metadata attribute the retrieval were according to expectations with the desired documents.

5.3.2 LLM's Response

Assessing the results from different LLMs I,e GPT4All, ChatGPT 3.5 turbo, and ChatGPT 4. There was a significant difference in the response between all of them.

The overall results from LLM were in the following order:

- ChatGPT 4: Outperformed both others in all aspects.
- ChatGPT 3.5 turbo: Found difficulties in getting the sense out of uncleaned documents from special characters.
- GPT4All: Could not answer the prompts in any case.

GPT4All performed the worst of all in very imprecise results for the prompt and also very high time complexity while running locally. We have tested GPT4All in RAG, but the results were below par. Also, because of the very high complexity of the time, it wasn't tested very much.

The results from ChatGPT 3.5 turbo were also not satisfactory, as the structure of the document created problems for it, and it wasn't able to do the summarization or find any irregularities in the documents. The text documents were converted from PDF to text files, due to which there were a lot of special characters and white spaces. And here, the ChatGPT 3.5 turbo failed.

The results regarding the use of ChatGPT 4 were according to the expectations and performed very much better in all aspects compared to the other two LLM's. The ability to summarize the documents containing special characters and white spaces didn't affect its capability. And has also done well accordingly in the prompting.

Overall, the results from LLM's are satisfactory, but the problem lies in the retrieval of required documents. When there is no metadata field specified, the results are not optimal, but after specifying the additional metadata filters like date e,g from-to, case serial number, etc. The results were according to expectations.

CHAPTER 6
Conclusion

In this chapter, we bring our research to a close by reflecting on our findings and observations. Also, it gives recommendations for future work.

6.1 Summary

In this study, we designed and developed a system leveraging an LLM's and vector store for assisting Journalists in their investigative journalism work. By outlining the objective of our project work, described the methodology used in implementing such a system, and key findings from the work were presented.

6.1.1 Main Contributions

The main contributions of our project to the Journalism field and some of the applications are:

- A system developed for efficient retrieval and analysis of documents, resulting in aiding the journalists in identifying relevant information quickly.
- The capabilities of different LLM's were explored in summarizing and processing unstructured documents.
- A vector store-based architecture, implementation for fast and efficient retrieval based on similarity search.

6.2 Implications

For this project, the media industry has several implications and can be integrated into the journalistic processes.

- The adaption of such kinds of systems can significantly improve the processes involving analyzing documents, summarizing them according to prompts, and thus streamlining the process of investigative journalism.
- The introduction of LLM and vector databases can significantly help in uncovering hidden insights from a large corpus of unstructured documents. This can lead to more enhanced, comprehensive reporting that can be more impactful both in terms of readers' interests and problems to be addressed.

6.3 Future Work

The project implementation and testing has uncovered limitations, findings, and potential for improvements as a future research and developments.

- The data should be in cleaned form the TXT files should not have any special character and white spaces while the issue of missing metadata should also be taken care of by finding ways to not effect the retrieving process. By addressing these issues the overall quality of the data can be significantly enhanced, leading to more accurate and robust retrieval.
- One of the potential areas for research is to further optimize and customize, fine-tuning an LLM for journalistic purposes in order to be able to clearly recommend stories for news. For this purpose, collaboration with technology experts and data scientists can help tailor such a system specifically for journalism.
- Multiple data sources can be integrated into vector store e,g government open databases, social media posts feeds. This will provide journalists with comprehensive relevant information for investigating local living conditions. The user interface could be further enhanced keeping in view of multiple data sources made available to investigate any case.
- Further ethical considerations and the public's private data privacy should be prioritized before any data gathering and analysis can be performed. This should be in accordance with and following General Data Protection Regulation (GDPR) guidelines.

In conclusion, the continuation to develop such kind of systems by following the recommendations can put the media industry at the forefront of today's digital age while producing impactful journalism.

Bibliography

[1] G. Salton, A. Wong, and C. S. Yang, “A vector space model for automatic indexing,” *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975. → [p6]

[2] D. Jurafsky and J. H. Martin, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.* Prentice Hall series in artificial intelligence, Prentice Hall, 2000. → [p7]

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” 2013. → [p7], [p8], [p45]

[4] Y. A. Malkov and D. A. Yashunin, “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs,” 2018. → [p9]

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023. → [p9], [p10], [p11], [p12], [p45]

[6] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, “Retrieval-augmented generation for large language models: A survey,” 2024. → [p12], [p13], [p14], [p15], [p45]

[7] A. Piktus, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” 2020. → [p13]

[8] H. B. Abdalla, “A brief survey on big data: technologies, terminologies and data-intensive applications,” 2022. → [p15]

List of Figures

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continous Bag of Words
CSS	Cascading Style Sheet
GDPR	General Data Protection Regulation
HNSW	Hierarchical Navigable Small World Graph
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LLM	Large Language Model
PDF	Portable Document Format
RAG	Retrieval Augmented Generation
TXT	Text File Document
UI	User Interface

You yourself are your own obstacle, rise above yourself.

– Hafez