DECLARATION OF HONOUR

I declare on my word of honour that I have written this Bachelor Thesis on my own and that I have not used any sources or resources other than stated and that I have marked those passages and/or ideas that were either verbally or textually extracted from sources. This also applies to drawings, sketches, graphic representations as well as to sources from the internet. The Bachelor Thesis has not been submitted in this or similar form for assessment at any other domestic or foreign post-secondary educational institution and has not been published elsewhere. The present Bachelor Thesis complies with the version submitted electronically.

Patrik Palenčár
20.04.2025

ABSTRACT

This thesis aims to find a way to create a chat bot for live-stream video content by comparing retrieval augmented generation and prompt engineering approaches combined with Large Language Model (LLM). The main focus is to develop and improve a method that effectively provides live-stream video context to a Large Language Model (LLM). The understanding of context will be achieved with captions of video, which will serve as an input for the RAG or prompt engineering process to provide the best context to LLM as possible. Furthermore this study will explore different chunking and retrieval methods, as the quality of information retrieval mainly depends on this. Lastly, the thesis tries to find the best suitable LLM for the use case and implement the whole project as a service.

Keywords: Retrieval Augmented Generation, Prompt engineering, Large Language Models, Chatbot, Live-RAG

ACKNOWLEDGEMENTS

This is an **optional** page. Use your choice of paragraph style for text on this page. Usually, this space is for thanking your supporters and getting emotional about how grateful you are to everyone.

Contents

Declaration of honour i

Abstract ii

Acknowledgements v

Table of Contents vi

List of Tables viii

List of Figures viii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Research Questions . . . 2
- 1.3 Research Method . . . 2
- 1.4 Thesis Structure . . . 3

2 Background 5
- 2.1 Related Technologies . . . 5
- 2.2 Transformers . . . 5
  - 2.2.1 Large Language Models . . . 7
- 2.3 Chat Bots . . . 8
- 2.4 Retrieval-Augmented Generation . . . 8
  - 2.4.1 Chunking . . . 10
  - 2.4.2 Embedding . . . 10
  - 2.4.3 Retrieval . . . 11
- 2.5 Prompt Engineering . . . 11
- 2.6 Speech to text . . . 12
- 2.7 Evaluation . . . 12
  - 2.7.1 Context Precision . . . 13
  - 2.7.2 Faithfulness . . . 14
  - 2.7.3 Answer Relevancy . . . 14
  - 2.7.4 Context Retrieval . . . 14
  - 2.7.5 Semantic Similarity . . . 14

3 Related Work 15
- 3.1 Academic Research . . . 15
  - 3.1.1 Chat bots . . . 15

3.1.2 Attention is All You Need . . . 16
3.1.3 GPT – Generative Pre-trained Transformer . . . 16
3.1.4 Retrieval Augmented Generation . . . 17
3.1.5 Prompt Engineering . . . 17
3.1.6 RAG Evaluation . . . 18
3.2 State of Practice . . . 18
3.3 State of the Art . . . 19

4 Methodology **21**
4.1 System Architecture and Implementation . . . 21
4.1.1 Software and technologies used . . . 22
4.1.2 System architecture . . . 23
4.1.3 Data Handling . . . 24
4.1.4 Backend Development . . . 25
4.1.5 Chunking . . . 27
4.1.6 Embedding . . . 27
4.1.7 Response Generation . . . 27
4.1.8 Prompt Engineering . . . 27
4.1.9 Evaluation . . . 27
4.1.10 Experimental Visualization . . . 27
4.2 Theoretical Background . . . 27
4.3 Experimental Setup Plan . . . 28
4.4 Evaluation Plan . . . 29
4.5 Limitations of the Proposed Methods . . . 29

5 Summary **31**
5.1 How will I achieve research goals . . . 31

Bibliography **32**

List of Tables

List of Figures

Figure 1.1 Comparison of the two LLMs with RAG vs Prompt Engineering approaches. 3

Figure 2.1 Transformer model architecture. Adapted from [1] 6
Figure 2.2 RAG Architecture Adapted from [2] 9
Figure 2.3 Evaluation diagram from blog about RAG evaluation [**?**] 13

Figure 4.1 Overall Project Flow 23
Figure 4.2 Overall Data Flow 24
Figure 4.3 Data Object passed from video stream 25
Figure 4.4 Folder structure of the project 27

Chapter 1
INTRODUCTION

Recent advancements in the field of artificial intelligence have unlocked many topics for research. The development of large language models (LLMs), like Generative Pre-trained Transformers (GPT) [3], has made AI the most talked-about area in modern information technology. This thesis explores the application of LLMs in scenarios that need real-time integration of external knowledge, specifically by the use of Retrieval-Augmented Generation (RAG) and prompt engineering. The study will specifically focus on how can we use RAG or prompt engineering with LLMs to create a chat bot capable of communicating about sports video live stream. The key objective is to implement real-time context retrieval to create a chat bot, which will be researched with the aim of developing a state-of-the-art solution. There are several limitations that I might be facing during this research such as the need to provide context in real time without access to future data or any problems that could affect the final chat bot.

1.1 Motivation

The motivation behind this thesis lies in improving the viewer experience during live sports broadcasts by integrating artificial intelligence technologies such as Large Language Models (LLMs).

Live sports content is fast-paced, and viewers often face moments where they are forced to miss short segments of the game due to many different interruptions, such as stepping away from the screen or answering a call. In these situations, being able to ask an intelligent assistant capable of providing truthful and relevant information about the match in real time can significantly improve the overall watching experience. It would make the viewer feel like watching with a friend.

Live audiences often seek additional information about players, teams, statistics, or specific events occurring during the game. Rather than manually searching through various sources, viewers would benefit from a conversational interface (chat bot) that allows them to ask questions about the match directly while watching. This kind of interaction not only makes it easier for people to get the information they want, but also connects understanding of additional knowledge and watching in a smooth way.

This research goes over the practical use case as it contributes to the field of real-time natural language processing and conversational AI. It looks at the integration of Retrieval-Augmented Generation and prompt engineering in a time constrained streaming context, creating something new in comparison to many static LLM use cases. By building and evaluating a chat bot capable of operating on live streamed content, this study aims to explore how modern AI models can be tweaked for time sensitive, dynamic data scenarios.

Lastly, the goal is to build a tool that is both technically useful and easy to use, giving viewers a smarter, more fun, and more helpful way to follow live stream content.

1.2 Research Questions

- How can context from live-stream content be provided to a Large Language Model (LLM) in order to build an accurate chat bot?
  - What are the most effective hyperparameters for Retrieval-Augmented Generation and prompt engineering in the context of a live-streaming chat bot?
- How does Retrieval-Augmented Generation compare with prompt engineering in optimizing the chat bot's performance for live video stream applications?
  - What are effective methods to evaluate and compare the performance of RAG and prompt engineering in this use case?

1.3 Research Method

My research aims to explore techniques for providing context to LLM for live video streaming content to create a chat bot. For this kind of topic quantitative research method will be used. With this approach, I can conclude that findings of the study are based on objective and measurable outcomes.

The data used for the research, consists of commentary captions from replays of live sports games, simulating real data flow. Technique to get these data is a speech to text technology that converts the live stream commentary to text in real time.

Next, to answer the research questions, the study will focus on how to evaluate the chat bot as accurately as possible. For this, it will examine different objective evaluation methods, such as statistical comparisons of system performance or ground truth analysis

[4]. It will create a solid ground for comprehensive analysis between RAG and prompt engineering approach and determine which is most suitable for the use case.

After the best-performing parameters for the chat bot are researched with LLM GPT-4o-mini [5], its performance will be further evaluated using an additional LLM. These LLMs will be chosen based on current LLM benchmarks for the informational chat bot LLMs. Important measures for choosing a LLM for this use case will be further researched in upcoming chapters. The responses generated by both LLMs and the two techniques will then be analyzed and compared in contingency tables as shown in Figure 1.1.

Lastly, the thesis will address the best approach to implementing live-rag as a service that can be used by other developers. To achieve this, a server with different APIs will need to be built.

Measure	RAG	Prompt Engineering
LLM1	SCORE	SCORE
LLM2	SCORE	SCORE

Figure 1.1: Comparison of the two LLMs with RAG vs Prompt Engineering approaches.

1.4 Thesis Structure

This thesis is divided into several parts to guide the reader from background knowledge to the final evaluation of the chat bot.

The structure of this thesis begins with an explanation of needed technologies such as Large Languafe Models (LLM's), Retrieval-Augmented Generation (RAG), and prompt engineering.

Then, the thesis takes a closer look at the parts that were needed for working with live-stream content. This includes how live video captions are handled, how text is prepared and divided, and how the chat bot finds the right information from the context.

After the background and technical explanation, the thesis continues with the step-by-step process of building the chat bot. This includes details about the tools and methods used, and how they were put together to create a working system. It shows how the live data is received, processed, and turned into answers by the chat bot.

In the last part, the chat bot is evaluated and its performance is measured. The results from using RAG and prompt engineering are compared, and the best options for the live-stream chat bot are discussed. This will help answer the research questions. The thesis ends with discussion and by talking about what could be improved and what future work can be done.

Chapter 2
BACKGROUND

The field of AI is currently one of the most active in research advancements, and new scientific papers are being released daily. This chapter introduces the technologies essential for the research and explains why they are important for this study.

2.1 Related Technologies

The technologies used in the research will include the Python programming language, specifically the open-source library LangChain, which is designed for working with Large Language Models (LLMs). Furthermore, two different LLMs will be used and compared in the research. To implement the result of the research as a service, the study will use another Python library, Flask, to create the server. For evaluation purposes, this thesis will leverage the Ragas Python library, which is specifically designed for evaluating Retrieval-Augmented Generation (RAG) pipelines.

2.2 Transformers

Understanding the transformer architecture is needed in the context of this research. As the study relies on models built on this technique. This goes for tasks such as embedding generation, semantic similarity search, and context retrieval.

Transformers were the revolutionary technology that enabled transformer based LLMs to gain bigger visibility in community of researchers and developers. Introduced in the paper “Attention is All You Need”(2017) [1], transformers presented a paradigm shift in how parts of data are processed in natural language processing (NLP) tasks. Differently from previous architectures such as recurrent neural networks (RNNs), which processed

data sequentially and were limited by vanishing gradient issues, transformers allowed for parallel processing of sequence data. This significantly improved efficiency and scalability.

At the core of the transformer architecture is a technique called self attention. This enables the model to look at all the words in a sentence and decide which ones are most important for understanding each word. Because of this, the model can understand connections between words even if they are far apart in the sentence, which helps it better understand the overall meaning compared to older models.

The architecture of transformers is build of an encoder and decoder structure, however many modern applications, such as BERT [6] and GPT [7], use only the encoder (BERT) or decoder (GPT) for specific tasks. Each encoder or decoder layer has several parts: A multi-head self-attention mechanism that helps the model focus on the most relevant words in the sentence. A small neural network (feed-forward network) that processes each word individually after the attention step and additional techniques such as layer normalization and residual connections, which help the model learn better and train more efficiently.

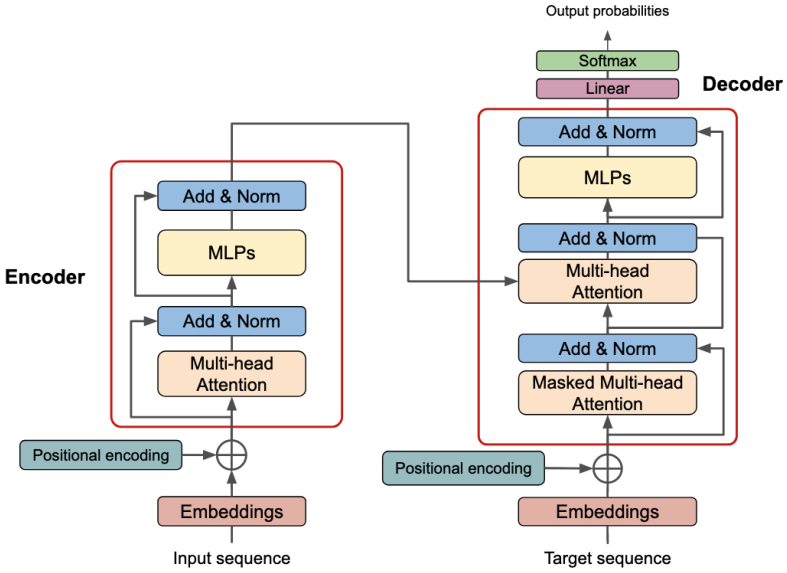

Figure 2.1: Transformer model architecture. Adapted from [1]

One problem with self attention is that it does not care about the order of the words in the sentence. Therefore a key innovation of transformers is the use of *positional encoding*, which provides information about the position of words in the sequence. This allows the model to maintain an understanding of word order.

Transformers have become the foundation for many state of the art models in NLP, such as image processing, speech recognition, and protein folding. Their modularity, scalability, and ability to generalize across tasks have made them a base of modern AI systems.

2.2.1 Large Language Models

The invention of transformers enabled the creation of transformer based LLMs which are also used in my thesis. These models are trained on a huge amount of text data, from which LLMs learn to understand and generate human language. Thanks to the transformer architecture, LLMs are faster and more accurate than older models. Thanks to this, many new use cases started to appear, including the topic of this study.

LLMs are trained by showing them lots of examples of human text. Out of this text the model learns how words and sentences are usually built. The results of this training are saved in the form of model parameters, which are just numbers that help the model decide what word or sentence to generate next. The more parameters a model has, the more information it can store and use and it also needs more computer power to run.

Model size: LLMs can have from millions to even hundreds of billions of parameters. For example, GPT-2 has 1.5 billion parameters, while GPT-4 is believed to 1.7 trilion parameters. A bigger model usually gives better results, but it is also more expensive and slower to use.

Model specialization: As the cost gets higher as the vague LLMs get more parameters. There are also many LLMs that are trained for specific tasks like solving math problems, writing code, or working with images. Since this research is focused on text, the model used must be good at language understanding and generation. So the model must be specialized for text-based tasks like chatbots, summarization, or question answering. Additionally, the LLM used for this study should be smaller and efficient, even if it is not as smart as a big, slow one.

Tokenization: LLMs do not read text word-by-word like humans. Instead, they split the text into smaller parts called tokens. A token can be a word, part of the wort or just a letter. The model understands and processes everything using these tokens. Understanding tokenization is useful when you want to know how much text can fit into the context window or how much the model will cost to run.

Context window: One important parameter of LLMs is the size of their context window. This shows how much text the model can process at once. The more context is provided to the LLM the better responses it tends to give. But this also makes the LLM slower to process the context. A large context window is important for the prompt engineering approach of this research as we need the LLM to process massive prompt. Good thing is that the most of the newest LLMs have big context windows of 100 000 tokens and more.

Temperature: The next parameter of the LLM is temperature. This setting controls how random and creative the model's answers are. A lower temperature (like 0.1 or 0.2) makes the model give more safe and focused answers. A higher temperature (like 0.8 or 1.0) makes the model more creative, but it can also start making things up and hallucinate. In

this research, the chat bot needs to stay close to facts, so lower temperature settings are more useful.

LLM usage and token pricing: Most public LLMs (like GPT models from OpenAI) are not free. They charge users based on the number of tokens used. For example, asking a very long question or giving a lot of tokens context costs more than asking something short. Also, different models have different token prices depending on their size and quality. This is important when deciding which model to use in production or for experiments.

2.3 Chat Bots

Chat bots are computer programs designed to simulate conversation with human users. They are used in many areas today, from customer support to personal assistants or helping users interact with websites or apps. A chat bot usually understands what the user writes and tries to respond in a helpful way.

Traditionally, many chat bots are so called straightforward chat bots. This means the LLM used for the chat bot receives just the user's message (query) together with a small text prompt called a context prompt. A context prompt includes written instructions or rules that help guide the LLM on how to answer. These prompts can include things like tone, personality, or specific information about the topic. They are often improved using a technique called prompt engineering [4]. Which means carefully designing the prompt to get better answers from the LLM.

However, there are some limits to this straightforward approach. These chat bots can only work well if all the information they need is already in the model or the prompt. If they need to answer based on data that changes often or is not known in advance (like live sports commentary), this approach becomes difficult.

This thesis focuses on developing a chat bot that goes beyond the basic "straight forward" method. To handle more complex or dynamic information, one more step of retrieval-augmented generation (RAG) or advanced prompt engineering techniques is added. With RAG, the chat bot can first look up information from external sources, then use the information (called context) to answer the user's question. This makes the chat bot more flexible and useful for live-stream use cases. More about how RAG work is described in the next section.

2.4 Retrieval-Augmented Generation

the Retrieval-Augmented Generation (RAG) introduced in [2], is a process that enhances responses of Large Language Models (LLMs) by providing custom context. Through RAG, LLMs can effectively use specific data that may not be publicly accessible or that the model was not initially trained on. Common data for RAG use cases are for example confidential

data in a company or dynamically updated data — such as the data used in this thesis. This allows LLMs to generate responses with contextually relevant information to specific use cases. By augmenting the input with relevant external information, RAG strengthens the LLM's understanding and reduces the inaccuracies or irrelevant outputs, commonly known as "hallucinations". These might otherwise occur without contextual guidance.

RAG is useful in situations where the base model is not allowed or able to learn all the information in advance. This can include live updates, private data, or large collections of documents. Instead of fine-tuning the LLM with this data (which can be expensive or slow), the data is stored separately and only retrieved when needed.

In simple terms, the process of RAG has two main parts:

Retrieval – finding the most relevant documents or pieces of information from a database.

Generation – giving this information to the LLM along with the user's query so that it can generate a more accurate answer.

This creates a dynamic system that instead of using only the model's pre-trained knowledge becomes more like a smart assistant that looks something up before answering. Because of this, RAG is very useful in real-time applications like the live sports chat bot researched in this thesis.

To make this work, the external information (such as transcripts or documents) needs to be prepared in a special way. This process includes three key steps:

Chunking – splitting long texts into smaller parts that are easier to search through.

Embedding – converting each chunk into a vector (a list of numbers) so that the system can search them and understand their meaning.

Retrieval – finding the most similar chunks based on the user's question and generating the answer based on it.

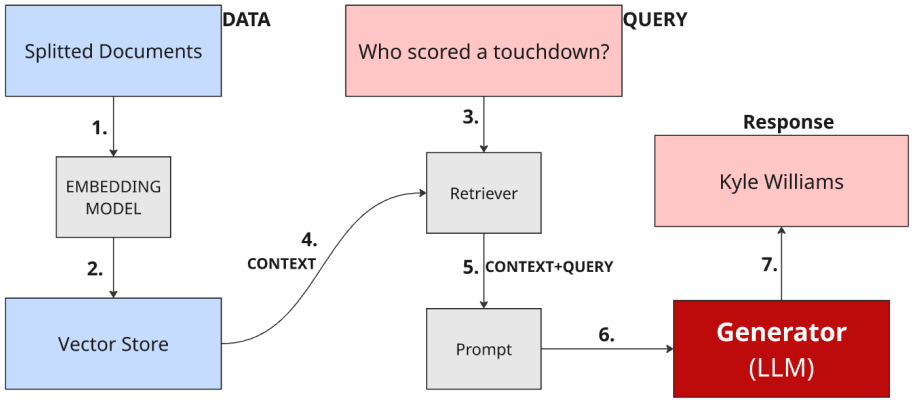

Figure 2.2: RAG Architecture Adapted from [2]

The following sections will explain each of these steps in more detail.

2.4.1 Chunking

Chunking methods discussed in [8] serve as rules to split the text into parts that can later be searched and retrieved. Chunking is an important step in RAG systems because Large Language Models (LLMs) have a limited context window, meaning they cannot process very long texts at once. And if the provided context is long the LLM looses its efficiency and has higher cost. By dividing documents into smaller parts, the system can later choose only the most relevant ones to include in the final prompt.

In this thesis, the use case deals with shorter but information dense textual data, such as live sports transcripts. This is different from most RAG pipelines which usually work with long articles or documents. Because of that, chunking parameters need to be carefully selected and tested.

Important parameters include chunk size (how many tokens or sentences are in one chunk) and chunk overlap (how much text is repeated between neighboring chunks). These affect both the retrieval quality and performance of the chatbot. Finding the right balance is important — too short chunks may miss context, while too long ones may be not effective or reduce precision.

2.4.2 Embedding

Embedding is the process of turning chunks of text into numbers (vectors) so that a retrieval method can understand and compare them. Each chunk is converted into a vector where similar texts get vectors that are close together in the vector space. This is very useful for searching. For example, when a user asks something, the retrieval method can also turn the question into a vector and then find the most similar chunks based on distance between the vectors.

In RAG, embeddings are used for both storing the chunks and for comparing them to the query. A good embedding model makes sure that chunks with similar meaning are placed close to each other, even if the wording is different.

There are different models available for generating embeddings. These models are usually trained on large amounts of text to understand the meaning behind words and sentences.

Choosing the right model depends on the use case. Some models focus on speed, others on precision. For this thesis, where the texts are short and fact-rich (like sports commentary), the embedding model must be able to capture fine details and return highly relevant chunks.

In short, embedding is a key step that connects the question from the user to the correct pieces of information in the data.

2.4.3 Retrieval

Retrieval methods are described in [8]. Retrieval is a part of RAG that searches for text chunks that have a similar meaning to the user query. Chunks of text are retrieved by using a retrieval method and then provided to the LLM together with the question and prompt as context. This step is important because the LLM does not have access to all data and relies on these selected chunks to generate a good answer.

There are different retrieval methods to chose from. The selection of the right retrieval method depends on the data and the use case.

In this thesis, similarity search is used for the retrieval part of RAG. This method compares the meaning of the user query with the meaning of the stored text chunks. It finds the chunks that are most similar to the question and sends them to the LLM as context for answering.

Similarity search works by comparing vector embeddings. Both the query and the text chunks are turned into vectors using an embedding model. These vectors are then compared using a similarity metric, such as cosine similarity. The more similar the vectors, the more relevant the chunk is to the query.

This method is especially useful when the question and the text do not use the exact same words but still talk about the same topic. This is common in natural language and makes similarity search better than simple keyword matching.

Also, retrieval is usually the step where ranking happens. The selected chunks are ordered based on how relevant they are to the question. This ranking plays a big role in the quality of the final answer, as the LLM receives only a limited number of top-ranked chunks due to the context window size.

2.5 Prompt Engineering

As different LLM-powered agents and chat bots began to appear, the need to navigate and customize their responses was increasingly needed. LLMs, particularly earlier versions, often produced outputs that were vague, generic, or factually inaccurate [9]. To minimize these issues and improve the quality of generated responses, researchers introduced techniques now known as prompt engineering techniques.

Prompt engineering is the process of designing and optimizing the input prompts given to an LLM, in order to have its output in a desired way. This includes not only the writing of the query itself but also additional instructions or contextual information that can influence how the model interprets the task.

As discussed in [4], there are numerous prompt engineering strategies. These techniques sound simple and straightforward, but they are effective. Some of these are: rephrasing the prompt for clarity, explicitly stating the desired format of the response, assigning the LLM a specific role (e.g., "Act as a sports analyst"), or using techniques like resampling

to generate multiple outputs and choose the best one. More advanced approaches may also involve few-shot prompting, chain-of-thought prompting, or the use of external tools for dynamic prompt generation.

In this thesis, prompt engineering is an essential part of guiding the chatbot's behavior, especially in combination with retrieved context from the RAG process. Since the effectiveness of an LLM can be heavily influenced by the way it is prompted. A careful prompt design has a crucial role in securing that the chat bot's responses are relevant and accurate.

2.6 Speech to text

2.7 Evaluation

To assure the quality and reliability of the chat bot developed in this thesis, a powerful and systematic evaluation framework is needed. Without clearly defined evaluation metrics, it would be difficult to objectively assess whether the system meets its intended goals, such as providing accurate, contextually relevant, and trustworthy answers. Evaluation enables both the identification of system strengths and the exposure of weaknesses.

There are several approaches to evaluating a chat bot, from subjective manual human annotation to automated objective metrics based on language similarity or information retrieval performance. Since this thesis focuses on a chat bot built using Retrieval-Augmented Generation (RAG), the evaluation is structured around both the generation and retrieval components. The selected evaluation metrics align with the key goals of such a system: retrieving meaningful context, generating faithful and relevant answers, and minimizing hallucinations.

To be able to create results with all these metrics there is need for some data to compare the RAG outputs. For this it is possible to create so called "Ground Truth Question-Answer Pairs". These represent the truthful standard against which the RAG system's outputs can be compared. Ground truth QA pairs serve two primary purposes: they provide a reliable benchmark for evaluating the result metrics of the chat bot's responses, and they help identify weaknesses in both retrieval and generation.

In the context of this thesis, ground truth QA pairs are especially important for assessing the semantic similarity and answer relevancy of the system-generated responses. Without them, it would be difficult to perform objective, repeatable, and meaningful evaluations.

There is discussed in PAPER https://arxiv.org/pdf/2309.15217 that for precise and reliable evaluation RAG system needs at least 50 ground truth questions and answer pairs. The number can be much bigger depending on the use case and how big the document data set is.

But how do we define and select these ground truth QA pairs? To answer this, there are several strategies for generating ground truth pairs:

- **Manual Annotation:** Knowledgeable users read through the documents and create questions based on specific factual information. They then write precise answers that are considered 100 percent accurate.
- **Document-Derived Questions:** Questions can be semi-automatically generated from the documents used in the retrieval pipeline. Techniques such as answer aware question generation can be used to create realistic pairs from existing knowledge. After questions generation domain experts create factually correct answers to these questions.

There are also other techniques but understanding of these two is the most important for this study.

Lastly, for the evaluation of different metrics, naturally a LLMs is used. There can be multiple LLM calls for each question and answer pair per each metric what can make the evaluation really expensive.

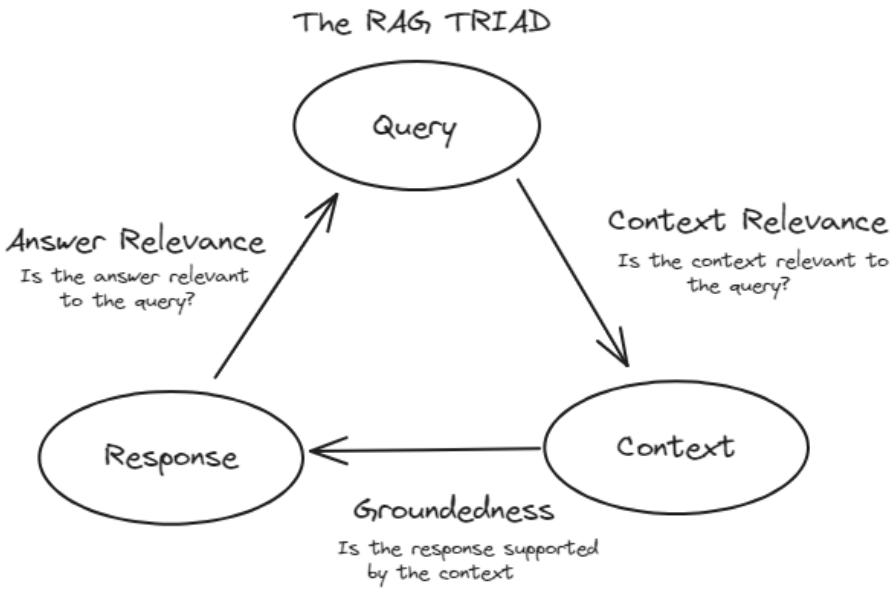

Figure 2.3: Evaluation diagram from blog about RAG evaluation [**?**]

The following subsections describe the evaluation metrics used in this research:

2.7.1 Context Precision

This metric evaluates how accurately the retrieved documents align with the information required to answer the query. High context precision indicates that the retriever component

successfully identifies the most relevant chunks from the knowledge base, reducing noise and increasing the LLM's ability to generate accurate responses.

2.7.2 Faithfulness

Faithfulness means how closely the answer follows the information from the retrieved documents. In simple words, it checks if the response is making things up or if it is sticking to the facts that were actually found. If the answer is based only on the provided context, it is considered faithful.

2.7.3 Answer Relevancy

This criterion checks how well the answer responds to the user's question. Even if the answer is based on the context, it should still be clearly connected to the original question. We can check this by comparing it to correct answers (ground truth questions and answers).

2.7.4 Context Retrieval

Context retrieval evaluation looks specifically at how effective the retriever is in selecting the most appropriate documents. This can be measured using numbers (like recall or precision), or by looking at what was retrieved and comparing it to the ideal context that was prepared in advance.

2.7.5 Semantic Similarity

Semantic similarity measures how close the meaning of the generated answer is to a ground truth answer. This is often computed using embedding based similarity scores (cosine similarity using sentence embeddings). It is not always perfect, but it provides an automated way to approximate answer quality.

Chapter 3
Related Work

This chapter reviews the existing literature and examines the current state of use of RAG and prompt engineering with LLMs in both research and practice. It aims to provide a comprehensive understanding of the advancements and challenges in technologies related to this study. It provides an important foundation for addressing the research questions in this thesis. There are three fundamental concepts critical to understanding upcoming research content: Large Language Models (LLMs), Retrieval augmented generation (RAG), and prompt engineering. This chapter will review the existing literature on each concept in detail.

3.1 Academic Research

Since the topic of the thesis consists of multiple connected parts, it's helpful to explore key research papers that contribute to the understanding of each part.

3.1.1 Chat bots

The idea of chat bots and artificial intelligence was first mentioned by Alan Turing in 1950. He asked a question if a computer program can 'think' and artificially reply in a human like way. This was called the Turing test [10].

The first form of a chat bot, called ELIZA appeared in the years 1964 and 1966 [11]. It was designed to act as a psychotherapist. Since the hardware back then was very limited the technology behind this chat bot was simple. ELIZA worked on identifying keywords in user input, matching them to pre-programmed patterns, and generating responses by rephrasing the input. This gave an illusion of understanding, even though it had no real idea what the user was writing [12].

Later in 1988, another chat bot called Jabberwacky was developed. It was one of the first bots to use some form of artificial intelligence. It worked by remembering past conversations and trying to respond with context based on them. Still, it was far from perfect and often gave slow or incorrect answers [13].

These chat bots improved as the years went by, and they started to be used as a real world products by companies. . Around 2001, chat bots were built into messaging platforms and used for things like customer support, where they helped users by pulling answers from a database. Then, in the early 2010s, big tech companies introduced voice assistants like Siri, Google Assistant, and Alexa. These voice-based chat bots used speech recognition and web services to answer basic questions [14]. Over time, these assistants became smarter thanks to natural language processing, artificial intelligence, and even on-device learning [15].

However, these earlier voice assistants still lacked deeper understanding. In the context of live video streaming, creating such a chat bot wouldn't be possible with these technologies, because the content of a livestream is unpredictable. This all started to change with the introduction of transformers, a new model architecture introduced in the paper Attention is All You Need [1]. Transformers made a huge difference in how machines understand and generate language. They became the building blocks for today's Large Language Models (LLMs), which are the core of modern chat bot systems like ChatGPT.

These improvements opened the door to more intelligent and flexible chat bots,especially those that can work with external knowledge using techniques like Retrieval-Augmented Generation (RAG). That shift from static, rule-based bots to context aware, real-time LLMs is what made projects like this thesis possible.

3.1.2 Attention is All You Need

A major turning point in natural language processing came with the paper Attention is All You Need by Vaswani et al. in 2017 [1]. In this paper, the authors introduced the transformer architecture, which replaced older methods like RNNs and LSTMs. The big idea was that instead of processing words in sequence, transformers look at all words at once using something called self-attention. This allows them to better understand the full meaning of a sentence or paragraph and handle longer-range dependencies.

Thanks to this new architecture, models could be trained much faster and could handle much more data. This also set the foundation for the large language models we use today. Without transformers, modern systems like GPT or Claude wouldn't exist.

3.1.3 GPT – Generative Pre-trained Transformer

Based on the transformer architecture, OpenAI introduced GPT (Generative Pre-trained Transformer) [3] 2018, starting with GPT-1 in 2018. Over time, each version grew bigger

and more capable. The latest, GPT-4 [5], can understand and generate high-quality language in many domains. GPT models are trained in two main steps: first on huge amounts of text (pre-training), and then fine-tuned on more specific tasks (like answering questions or summarizing text).

GPT models are also autoregressive, which means they generate one word at a time by predicting the next most likely word, making them great at continuing conversations or writing responses. Because of this structure, GPT models are widely used in chatbots and other LLM applications today.

3.1.4 Retrieval Augmented Generation

Even though GPT and similar models are strong, they still have a problem: they don't know about new information after their training cutoff and can "hallucinate" facts. To solve this, the paper Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks by Lewis et al. (2020) proposed a method called RAG [2].

RAG combines the power of LLMs with external knowledge. It works by first retrieving documents from a database that are relevant to a user's question and then generating a response based on those documents. This makes the answers more accurate, grounded in real data, and up-to-date. The original paper used dense retrievers and BART-style generators, but modern versions use even stronger models such as GPT-4 or Claude [16].

The study [9] describes a step by step guide for the development of enterprise RAG pipelines. This practical approach outlines this study's solid path to follow. That is interesting for this thesis as the final product is supposed to be production ready.

The paper [4] provides a detailed analysis of prompt engineering techniques, which are critical for customizing the responses of LLMs. It explores different strategies for constructing prompts, optimizing their structure, and adapting them to improve LLM performance. This is relevant to this thesis, as effective prompt engineering is needed for creating an intelligent live RAG chatbot. The insights from this paper can help customize the structure of prompts to deliver accurate responses.

3.1.5 Prompt Engineering

As LLMs got better, people realized that how you give them the information, how you "prompt" them, makes a huge difference. This led to a new field called prompt engineering. The paper [4] provides a detailed analysis of prompt engineering techniques, which are critical for customizing the responses of LLMs. It explores different strategies for constructing prompts, optimizing their structure, and adapting them to improve LLM performance. This is relevant to this thesis, as effective prompt engineering is needed for creating an intelligent live RAG chat bot. The insights from this paper can help customize the structure of prompts to deliver accurate responses.

3.1.6 RAG Evaluation

Finally, since RAG systems combine both search and generation, evaluating them is more complicated. In the paper RAGAs: Evaluating Open-Domain Question Answering with Faithfulness and Factuality by Shuster [17], the authors propose a framework called RAGAs that focuses on different parts of the pipeline, like whether the retrieval step gets useful documents, whether the response is faithful to that context, and whether it actually answers the question.

One key challenge in this area is having good ground truth Q and A pairs and reference contexts to compare the model's output against. Without these, it's hard to say if the system is doing a good job or just sounding smart. The paper also talks about automating these evaluations to scale them across datasets.

3.2 State of Practice

There is currently no chat bot for live-stream video content in practice. Although chat bots created by combining LLM with RAG or prompt engineering approach are currently widely applied across various domains.

Firstly, chat bots are used in customer support, where they improve efficiency and reduce the need for human employees [18]. For example, in eToro,[1] uses customer support chat bot that provides guidance on functionalities of the platform and financial regulations for different countries.

Secondly, chat bots are used in education. They provide personalized learning by retrieving data consisting of course materials from different study fields. Applications can interactively explain topics or give exercises for the student with these data. An example is the Duolingo Lily chat bot [2], which engages users in conversations to improve their language skills. Lily's conversational approach makes learning fun and interactive while helping users practice and retain knowledge.

Furthermore, they are applied in healthcare to help patients by accessing medical databases and retrieving accurate and relevant information. For example, Babylon Health[3] provides a chat bot that offers personalized medical consultations based on patient input. The chat bot evaluates symptoms and offer possible diagnoses, it helping patients understand their condition and determine when to seek professional medical advice.

[1] eToro is an investing broker platform where users can invest in stocks, commodities, and more. Further details at: https://www.etoro.com

[2] Duolingo is a language learning platform with many languages.
https://blog.duolingo.com/duolingo-chatbots/ for more information.

[3] Babylon Health a digital health service that uses artificial intelligence to provide medical consultations https://www.babylonhealth.com/uk for more details.

Lastly, the thesis topic has potential for real-world application, as it is being researched in collaboration with the video stream solutions company Bitmovin.

3.3 State of the Art

Currently, there is no state of the art solution specifically for live-RAG chat bots. However, advanced document-based chat bots have been already studied. For example, dynamic RAG has been explored in different applications, such as the approach described in [19]. It focuses on adapting retrieval and generation processes dynamically based on the needs of large language models.

Chapter 4
METHODOLOGY

This chapter presents the practical implementation of the proposed research. It writes about a detailed description of the technical choices, system design, development process and evaluation. It begins with the selection of right software tools and programming languages used for the projects's development. This is followed by a explanation of the system architecture, which illustrates how the components interact with one another through data flow mechanisms and API endpoints.

Next, it will be described how are the data handled, collected, preprocessed, and managed to support efficient processing. Backend development is discussed with a focus on the programming language and frameworks in the language used for implementation. Next, the chunking strategy is explained, detailing how documents are divided into smaller units to support retrieval-augmented generation. The embedding step covers how these chunks are transformed into vector representations using a suitable embedding model and different models are introduced as well.

Continuing with explanation of response generation where is shown how did I implemented the system so the relevant chunks are used to generate meaningful and context aware responses. Finally, the chapter concludes with a discussion of the evaluation process and visual representations of experimental results to understand the best parameters for the chat bot.

4.1 System Architecture and Implementation

The project consists of three main parts. First is the video live stream which is the main data source. The second is the frontend, which holds the web browser user interface. It is responsible for collecting data from the video stream and communicating with the third component, the backend server. The backend holds most of the project logic. From

embedding of the data till generating the respose and evaluation. The following sections describe each component in detail and explain how they are integrated to form a connected system.

4.1.1 Software and technologies used

The decision of which software to use seemed relatively straightforward. Since the project is focused on practical implementation, I prioritized tools that I know already and I am comfortable with. Whole project was developed in the Windows Subsystem for Linux 2 (WSL2), which provides a full Linux environment running directly on Windows. This allowed me to use Linux-based tools and packages.

For code development, I chose Visual Studio Code as the integrated development environment (IDE). It offers good support for Python, Git integration and has a WSL extension that allows editing and running code directly in the WSL environment.

Technologies used for this project are:

1. **Backend**
   (a) **Python** – Programming language used for backend development, evaluation, and result visualization.
   (b) **Flask** – Python framework used for API development.
   (c) **Git** – Version control system used to track project progress in the cloud.
2. **Retrieval-Augmented Generation and Prompt Engineering**
   (a) **LangChain** – Open-source framework that provides tools for working with LLMs and AI technologies.
   (b) **ChromaDB** – Vector database used for locally storing embedded documents.
   (c) **RAGAS** – Framework for evaluating LLM applications such as RAG.
3. **Frontend**
   (a) **TypeScript** – Programming language used for frontend development.
   (b) **BitDash** – Bitmovin's media player framework for adaptive streaming (DASH, HLS), with support for DRM and low-latency playback.
   (c) **Jest** – TypeScript testing framework used for unit testing.
   (d) **HTML and CSS** – Markup and styling languages used to create the user interface.
4. **Result Interpretation**
   (a) **Matplotlib** – to DO

4.1.2 System architecture

This section provides high level overview of the system architecture.

The overall purpose of the project is to enable real-time interaction between users and a domain-specific large language model system based on video stream content. As shown in the diagram, the user interacts with the system by sending a question through the BitDash Player interface. This frontend component gets the query and sends it to the server, where is the project logic based on RAG. This setup allows users to receive accurate answers based of the video content, transforming the passive video consumption into an interactive experience.

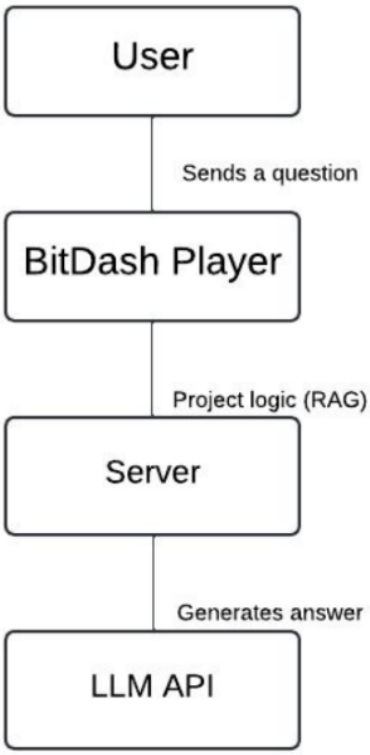

Figure 4.1: Overall Project Flow

The first component is the video stream, which is a data source for the research. The data are then processed in the frontend part of the project. Once the data is cleaned and prepared, it is sent to the backend via two API endpoints:

1. **Embed** - The purpose of this endpoint is to send the data that are ready to be embedded to the backend. This API endpoint has two parameters. First is `"text"`: since the chunking logic is implemented in the frontend part of the project, these chunks are sent independently to the backend, where the embedding logic is executed. The second parameter of this endpoint is `"sourceID"`. In the research I thought that there might be multiple video stream data sources in the embedding database. So each

chunk that is sent to be embedded has its source identification tag based on the video stream source. This means that all embedded chunks from one source end up in the same vector database.

2. **handleQuery** - The role of this endpoint is to handle user queries. It accepts two parameters that are sent to the backend. First `"query"`: this contains the user's input, which is sent in JSON format to the backend for processing. Second `"character"`, which defines the chat bot assistent's persona in the final prompt. This parameter is intended to be configured on the frontend by a system administrator. This parameter value can for example be, "American Football Expert", "Tennis Commentator", "Basketball Enthusiast" etc. This flexibility enables the assistant to act in the conversation, in communication style based on the video stream sport type.

Last part of the architecture is the backend. It handles the core logic of embedding, querying, response generation and evaluation.It retrieves relevant contexts from embedded video data and formulates a prompt for the LLM API. The LLM API then generates an answer based on this prompt and returns it to the backend that passes the response to frontend. All the main components are also shown on the figure 4.2

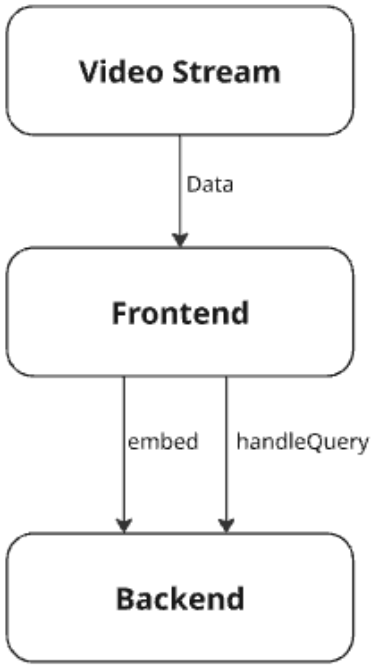

Figure 4.2: Overall Data Flow

4.1.3 Data Handling

The data handling is done mostly in the frontend part. The data with which this project works are commentary captions from sports live stream. These captions are passed by

the stream in an object that can be accessed. This object with captions is send by the HTTP Live Streaming (HLS) video stream every time when there are any words said by commentators. The HLS is a video streaming protocol. It works by breaking down video files in to smaller downloadable HTTP files and delivers them using the HTTP protocol [**?**]. WRITE ABOUT PROTOCOL CREATING LIVE CAPTIONS

On the figure 4.3 is shown the one object passed to the frontend. There are many fields in the object and each field in the object has a specific role. I will explain the most important fields that are used in the project:

timestamp: A numeric value representing the exact time the caption was generated or received. With this field we know when the text was spoken and we can ensure that text stays in the order.

subtitleId and channel: These both show the subtitle track from which the cue originated. This is important for keeping namespaces for each embedding source as discussed in System Architecture section in the API endpoints.

start and end: Floating point values that mark the start and end times (in seconds) of the subtitle cue in the video timeline. These fields may be useful for providing metadata for the chat bot.

text: Contains the actual spoken content extracted from the video stream. It is the most important field because it is the core data used for chunking, embedding and retrieval.

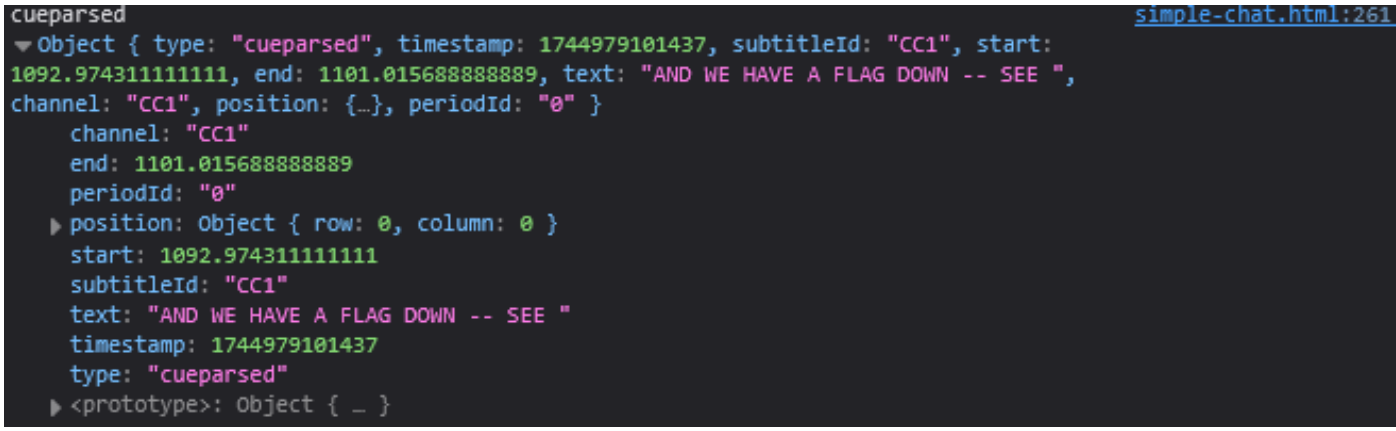

Figure 4.3: Data Object passed from video stream

This structured data object is the foundation for further processing in the frontend, where it is chunked, tagged with metadata (such as source ID), and sent to the backend for embedding into the vector database.

4.1.4 Backend Development

When thinking about the backend implementation the first thing I thought about was a programming language and frameworks. As this project has to do a lot with AI and most of

the frameworks that have tools to work with LLMs are naturally build for Python. However the company stack is fully build in TypeScript language. This made me research possibilities of building the backend in TypeScript with NodeJS for the API endpoint logic an LLamma index framework for the LLMs logic. So the first version of backend server was build in TypeScript with technologies described above. This version worked and communicated with the frontend well. Although as this was only the first version of backend I managed to create only really simple logic for the rag system and it was already difficult because of insufficient documentation. The more advanced methods I wanted to use the less documentation there was and the more time consuming it was. This made me think about rewriting whole backend server to python. And I did so. I recreated whole project into python server with flask for API endpoint handling and langchain for LLM and RAG logic as there is much more documentation and development with this framework.

Figure 4.4 shows the backend folder structure of the project. The main directory is organized as follows:

- `flask_server/src/`: This is the main source directory for the Flask backend server.
  - `chroma/`: Contains ChromaDB collections of embeddings and it is the main data storage of the project.
  - `controllers/`: Holds the controller files, which are responsible for handling incoming requests and dealing with responses.
  - `lib/`: Contains the core logic and functions used across the backend. All main logic implementations are placed here.
  - `middleware/`: Includes middleware components that process requests and responses, such as authentication or logging.
  - `routes/`: Defines the API routes and endpoints. It connects URLs to specific controller actions.
  - `app.py`: The entry point of the Flask application, where the server is initialized and configured.
- `node_server/`: Contains the Node.js server. The first version of the project.
- `venv/`: The Python virtual environment. It manages package dependencies for the backend.
- `.gitignore`, `DISCLAIMER.md`, `README.md`: Standard project files for version control, legal information, and project documentation.

This modular structure has clear separation of roles, making the backend codebase maintainable and scalable.

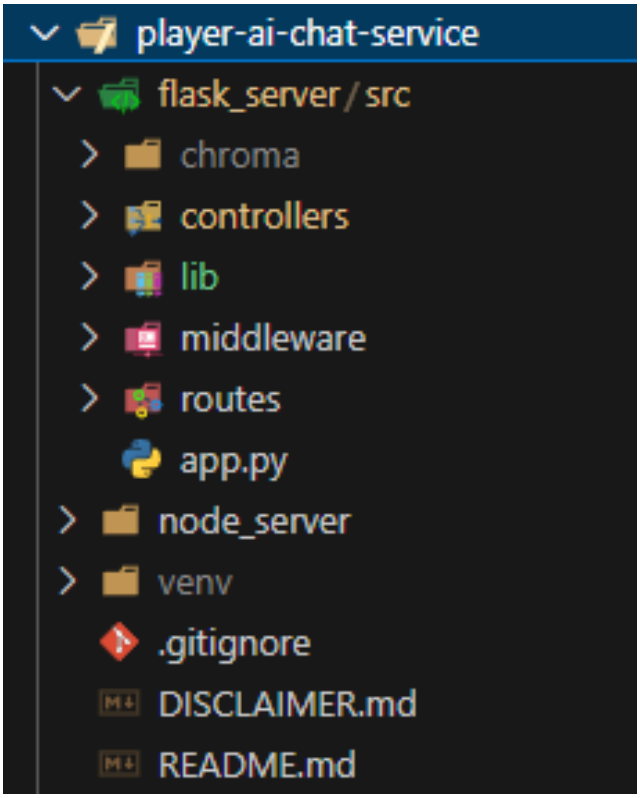

Figure 4.4: Folder structure of the project

Describe how the backend works...

4.1.5 Chunking

4.1.6 Embedding

4.1.7 Response Generation

4.1.8 Prompt Engineering

4.1.9 Evaluation

4.1.10 Experimental Visualization

4.2 Theoretical Background

Before the implementation, we need to address the theoretical background of this thesis. It is important to gain a high level of understanding of the background of different topics that need to be combined to achieve the hypothesized chat bot responses. These are concluded by a comprehensive literature review from past sources until the most recent ones.

The development of transformer architecture revolutionized natural language processing (NLP) by improving the understanding of contextual relationships in text [1]. As a reaction to this research, big organizations developed transformer-based LLMs called generative pre-trained transformers [3] like ChatGPT by OpenAI and by Anthropic Claude [16]. They show the capability of understanding natural language and generating contextually rich and semantically accurate responses as no technology before. Their rise powered a new hype in chat bot technology because of their ability to produce human-like interactions.

This study will examine two different approaches to enhancing LLM responses: retrieval augmented generation (RAG) and advanced prompt engineering techniques. The RAG connects LLMs and external knowledge retrieval to improve response accuracy. By using a retriever module to get contextually relevant information from external knowledge sources and a generator module to produce responses. RAG overcomes the limitations of LLMs, such as dependence on static training data. Prompt engineering focuses on creating precise input prompts to maximize LLM knowledge without external data retrieval. This is achieved by adjusting input, structure, and context within prompts. This method is computationally efficient and relies only on the language model.

4.3 Experimental Setup Plan

The experimental setup for this research involves several key steps to ensure the validity and reliability of the results.

First, a way to get and pre-process the data must be studied. The captions data are encoded in an object with information about the live stream that is sent every 3 seconds. Therefore, a function to extract only the commentary text every time the object is sent is needed. Additionally to the captions, it is possible to get some interesting metadata from the object as well. Furthermore, some metadata from different websites about the match may be scraped.

Secondly, the architecture of retrieval augmented generation (RAG) and prompt engineering needs to be implemented and separately researched. The main focus of the research is to find out what parameters work in both approaches for the chat bot the best.

To compare different versions of chat bots and get conclusions, the best set of objective and subjective evaluation methods for chat bots needs to be studied.

All research on parameters for the chat bot will be conducted using two LLMs: GPT-4-o [5] and Claude-3 [16]. Both LLMs will be compared for each parameter using the evaluation methods. These LLMs are not fixed, as they may be replaced during the research due to the frequent release of new LLMs.

Lastly, there is a need to implement a server with different APIs to be able to use the chat bot as a service.

4.4 Evaluation Plan

By following this evaluation plan, the research aims to provide a comprehensive comparison of chat bot creation using Retrieval-Augmented Generation (RAG) and prompt engineering. The evaluation will be conducted using both quantitative and qualitative methods for deep analysis of the two approaches.

Quantitative Evaluation Methods Quantitative performance will be measured using metrics that track the chat bot's technical effectiveness. These include:

Response Accuracy: Comparing generated outputs with ground-truth responses. Latency: Measuring the time taken to generate responses for each approach. This is particularly important for real-time applications. Precision and Recall: Evaluating how effectively relevant information is retrieved and utilized in the responses.

Qualitative Evaluation Methods Qualitative evaluation will focus on user experience and perceptions of the quality of chat bot responses. As the tester, I will assess aspects such as the flow, logic, and usefulness of the responses.

These evaluation methods will generate a set of metrics to compare different chat bot versions. The results will be presented in comparison tables and graphs, providing a clear visualization of performance trade-offs. This comprehensive evaluation will produce reasonable conclusions about the strengths and limitations of RAG and prompt engineering.

4.5 Limitations of the Proposed Methods

There might be some limitations associated with both researched approaches. For the RAG approach, the size of text chunks might be too small to work with most embedding models. This may lead to incomplete or less meaningful embeddings, which could degrade the chat bot's ability to generate accurate responses. On the other hand, in the prompt engineering approach, the overall text size might be an issue for the context window size of LLMs. This would result in a loss of critical context, which may block the model's ability to understand and respond effectively. These limitations will need to be addressed to ensure the effectiveness of the proposed solutions.

Chapter 5
SUMMARY

The summary will consist of findings based on evaluating different techniques used in the study and sharing my own opinion about it. This will hopefully help us understand the final result of the thesis and answer the research questions.

5.1 How will I achieve research goals

To achieve the research goals I created a systematic step-by-step guide that includes the following steps:

Literature Review: Making a good review of existing literature to understand technologies that are going to be combined and researched in the practical part.

Data Preparation: Getting data from live-stream by function and scraping additional metadata.

Chat bot development: Developing and implementing both researched approaches will be underlined solid ground for iterating evaluation and improvement of the chat bot.

Evaluation: Evaluating the performance of both approaches to determine the best parameters for both of them.

Comparative Analysis: Conducting a comparative analysis to identify the strengths and weaknesses of each approach. This analysis will help in understanding which technique is most effective for the chat bot.

Visualization and Interpretation: Using various visualization tools to present the results clearly, allowing for easy comparison and interpretation of the findings.

By following these steps, the research aims to achieve its goals, providing valuable insights into the effectiveness of both approaches for live-stream chatbot implementation.

BIBLIOGRAPHY

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017.

[2] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[3] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI*, 2018.

[4] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering in large language models: a comprehensive review," *Guangdong Provincial Key Laboratory of Interdisciplinary Research and Application for Data Science*, 2023. BNU-HKBU United International College, Beijing Normal University.

[5] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[8] P. Finardi, L. Avila, R. Castaldoni, P. Gengo, C. Larcher, M. Piau, P. Costa, and V. Caridá, "The chronicles of rag: The retriever, the chunk and the generator." arXiv preprint arXiv:2401.07883, 2024. Preprint.

[9] R. Akkiraju, A. Xu, D. Bora, T. Yu, L. An, V. Seth, A. Shukla, P. Gundecha, H. Mehta, A. Jha, P. Raj, A. Balasubramanian, M. Maram, G. Muthusamy, S. R. Annepally, S. Knowles, M. Du, N. Burnett, S. Javiya, A. Marannan, M. Kumari, S. Jha, E. Dereszenski, A. Chakraborty, S. Ranjan, A. Terfai, A. Surya, T. Mercer, V. K. Thanigachalam, T. Bar, S. Krishnan, J. Jaksic, N. Algarici, J. Liberman, J. Conway, S. Nayyar, and J. Boitano, "Facts about building retrieval augmented generation-based chatbots," 2024.

[10] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, 1950.

[11] M. T. ZEMČÍK, “A brief history of chatbots,” 2024.

[12] D. M. Berry, “The limits of computation: Joseph weizenbaum and the eliza chatbot,” 2024.

[13] E. Adamopoulou and L. Moussiade, “Chatbots: History, technology, and applications,” *ScienceDirect*, 2020.

[14] J. R. Bellegarda, “Large–scale personal assistant technology deployment: the siri experience,” in *Interspeech 2013*, 2013.

[15] A. Inc., “Siri - apple,” 2023.

[16] Anthropic, “Introducing the next generation of claude,” March 2024.

[17] K. Shuster, D. Vyas, P. Mazaré, D. Kiela, and S. Roller, “Ragas: An evaluation framework for retrieval augmented generation,” 2023.

[18] D. Reddy, “Increasing customer service efficiency through artificial intelligence chatbot,” *Revista de Gestão*, vol. 29, no. 1, pp. 19–33, 2021.

[19] W. Su, Y. Tang, Q. Ai, Z. Wu, and Y. Liu, “Dragin: Dynamic retrieval augmented generation based on the information needs of large language models,” 2023.