HOCHSCHULE
MITTWEIDA
University of Applied Sciences

INTERNSHIP REPORT

AI Chatbot Implementation in Low-Code Application.

Mr.
Sandeep Kasinampalli Rajkumar

Course of Study:
Applied Mathematics for Network and Data Sciences

Seminar Group:
MA21w1-M

E-Mail:
skasinam@hs-mittweida.de

Supervisor:
Prof. Dr. Ing. Alexander Lampe
Mr. Diego Vivas

Mittweida, 20. November 2023

Contents

Additional Lists

List of Figures

List of Tables

Listings

Acknowledgment

I wish to express my heartfelt thanks to Professor Dr. Alexander Lampe for approving the topic of my internship and providing me with the opportunity to pursue this endeavor. His guidance and mentorship have been instrumental in shaping my internship experience and ensuring a rich learning journey.

I also extend special thanks to Mr. Diego Vivas, Mr. Reny M John, and Mrs. Reyhan Tosun Bulus, as well as my colleagues at European Computer Telecoms AG (ECT). Their unwavering support and guidance throughout the project were invaluable. Their belief in my abilities was a great source of motivation, and I am thankful for the chance to learn and contribute to the company under their leadership. Their mentorship has significantly aided my professional development during this internship.

Finally, I am grateful to my university for facilitating this internship and creating a platform for students to acquire practical industry experience. Their support has been crucial in making this internship a reality.

Abstract

This internship report offers a thorough analysis of the project undertaken at European Computer Telecoms AG (ECT), which centered on the development of a cutting-edge low-code application for chatbot creation. The project's primary objective was to enable users to construct multiple chatbots, incorporating advanced AI functionalities powered by OpenAI's GPT-3.5 Turbo and Ada-v2 models.

The report commences with an exposition of the project's fundamental objectives, methodologies, and the employed technologies. It progresses to discuss the practical implementation, with a particular focus on the integration of LangChain, a framework designed to maximize the capabilities of language models. A noteworthy feature of the application is its ability to store PDF documents in QdrantDB in a vector format. This allows chatbots to access and relay information from these documents, significantly enriching user interactions with detailed and relevant responses.

Additionally, the report explores the potential future advancements of the project. These include the integration of more sophisticated AI models, improvements in PDF parsing and comprehension, the introduction of user customization features, and the strengthening of security and compliance protocols.

In summary, this report provides a comprehensive overview of the development and prospective enhancements of AI-based chatbot technology within ECT's low-code application. It highlights the promising future and ongoing advancements in the realms of artificial intelligence and chatbot technologies.

1 Introduction

Chatbots have undergone a significant surge in popularity recently, transforming customer service, automating routine tasks, and revolutionizing user interactions across various industries. These conversational AI agents are increasingly utilized in diverse sectors such as e-commerce, healthcare, and finance, where they significantly enhance user experiences and operational efficiency. The advent of advanced natural language processing techniques, particularly Transformer models, has elevated chatbots to unprecedented levels of sophistication, enabling them to understand and respond to a broad range of user inquiries.

The primary goal of this internship was to delve into the realm of sophisticated bot-building tools, semantic search capabilities, and the utilization of various natural language processing services as shown in Figure 1.1. In an era where technology is rapidly evolving and artificial intelligence (AI) is reshaping our interactions, integrating AI-driven chatbots into low-code applications represents a significant innovation. This report thoroughly explores the intricate journey and key insights obtained during the project titled 'AI-based Chatbot Implementation in ECT's Low Code Application.' This initiative underscores our company's dedication to advancing the field of chatbot development. Our ambition was to equip our agents with the tools to effortlessly create multiple chatbots, seamlessly integrating them with pertinent PDF files. This endeavor is set against the backdrop of LangChain, an innovative framework designed for developing applications powered by large language models. A notable feature is the ability of these chatbots to store PDF files in QdrantDB in vector format, leveraging this extensive data repository to provide users with precise and relevant information.

The motivation for this project stemmed from recognizing the dynamic and increasingly crucial role of chatbots in contemporary technology landscapes. Chatbots are essential for enhancing customer engagement, streamlining operations, and providing effective support. However, we identified a significant gap in the market: the absence of user-friendly, low-code platforms for creating AI-driven chatbots capable of accessing, extracting, and presenting information from various sources, such as PDF documents.

1.1 Motivation

Our primary motivation was to bridge the existing gap in the market, offering a versatile solution that caters to both businesses and individuals. Integrating LangChain, renowned for its advanced language models, into our low-code application presented a unique opportunity to revolutionize chatbot development.

Furthermore, we opted to incorporate QdrantDB as the chosen repository and retrieval system for PDF files. This decision was driven by the evident need for efficient and scalable data management. QdrantDB significantly enhances our chatbots' capabilities to deliver relevant and precise information to users. This not only improves user experiences but also plays a crucial role in automating tasks and elevating productivity.

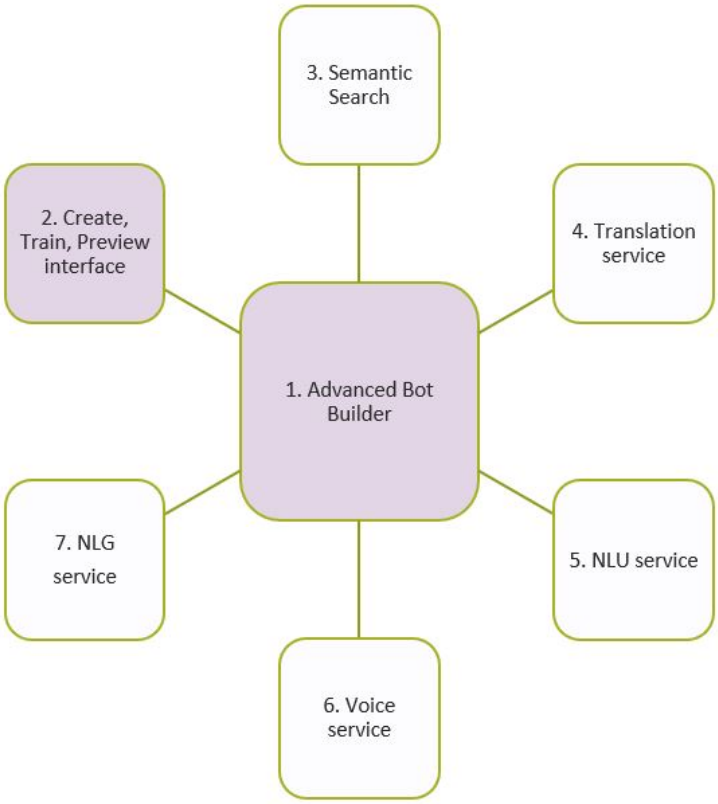

Figure 1.1: Advanced bot-building tools [1].

1.2 Objective

The primary objective of this internship was to develop a chatbot capable of achieving the following goals:

- Understand user intent and extract entities from user queries.
- Provide accurate and personalized responses to a wide range of user queries.
- Integrating seamlessly with advanced bot-building tools, semantic search, and a variety of natural language processing services.
- Ensuring accessibility and user-friendliness, with prospective capability to comprehend and respond to voice commands, which will be further discussed in Section 8.1.

In addition, the fundamental aim of this internship report is to present a detailed and insightful exploration of the 'AI-based Chatbot Implementation in ECT's Low Code Application' project. Our intention is to create a comprehensive document that elucidates the development process, outlines the challenges encountered, and celebrates the achievements realized throughout the project's duration. This report will delve into the methodologies employed, the hurdles overcome, and the concrete results achieved. Our ultimate goal is to significantly contribute to the field of chatbot development and augment the capabilities of low-code applications in utilizing AI for enhanced user experiences and operational efficiency. This

report serves as a portal into the intricate journey of this project, highlighting its importance in the dynamic and rapidly advancing landscape of AI-driven communication and low-code development.

1.3 Structure of the Internship Report

This internship report is structured to provide a comprehensive understanding of the project, from foundational concepts to specific applications and results. The report is organized as follows:

- **Chapter 2: Fundamental Concepts in Artificial Neural Networks (ANNs)** This chapter introduces the basic concepts and underlying principles of Artificial Neural Networks (ANNs). The discussion sets the foundation for a deeper understanding of Transformers, Large Language Models (LLMs), and their significance in the field of AI.

- **Chapter 3: Comparative Analysis of Model Architectures** We explore and compare various model architectures within the field of machine learning, highlighting their unique features, strengths, and areas of application.

- **Chapter 4: Insights into OpenAI's and Hugging Face Contributions** This chapter delves into OpenAI and Hugging Face, examining its groundbreaking contributions and the advancements it has brought to the field of AI, with a particular focus on its renowned models.

- **Chapter 5: Exploring LangChain and Its Applications** We discuss LangChain, its role, and its applications in building AI-powered applications. This includes an analysis of how LangChain enhances the capabilities of language models.

- **Chapter 7: Developing and Demonstrating the Chatbot Experience** Here, we present a detailed account of the experiments conducted during the internship, including methodologies, results, and insights gathered from these activities.

- **Chapter 8: Concluding Remarks and Future Directions** The report concludes with a summary of key findings and insights gained throughout the project. We also discuss potential future work, envisioning the next steps and possibilities for AI chatbot development.

Each chapter is designed to build upon the knowledge and insights from the previous sections, culminating in a comprehensive understanding of AI chatbot development within a low-code application framework.

2 Machine learning concepts

2.1 Artificial Neural Networks

The term Artificial Neural Networks is used to describe mathematical models are inspired by biological neurons. We will briefly describe the Perceptron, the Neural Networks, further we discuss about various Recurrent Neural Networks which we used in our project.

2.1.1 Perceptron

A Perceptron is developed by the idea of biological neuron which has input as dendrites, weights as strength of the input, bias as a threshold function, output as a binary output, and the activation function to activate the neuron.

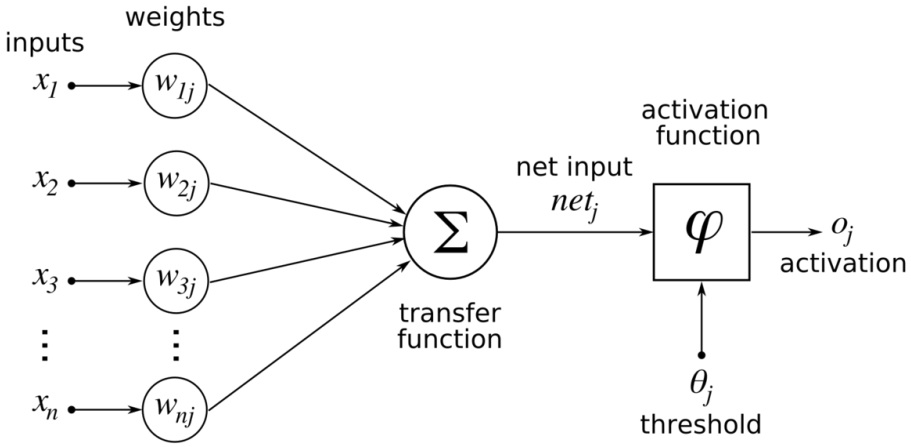

Figure 2.1: Artificial Neuron. [2]

The figure 2.1 shows that the artificial neuron receives inputs. In the context of the artificial neuron, the inputs are represented by a vector $x = (x_1, x_2, \ldots, x_m)$, which corresponds to the dendrites of the biological neuron. The weights of the artificial neuron are denoted by $w = (w_{k1}, w_{k2}, \ldots, w_{km})$, reflecting the strength of the input connections. Lastly, the activation function is responsible for activating the artificial neuron, mirroring the role of activation in a biological neuron.

Definition 2.1

A neuron is characterized by an input dimension $n \in \mathbb{N}$ of the vector x, a weight vector $\mathbf{w} \in \mathbb{R}^n$ and a parameter θ (bias/threshold) where $\theta \in \mathbb{R}$, and an activation $f : \mathbb{R} \to \mathbb{R}$.

We can distinguish two kinds of neurons:

- Based on the (Euclidean) inner product:

$$x \to f(w^T x - \theta) \tag{2.1}$$

with

$$x^T w = w^T x = \sum_{i=1}^{n} x_i \cdot w_i \tag{2.2}$$

- Based on the distance (metric) :

$$x \to f\left(\frac{\|w - x\|^2}{\theta^2}\right) \tag{2.3}$$

2.2 Keyword search and Semantic Search

2.2.1 Keyword search

In the past, conventional keyword-based search engines were the primary means of accessing information. These search engines would scour through a vast collection of documents and present results containing the specified keywords from your search query. However, it is important to explore the inherent constraints of this approach, which we will discuss in the following section.

Limitations of keyword search:

- **Word ambiguity and synonyms:** Some words are ambiguous like 'Jaguar' (which can mean automobile company, animal, American professional football team, or something completely different.

- **Limited Semantics:** Keyword search does not understand the meaning of words or their relationships. It cannot distinguish between different senses of a word or handle synonyms, which can lead to imprecise results.

- **Limited for Multilingual Content:** Keyword search may struggle with multilingual content, as it often depends on the language of the query and the keywords provided.

2.2.2 Semantic Search

The chatbot developed during this internship boasts advanced semantic search capabilities, allowing it to deeply understand and accurately respond to user queries. This advanced functionality enables the chatbot to grasp the underlying intent behind a user's question, even when phrased in varied or unconventional manners. For instance, if a user inquires, 'What is the weather like in New York City?' the chatbot is adept at recognizing the user's intent to know the current weather conditions in New York City and responds appropriately.

Achieving this high level of semantic search capability involves the application of sophisticated natural language processing (NLP) techniques and algorithms. The chatbot employs context-aware language models, enabling it not only to identify key terms but also to comprehend the broader context and significance of the query. It interprets the context, subtleties, and user intent by analyzing the structure and semantics of the question, thereby facilitating more natural, intuitive, and human-like interactions.

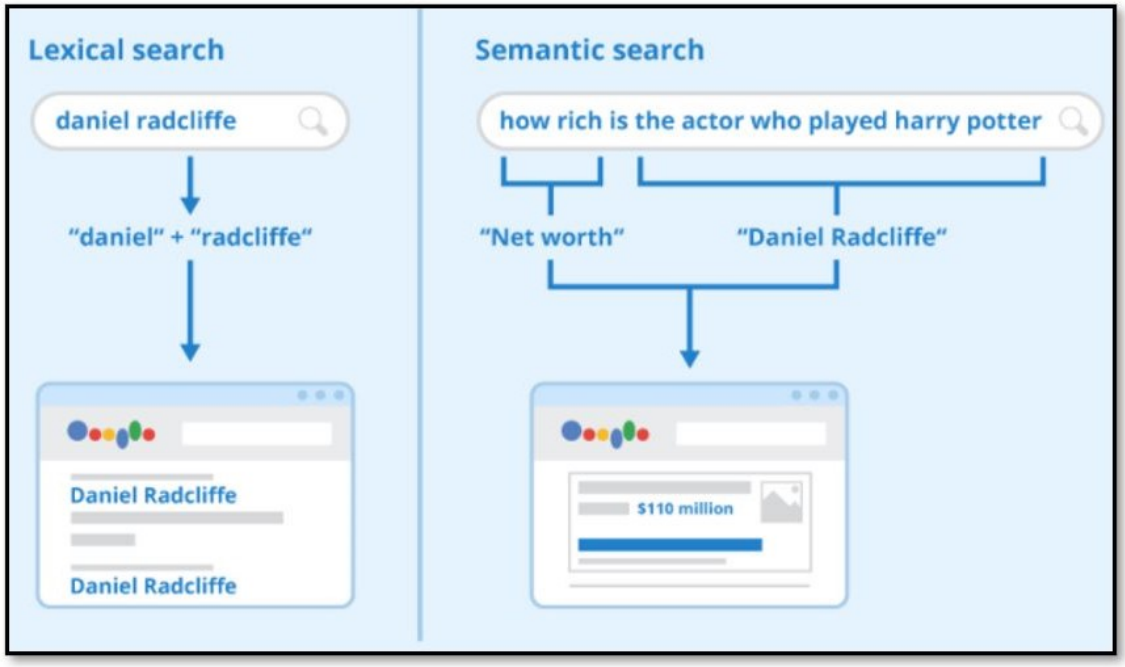

Figure 2.2: Semantic Search.[1]

In practical terms, this means users can pose questions in a more natural and conversational manner, without being constrained to rigid query formats or specific keywords. The chatbot's proficiency in discerning user intentions, coupled with its extensive knowledge base, ensures that users receive accurate and valuable information. This elevates the user experience, making interactions with the chatbot more engaging and effective.

The advanced semantic search capability of the chatbot signifies a commitment to enhancing user experience by delivering intelligent and context-aware responses, irrespective of the variations in users' question phrasing. The role of vector databases in facilitating semantic search will be further explored in the subsequent section.

2.3 Vector Databases

Vector databases are specialized databases designed for the efficient storage and retrieval of high-dimensional vectors. They differ significantly from conventional databases like OLTP (Online Transactional Processing) and OLAP (Online Analytical Processing), which organize data in tabular formats with rows and columns. Vector databases, such as Qdrant, are specifically tailored for use cases in image recognition, natural language processing, and recommendation systems, where data is predominantly represented in the form of high-dimensional vectors. These vectors are stored within a 'Collection' in the database, each comprising an ID and an associated payload.

In these applications, a vector mathematically represents an object or data point, with each component of the vector corresponding to a specific feature or attribute of that object. For example, in image recognition systems, a vector might represent an image, where each vector element corresponds to a pixel value or a descriptor of that pixel.

Vector databases excel in storing and retrieving these high-dimensional vectors efficiently. They utilize specialized data structures and indexing methods to facilitate this, such as Hierarchical Navigable Small World (HNSW) for implementing Approximate Nearest Neighbors and Product Quantization. These techniques enable the databases to perform fast similarity and semantic searches, allowing users to find vectors closest to a given query vector based on specific distance metrics. Qdrant supports several key distance metrics, including Euclidean Distance, Cosine Similarity, and Dot Product.

2.3.1 Distance metrics

Here's a brief overview of these three distance metrics:

2.3.1.1 Cosine similarity

Cosine similarity [3] is a way to measure how similar two things are. Think of it like a ruler that tells you how far apart two points are, but instead of measuring distance, it measures how similar two things are. It's often used with text to compare how similar two documents or sentences are to each other. The output of the cosine similarity ranges from -1 to 1, where -1 means the two things are completely dissimilar, and 1 means the two things are the same. It's a straightforward and effective way to compare two things!

- **Mathematical Formula:**

  The cosine similarity between two vectors **A** and **B** is calculated as the dot product of **A** and **B** divided by the product of their magnitudes.

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{2.4}$$

Here, $\mathbf{A} \cdot \mathbf{B}$ is the dot product of vectors **A** and **B**, and $\|\mathbf{A}\|\|\mathbf{B}\|$ are the magnitudes (or lengths) of the vectors.

To illustrate a similarity search between two sentences, consider the sentences "Hello, World!" and "Hello!". The frequency of each word in the sentences is represented in the table below:

	Hello	**World**
Hello, World!	1	1
Hello!	1	0

Table 2.1: Bag-of-words model for the sentences "Hello, World!" and "Hello!"

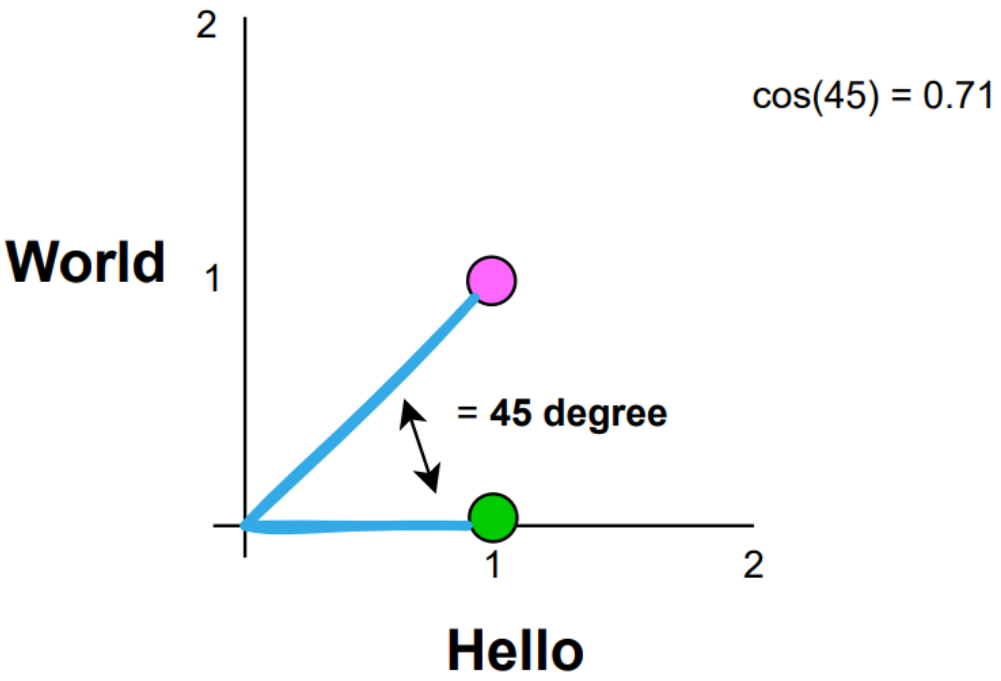

Figure 2.3: Bag-of-words model. [4]

- The first row contains the column headers, which are the words encountered in the text strings.
- The first column contains the text strings themselves.
- The following cells contain the count of the words in each text string.

Calculation:

$$\mathbf{A} = [1, 1], \quad \mathbf{B} = [1, 0]$$

We calculate the cosine similarity using the formula:

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{2.5}$$

Step 1: Compute the dot product ($\mathbf{A} \cdot \mathbf{B}$):

$$\mathbf{A} \cdot \mathbf{B} = (1 \times 1) + (1 \times 0) = 1$$

Step 2: Compute the magnitude of $\mathbf{A}$ ($\|\mathbf{A}\|$):

$$\|\mathbf{A}\| = \sqrt{1^2 + 1^2} = \sqrt{2}$$

Step 3: Compute the magnitude of $\mathbf{B}$ ($\|\mathbf{B}\|$):

$$\|\mathbf{B}\| = \sqrt{1^2 + 0^2} = 1$$

Step 4: Insert values into the cosine similarity formula:

$$\text{Cosine Similarity} = \frac{1}{\sqrt{2} \times 1} = \frac{1}{\sqrt{2}}$$

Step 5: Calculate the numerical value:

$$\text{Cosine Similarity} \approx \frac{1}{1.4142} \approx 0.7071$$

Therefore, the cosine similarity between "Hello, World!" and "Hello!" is approximately 0.7071, indicating a high similarity.

- **Interpretation of the Values:**
  - A cosine similarity of 1 means that the two vectors are identical in orientation, implying very high similarity or even identical content.
  - A value of 0 indicates orthogonality (no similarity).
  - A value of -1 implies opposite vectors.

- **Advantages:**
  - One key advantage of cosine similarity is its independence of the magnitude of vectors. It purely measures the direction, not the length. This is particularly useful in text analysis where the lengths of documents can vary greatly.

- **Limitations:**
  - It does not take into account the magnitude of vectors, which can be a drawback in certain contexts where the magnitude is important.

2.3.1.2 Dot Product

The dot product [5] similarity metric is another way of measuring how similar two things are, like cosine similarity. It's often used in machine learning and data science when working with numbers. The dot product similarity is calculated by multiplying the values in two sets of numbers and then adding up those products. The higher the sum, the more similar the two sets of numbers are. So, it's like a scale that tells you how closely two sets of numbers match each other.

- **Mathematical Formula:**

  The dot product for two vectors $\mathbf{A}$ and $\mathbf{B}$, represented as $\mathbf{A} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{B} = (b_1, b_2, \ldots, b_n)$, is calculated as:

  $$\mathbf{A} \cdot \mathbf{B} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n \tag{2.6}$$

  This operation produces a scalar (hence "scalar product") that is the sum of the products of their corresponding components.

- **Advantages:**
  - Semantic Similarity: It is useful for determining the cosine similarity between words, sentences, or documents, indicating how semantically similar they are.

- **Limitations:**
  - Normalization Needed: The vectors often need to be normalized for the dot product to reflect the true cosine similarity, especially when the magnitudes of the vectors vary widely.
  - Contextual Ambiguity: The dot product might not always capture the nuances of meaning, especially in cases where the context plays a significant role in the interpretation of text.

  In summary, the dot product in NLP is a foundational tool for comparing the orientation of vectors in a high-dimensional space, reflecting the degree of similarity in their directions, which can be interpreted as a measure of semantic similarity in the context of text data.

2.3.1.3 Euclidean Distance

Euclidean distance [6] is a widely used distance metric in various fields, including natural language processing (NLP). It's particularly useful for tasks that involve comparing the similarity or dissimilarity of text data represented in a numerical format.

In NLP, texts (like sentences, documents, or even words) are often converted into numerical representations, such as vectors. These vectors can represent various features like word frequencies, TF-IDF scores, or embeddings from models like Word2Vec or BERT. Once in vector form, the Euclidean distance can be used to measure the 'distance' between these text representations.

- **Mathematical Formula:**

  The Euclidean distance between two points in Euclidean space is the length of a line segment between the two points. For two vectors $\mathbf{A} = (a_1, a_2, ..., a_n)$ and $\mathbf{B} = (b_1, b_2, ..., b_n)$, the Euclidean distance $d(\mathbf{A}, \mathbf{B})$ is given by the formula:

$$d(\mathbf{A}, \mathbf{B}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2} \tag{2.7}$$

  Example in NLP:

  Imagine you have two sentences that you've converted into vectors based on certain features (like word embeddings):

  Sentence 1: "The quick brown fox"
  Sentence 2: "The fast brown fox"

  Suppose their vector representations in a simplified 2D space (for illustration) are:

  Vector $\mathbf{A}$ for Sentence 1: (2, 3)
  Vector $\mathbf{B}$ for Sentence 2: (2.5, 3.5)

  The Euclidean distance between these two sentences, represented by their vectors, would be:

$$d(\mathbf{A}, \mathbf{B}) = \sqrt{(2 - 2.5)^2 + (3 - 3.5)^2}$$
$$d(\mathbf{A}, \mathbf{B}) = \sqrt{(-0.5)^2 + (-0.5)^2}$$
$$d(\mathbf{A}, \mathbf{B}) = \sqrt{0.25 + 0.25}$$
$$d(\mathbf{A}, \mathbf{B}) = \sqrt{0.5}$$
$$d(\mathbf{A}, \mathbf{B}) \approx 0.707$$

- **Advantages:**
  - Word Similarity: In word embedding spaces, it helps in finding words with similar meanings based on their proximity.
  - Information Retrieval: It can be used in search algorithms to find documents that are 'closest' to a query in the feature space.

- **Limitations:**
  - Sensitivity to Magnitude: In high-dimensional spaces, like those common in NLP, Euclidean distance can become less effective due to the curse of dimensionality.
  - Normalization: Often, it's important to normalize the data before using Euclidean distance to ensure that all features contribute equally to the result.

In summary, the Euclidean distance provides a straightforward geometric method to measure the similarity or dissimilarity between text representations in NLP, although it's important to be aware of its limitations in high-dimensional spaces

2.3.2 Which is better for NLP?

- **Cosine Similarity** is often preferred in NLP for tasks such as document similarity and text classification because it is scale-invariant and measures only the directionality of the vectors (i.e., the pattern of word usage), not their magnitude. This is especially important when the length of the documents can vary widely, which is often the case in text corpora.

- **Dot Product** can be useful when you want to measure the similarity and take the magnitude of vectors into account. This might be relevant in cases where the length of the text (like word count) is important.

- **Euclidean Distance** might be used in clustering algorithms like k-means; however, it is less common in high-dimensional text analysis because it can be heavily influenced by the dimensionality of the space.

In summary, while all three measures have their uses, **cosine similarity** is typically the most suitable for comparing the similarity of texts in NLP, primarily because it handles variable-length documents well and is less affected by the curse of dimensionality.

2.3.3 Why do we need Vector Databases?

Vector databases play a crucial role in applications that require similarity searches, such as recommendation systems, content-based image retrieval, and personalized search. They are particularly advantageous due to their efficient indexing and search capabilities, enabling faster and more precise retrieval of unstructured data that is already represented in vector form. This efficiency is vital in ensuring that users are provided with the most relevant results corresponding to their queries.

Additional benefits of employing vector databases include:

- Efficient storage and indexing of high-dimensional data.
- Ability to handle large-scale datasets with billions of data points.
- Support for real-time analytics and queries.
- Ability to handle vectors derived from complex data types such as images, videos, and natural language text.
- Improved performance and reduced latency in machine learning and AI applications.

2.3.4 Qdrant

Qdrant is a production-ready vector similarity search engine offering a user-friendly API for storing, searching, and managing points (vectors), alongside extra information referred to as payloads. These payloads serve as supplementary details that enhance your search accuracy and provide valuable information that can be shared with your users.

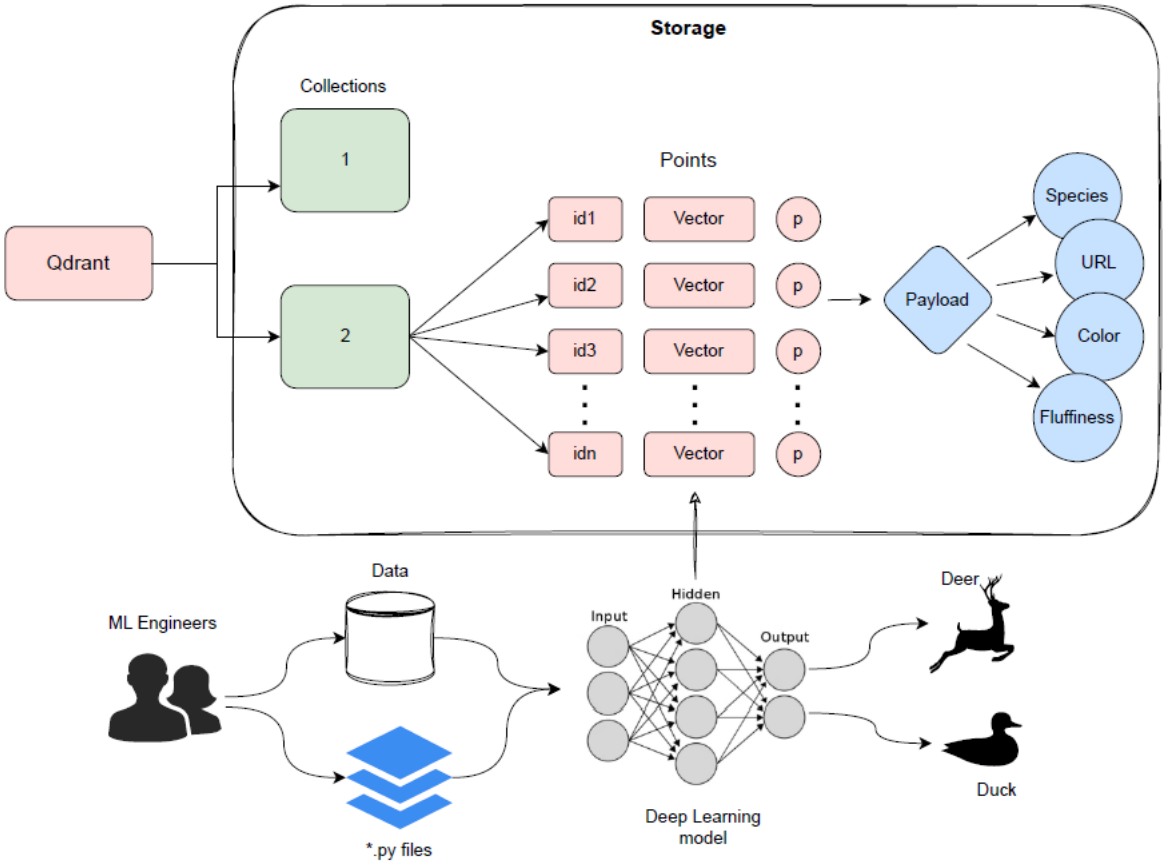

Figure 2.4: Qdrant architecture. [7]

The Architecture diagram represents a high-level overview of some of the main components of Qdrant. Here are the terminologies you should get familiar with.

- **Collections:** A collection is a named set of points (vectors with a payload) among which you can search. The vector of each point within the same collection must have the same dimensionality and be compared by a single metric.

- **Distance Metrics:** These are used to measure similarities among vectors and they must be selected at the same time you are creating a collection. The choice of metric depends on the way the vectors were obtained and, in particular, on the neural network that will be used to encode new queries.

Listing 2.1: Creating a Vector Store Collection

```
self.client.create_collection(
    collection_name=chatbot_name,
    vectors_config=models.VectorParams(
        size=1536,
        distance=models.Distance.COSINE),)
```

  - **size** represents the number of characters for each section of point.
  - **distance** represents we are using COSINE Distance Metrics.

- **Points:** The points are the central entity that Qdrant operates with and they consist of a vector and an optional id and payload.
  - **id:** a unique identifier for your vectors.
  - **Vector:** a high-dimensional representation of data, for example, an image, a sound, a document, a video, etc.
  - **Payload:** A payload is a JSON object with additional data you can add to a vector.

- **Storage:** Qdrant can use one of two options for storage, In-memory storage (Stores all vectors in RAM, has the highest speed since disk access is required only for persistence), or Memmap storage, (creates a virtual address space associated with the file on disk).

3 Comparison Among Models

In the realm of Natural Language Processing (NLP), it is imperative to carefully choose the appropriate model architecture to tackle various linguistic tasks effectively. In this section, we delve into the rationale behind utilizing Transformers in NLP and provide a comprehensive comparison among three prominent model types: Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Transformers.

3.1 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a type of artificial neural network designed for processing structured grid data, such as images and videos. CNNs are particularly well-suited for tasks like image classification, object detection, and image recognition, but they can also be applied to other domains where grid-like data is involved, such as speech recognition and natural language processing.

The key idea behind CNNs is the use of convolutional layers, which are specialized layers that automatically learn and extract features from the input data. These layers apply filters (also called kernels) to local regions of the input data, and as the network learns, these filters detect various patterns and features in the data. Convolutional layers are followed by pooling layers, which downsample the spatial dimensions of the data, reducing the amount of computation required while preserving important information.

- **Applicability:** CNNs excel in processing image datasets and are renowned for their remarkable performance in computer vision tasks.

- **Challenges in NLP:** While CNNs have showcased their prowess in the visual domain, their success in dealing with language data has been relatively limited. They struggle to capture intricate linguistic nuances effectively.

- **Sequential Data Handling:** One key limitation of CNNs in NLP is their inability to retain information from previous steps or tokens. This lack of sequential context comprehension can hinder their performance in tasks where understanding the context is vital.

3.2 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a type of artificial neural network designed for processing sequences of data. Unlike feedforward neural networks, where information flows in one direction from input to output, RNNs have connections that loop back on themselves, allowing them to maintain a hidden state representing information about the previous elements in a sequence. This makes RNNs well-suited for tasks involving sequential data, such as time series data, text, speech, and more.

The key characteristic of RNNs is their ability to capture temporal dependencies in sequential data. Each step in an RNN takes both the current input and the hidden state from the previous step, processes them, and produces an output as well as an updated hidden state. This recurrent structure allows RNNs to consider the context of past elements in the sequence when making predictions or decisions about the current element.

- **Sequential Data Expertise:** RNNs are well-suited for handling sequential data. They can effectively capture dependencies between words or tokens in a sentence, making them valuable in various NLP applications.

- **Natural Language Compatibility:** RNNs perform relatively well in specific natural language processing tasks, particularly those involving sequential or time-series data.

- **Sequential Processing Limitation:** An important constraint with RNNs is their sequential nature. They process data in a fixed order, making it challenging to parallelize training effectively. This limitation can slow down the training process and hinder their scalability for large datasets.

In summary, both CNNs and RNNs exhibit strengths in specific aspects of NLP. However, they are less efficient in processing lengthy sequences and fail to fully capitalize on parallelized training. This limitation becomes particularly evident when dealing with extensive textual data.

3.3 Transformer

Transformers are a type of deep learning architecture that was introduced in the paper "Attention Is All You Need" [8] by Vaswani et al. in 2017. They have since become a foundational and highly influential model in the field of natural language processing and have been applied to a wide range of other tasks as well.

The key innovation of the Transformer architecture is the self-attention mechanism, which allows it to capture relationships between different words or elements in a sequence, regardless of their positions. This is in contrast to traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs), which process sequences sequentially or with fixed receptive fields.

- **Parallelization:** Transformers stand out for their exceptional parallelization capabilities. Unlike CNNs and RNNs, there are no constraints on the order of processing, allowing for efficient use of modern hardware during training.

- **Global Sentence Processing:** Unlike RNNs that process sequences word by word, Transformers can process entire sentences as holistic units. This global approach to sentence processing often results in improved context understanding and performance.

- **Self-Attention Mechanism:** Transformers introduce a novel self-attention mechanism that facilitates the computation of similarity scores between words in sentences. This mechanism enhances the model's ability to capture long-range dependencies and relationships between words, making it highly effective in processing extensive linguistic sequences.

In conclusion, when it comes to processing large sequences in NLP, Transformers emerge as the superior choice due to their parallelization capabilities, global sentence processing, and innovative self-attention mechanism. These attributes make Transformers the model of choice for handling extensive textual data and addressing the complex challenges posed by natural language processing tasks.

3.4 Architecture of Transformer [9][8]

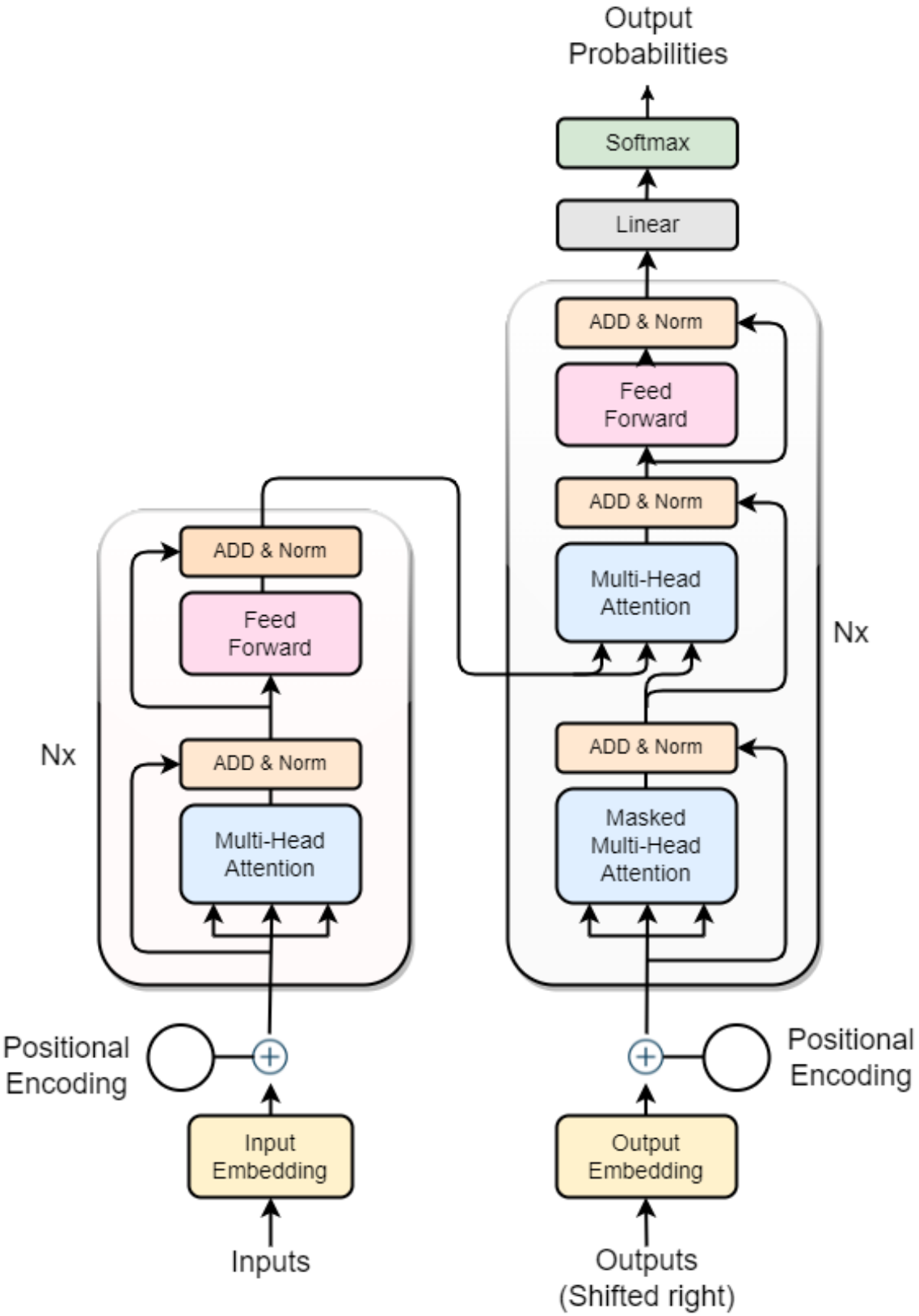

Figure 3.1: The Transformer - model architecture.

In this section, we look at a single block of the transformer. As in GPT-2, there are a large number of transformer blocks present. But for our better understanding, we will see the architecture of a single block of the transformer.

- Z - Embedding vector for i = 1 to n
- X - Inputs representation sequence for i = 1 to n
- Y - Output sequence for i = 1 to n

Transformer consists of 3 main parts:

- **Encoder:** Encoder are identical in structure. It consists of a Multi-Self Attention Layer and a Feed-Forward Network. The encoder's inputs first flow through a self-attention layer— a layer that helps the encoder look at other words in the input sentence as it encodes a specific word. The outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position.

- **Decoder:** Decoders are also identical in structure. It consists of Self Attention Layer, Encoder-Decoder Attention Layer, and Feed Forward Layer. These are to be used by each decoder in its "encoder-decoder attention" layer which helps the decoder focus on appropriate places in the input sequence.

- **Embedding:** Embedding is a numerical representation of words, usually in the shape of a vector. This vector will be all zeroes except one unique index for each word.

3.4.1 Modules in the transformer

Next, I will discuss the elemental components that comprise the original transformer architecture.

- Attention modules
- Position-wise feed-forward networks
- Residual Connection and Normalization
- Positional encoding

3.4.2 Attention modules

The transformer integrates Query-Key-Value (QKV) concept from information retrieval with attention mechanisms.

- **Scaled dot-product attention:**
  Here we are taking a single attention function.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{D_k}}) * V = AV = Z \tag{3.1}$$

  where,
  Q = Queries
  K = Keys

V = Values
D_K = Dimension of key vector.
A is called the attention matrix.

- **Multi-head attention:**
  Similar to above in Multi-head attention we are considering multiple simultaneous layers.

$$MultiHeadAttn(Q, K, V) = Concat(head_1, .., head_h)W^0 \tag{3.2}$$

$$head_i = Attention(QW_i{}^Q, KW_i{}^K, VW_i{}^V) \tag{3.3}$$

where,
$QW_i{}^Q$ = Q value is sent to the Neutral network with the weight $W_i{}^Q$
$KW_i{}^K$ = K value is sent to the Neutral network with the weight $W_i{}^K$
$VW_i{}^V$ = V value is sent to the Neutral network with the weight $W_i{}^V$

3.4.3 Feed-forward neural network (FFNN)

The feed-forward layer is the weight that is trained during training and the exact same matrix is applied to each respective token position.

Since it is applied without any communication with or inference by other token positions it is a highly parallelizable part of the model. The role and purpose are to process the output from one attention layer in a way to better fits the input for the next attention layer.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{3.4}$$

3.4.4 Residual Connection and Normalization

Residual connections are a way to address the vanishing gradient problem that can occur in deep neural networks.

Normalization techniques are used to improve the stability and performance of neural networks. In transformers, the most commonly used normalization technique is layer normalization. Layer normalization normalizes the inputs to a layer across the feature dimension, which helps to reduce the effect of input variations on the outputs of the layer. This can improve the stability of the network and make it easier to learn.

In combination, residual connections and normalization techniques can greatly improve the performance of neural networks used in NLP tasks such as language modeling and machine translation. For example, the original transformer architecture used both residual connections and layer normalization to achieve state-of-the-art performance on a range of language modeling tasks.

3.4.5 Positional encoding

Positional encoding are used to encode the relative positions of the words in a sequence.

A sequence of words is usually fed into the transformer model as a matrix, where each row corresponds to a word in the sequence and each column corresponds to a feature or embedding dimension. However, the transformer model does not have any inherent notion of word position, which is important for understanding natural language.

To solve this problem, positional embeddings are added to the input sequence of words before it is fed into the transformer model. The positional embeddings are learned during training and represent the position of each word in the sequence relative to the other words.

4 Exploring OpenAI and Hugging Face: Frontiers in AI and NLP

This chapter delves into two influential entities in the field of AI and natural language processing (NLP): OpenAI and Hugging Face. Their groundbreaking contributions and innovative approaches have significantly shaped the current landscape of AI research and application.

4.1 OpenAI: Advancing Artificial General Intelligence

OpenAI [10], founded in December 2015, stands as a pioneering force in the realm of artificial intelligence research. Their profound mission revolves around ensuring that artificial general intelligence (AGI) - which represents highly autonomous systems surpassing human performance in economically valuable tasks - serves the betterment of all humanity. OpenAI's dedication extends to constructing AGI that is both safe and beneficial or assisting others in this noble pursuit. The overarching commitment of OpenAI is to steer AGI towards beneficial applications while averting harmful uses.

4.1.1 Key Achievements and Models

OpenAI's contributions to the field of AI have been groundbreaking, and their models exemplify the cutting edge of research and development. Here's an overview of some of their most remarkable models:

- **GPT-3 (Generative Pre-trained Transformer 3):** Among their crowning achievements, GPT-3 is an exemplar of excellence. With a staggering 175 billion parameters, it holds the distinction of being one of the largest and most potent language models in existence. GPT-3 has exhibited exceptional capabilities in natural language understanding, generation, and text-based tasks, marking a significant milestone in AI research.

- **GPT-4:** OpenAI's GPT-4 represents their latest pinnacle of advancement. It is a large multimodal model, capable of processing both image and text inputs while delivering text outputs. Although it may fall short of human-level performance in various real-world scenarios, it shines in professional and academic benchmarks, setting new standards in AI capabilities.

- **DALL·E 3:** DALL·E 3 is a testament to OpenAI's commitment to detail and nuance. This model excels at translating conceptual ideas into remarkably accurate images, elevating the possibilities of AI-driven creative expression.

- **GPT-3.5 Turbo:** Despite the availability of the GPT-4 model, the GPT-3.5-Turbo Model remains a powerful and cost-effective option. It powers the widely popular ChatGPT and offers users the potential to create their chatbot with similar capabilities.

A noteworthy feature of the GPT-3.5-Turbo model is its multi-turn capability, enabling it to process a sequence of messages as input. This marks a substantial improvement over the GPT-3 model, which only accommodated single-turn text prompts. This enhancement allows users to employ preset scenarios and previous responses as contextual information, significantly enhancing the quality of generated responses. These features will be explored in greater detail in the subsequent section.

Ethical AI Advocacy: OpenAI has consistently been at the forefront of advocating for ethical and responsible AI usage. Their unwavering commitment to establishing guidelines and principles for the ethical use and safety of AI has played a pivotal role in influencing the industry's approach to responsible AI development and application.

4.1.2 Embedding

An embedding is a numerical representation of a piece of information, for example, text, documents, images, audio, etc. The representation captures the semantic meaning of what is being embedded, making it robust for many industry applications.

Given the text *"What is the main benefit of voting?"*, an embedding of the sentence could be represented in a vector space, for example, with a list of 384 numbers (for example, [0.84, 0.42, ..., 0.02]). Since this list captures the meaning, we can do exciting things, like calculating the distance between different embeddings to determine how well the meaning of two sentences matches.

Embeddings are not limited to text! You can also create an embedding of an image (for example, a list of 384 numbers) and compare it with a text embedding to determine if a sentence describes the image. This concept is under powerful systems for image search, classification, description, and more!

4.1.3 OpenAI pricing perspective

Now, let's consider the pricing perspective. The following table 4.1 shows the usage cost of OpenAI Embedding Models for every 1,000 tokens:

Model	Usage
ada v2	$0.0001 / 1K tokens

Table 4.1: OpenAI Embedding Model and Usage Cost [11].

To gain a comprehensive understanding of the pricing structure of GPT-3.5 and GPT-4, let's delve into the costs associated with different OpenAI models. The table 4.2 and 4.3 provides a breakdown of the usage expenses for each model, based on every 1,000 tokens, both for input and output:

Model	Input Cost	Output Cost
gpt-3.5-turbo	$0.0020 / 1K tokens	$0.0020 / 1K tokens
gpt-3.5-turbo-1106	$0.0010 / 1K tokens	$0.0020 / 1K tokens
gpt-3.5-turbo-instruct	$0.0015 / 1K tokens	$0.0020 / 1K tokens

Table 4.2: GPT-3.5 Models Input and Output Costs.[11]

Model	Input Cost	Output Cost
gpt-4	$0.03 / 1K tokens	$0.06 / 1K tokens
gpt-4-32k	$0.06 / 1K tokens	$0.12 / 1K tokens

Table 4.3: GPT-4 Models Input and Output Costs [11] .

4.1.4 OpenAI Pricing calculation

To calculate the pricing for OpenAI's services used in the project, we define the following parameters:

$p =$ Number of sessions per month
$q =$ Average number of tokens per conversation
$x =$ Approximate number of Tokens used for embedding
$r =$ Cost per token during Input and output
$y =$ Cost per token during embedding

With these parameters, we can determine the costs associated with embedding and usage of the GPT model as follows:

$$\text{Cost of Embedding} = x \cdot y \tag{4.1}$$

$$\text{Cost of GPT Model Usage} = p \cdot q \cdot r \tag{4.2}$$

Total monthly licensing cost to OpenAI

$$\text{Total cost} = \text{Cost of embedding} + \text{Cost of GPT model} \tag{4.3}$$

Equations 4.1 and 4.2 will be utilized to assess the financial implications of deploying OpenAI's models in our application.

Example:

Values such as the number of sessions per month and the average number of tokens per conversation have been provided by ECT's marketing team.

Let us consider the following values for the calculation:

$p = 900$ (Number of sessions per month)
$q = 213$ (Average number of tokens per conversation)
$x = 100{,}000$ (Approximate number of tokens used for embedding a 50-page book)

Pricing for embedding and model usage are referenced from Tables 4.1 and 4.2. For this example, the use of GPT-3.5-TURBO and ada-v2 models is assumed.

$r = \$0.000002$ (Cost per token during input and output)
$y = \$0.0000001$ (Cost per token during embedding)

Cost Calculation:

The cost of embedding and GPT model usage are calculated as follows:

Cost of Embedding

$$\begin{aligned} \text{Cost of embedding} &= x \cdot y && (4.4)\\ &= 100,000 \cdot 0.0000001 && (4.5)\\ &= \$0.01 && (4.6) \end{aligned}$$

Cost of GPT Model

$$\begin{aligned} \text{Cost of GPT model} &= (p \cdot q) \cdot r && (4.7)\\ &= (900 \cdot 213) \cdot 0.000002 && (4.8)\\ &= \$0.3834 && (4.9) \end{aligned}$$

Total Cost

$$\begin{aligned} \text{Total cost} &= \text{Cost of embedding} + \text{Cost of GPT model} && (4.10)\\ &= \$0.01 + \$0.3834 && (4.11)\\ &= \$0.3934 && (4.12) \end{aligned}$$

Approximate number of sessions that are done in a month (industry standard)	900
Average number of tokens per conversation	213
Total token used per month	191700
Tokens used for training (assuming a 25 page word file is used for training)	100000
Total cost of GPT model	$0.3834
Total cost of embedding model	$0.01
Total monthly licensing cost to OpenAI	$0.3935

Figure 4.1: OpenAI Pricing calculation.

Results:

The total monthly cost for licensing OpenAI services, based on the given parameters, amounts to $0.3934.

4.2 Hugging Face: Pioneering in NLP and AI Community

Hugging Face [12] is a trailblazing entity in the realm of artificial intelligence (AI), earning widespread acclaim for its monumental contributions to natural language processing (NLP). Established in 2016 by visionaries Clément Delangue and Julien Chaumond, Hugging Face initially embarked on its journey as a chatbot-centric venture. However, discerning the expansive potential in NLP, the company soon realigned its focus towards mastering this domain.

4.2.1 The Transformers Library: A Revolution in NLP

Central to Hugging Face's acclaim is its *Transformers* library, a toolkit that has fundamentally reshaped the NLP landscape within the AI community. This comprehensive library houses an extensive array of pre-trained models, each adept at handling a spectrum of NLP tasks. These range from text classification and information extraction to question answering, encapsulating diverse NLP functionalities. Notable among these models are BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), each pioneering in its own right.

4.2.2 Fostering Community and Collaboration

Hugging Face's ethos is deeply rooted in community engagement and collaborative progress. The company has established a vibrant platform where the global community actively contributes to its model hub. This open invitation to share research, insights, and work on collective projects has led to an expansive repository of models, datasets, and tools, further enriching the field of NLP.

4.2.3 Innovative Strides in Machine Learning

Staying abreast of the ever-evolving landscape, Hugging Face consistently integrates the latest breakthroughs in NLP research into its platform. This unwavering dedication to innovation cements the company's status as a premier resource for cutting-edge NLP technologies.

4.2.4 Sentence-Transformers: Enhancing Textual Understanding

A notable offering in Hugging Face's arsenal is the *sentence-transformers* library. This tool specializes in computing embeddings for sentences, paragraphs, and even images. It operates in a vector space where texts with similar meanings are positioned in proximity, enabling a range of applications including semantic search, clustering, and efficient information retrieval.

4.2.4.1 Exploring Sentence-Transformers in the Model Hub

The model hub on Hugging Face's platform features an impressive collection of over 500 sentence-transformer models. These models are adept at various tasks, including feature extraction for embedding generation and sentence similarity assessments. The official documentation offers a comprehensive overview of these pre-trained models.

4.2.4.2 Features of Models on the Hub

Each model on the Hugging Face Hub is accompanied by:

- An auto-generated model card detailing descriptions, code snippets, architecture overviews, and more.
- Metadata tags for enhanced discoverability, containing crucial information like licensing.
- An interactive widget allowing users to experiment with the model directly in a browser.
- An Inference API for executing inference requests.

In conclusion, while both OpenAI and Hugging Face have profoundly impacted the field of AI and NLP, OpenAI seems to have taken a step ahead in specific technical aspects such as sentence structuring, processing speed, and embedding processes. This, however, does not diminish the significance of Hugging Face's contributions to the AI community. The future of AI and NLP is likely to be shaped by the continued evolution and collaborative efforts of these pioneering organizations, each pushing the boundaries of what's possible in artificial intelligence.

5 LangChain: A Framework for Language Model-Powered Applications

LangChain is a comprehensive framework designed to develop applications that are empowered by language models. It represents a cutting-edge platform that facilitates interaction with data through conversational AI and natural language processing. LangChain enables users to engage in dynamic dialogues with their data, allowing for seamless access, analysis, and extraction of insights. The platform is versatile, supporting a wide range of industries and document loaders, with a particular proficiency in handling PDF files. Moreover, LangChain is committed to data privacy and security, ensuring that user information is managed with the utmost care and integrity.

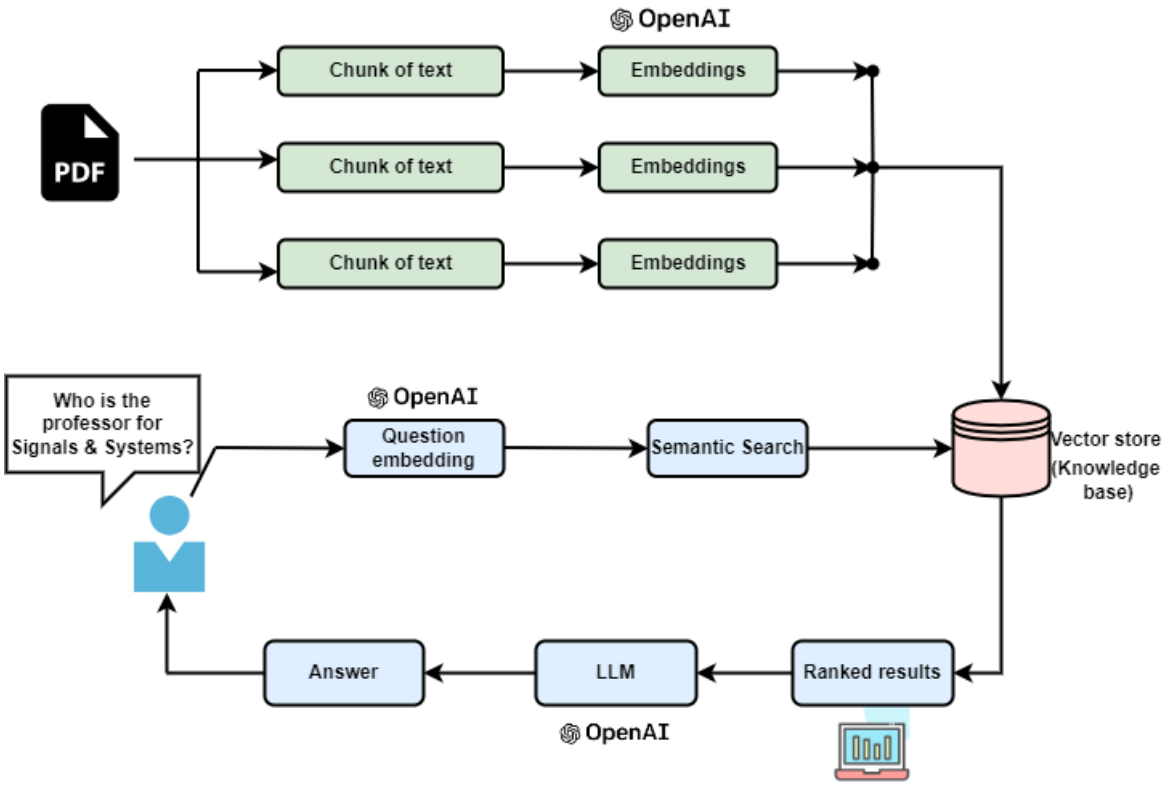

Figure 5.1: Langchain Architecture [13].

In the realm of Large Language Models (LLMs), chatbots emerge as a prime application, with their ability to conduct long-running conversations and provide access to sought-after information. Beyond basic prompting and LLM functionalities, memory and retrieval stand out as pivotal components of a chatbot's architecture. Memory enables chatbots to recall past interactions, thus providing a personalized user experience. Retrieval, on the other hand, arms chatbots with the ability to access up-to-date, domain-specific information, which is essential for delivering accurate and relevant responses.

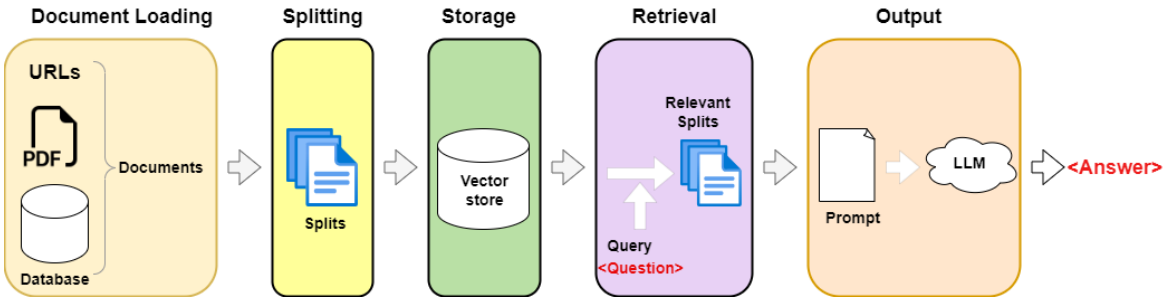

Figure 5.2: Langchain step by step procedure [14][15].

5.1 Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) represents a paradigm in natural language processing that enhances the capabilities of Large Language Models (LLMs) by integrating them with an external corpus of contextual documents. This process enriches the model's execution with a broader repository of information to draw from, leading to more informed and accurate outputs.

Also known as mixed-generation or hybrid generation, RAG is a technique that synergizes retrieval-based and generation-based methods in natural language processing. Retrieval-based methods involve sourcing pre-established responses or data segments from a curated database or knowledge base, with the selection hinged on the pertinence to the input prompt. Conversely, generation-based methods create responses anew, employing machine learning models such as sophisticated language models like ChatGPT.

In a RAG system, an initial attempt is made to fetch pertinent information or responses from a stored knowledge base. Should an apt response be located, it is then furnished directly to the user. In instances where the retrieval process does not yield a satisfactory answer, the system pivots to a generative model to construct a fresh response.

This hybrid approach marries the precision and specificity of retrieval-based methods—capable of presenting accurate data when it exists within the knowledge base—with the adaptive and inventive capabilities of generative methods, which are adept at tackling unprecedented inquiries or fabricating more varied and imaginative replies.

In essence, the objective of retrieval-augmented generation is to bolster the caliber and variety of dialogues in conversational AI interfaces by intelligently amalgamating the strengths of both retrieval and generative strategies.

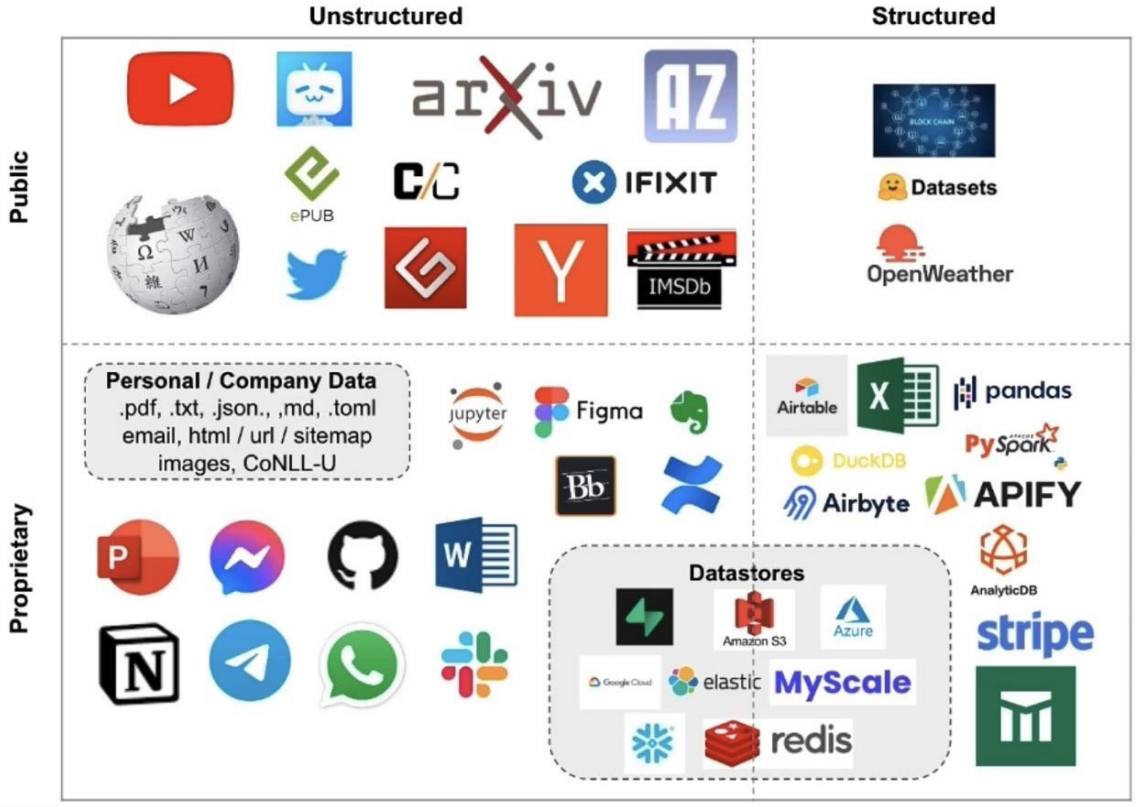

Figure 5.3: Langchain supported file-types. [14][15]

5.1.1 Document loader

Within the array of document loaders available, approximately 80 distinct types are noted, please refer to the Figure 5.3. For the purposes of this internship, our focus will be primarily on PDF files.

PyPDF is a versatile Python library designed for PDF interactions. The following code snippet illustrates how to utilize the PyPDFLoader from the LangChain document loaders to work with a PDF file:

Listing 5.1: Loading a PDF file using PyPDFLoader [14]

```
from langchain.document_loaders import PyPDFLoader

loader = PyPDFLoader("Vorlesung_Signal_und_Systemtheorie_Slides.pdf")
    pages = loader.load()
```

The task involves the PDF file entitled *Vorlesung Signal und Systemtheorie Slides*. The first action entails loading the document into the program. Subsequent steps, including document splitting, will be discussed in the next section 5.1.2.

5.1.2 Document Splitting

Splitting documents into smaller, more manageable chunks is a crucial step before sending data to the vector store. This process enhances readability and facilitates efficient data processing.

LangChain provides various types of splitters under the module 'langchain.textsplitter':

- **CharacterTextSplitter():** Implementation of splitting text that looks at characters.
- **MarkdownHeaderTextSplitter():** Implementation of splitting markdown files based on specific headers.
- **TokenTextSplitter():** Implementation of splitting text that looks at tokens.
- **SentenceTransformersTokenTextSplitter():** Implementation of splitting text that looks at tokens.
- **RecursiveCharacterTextSplitter():** Implementation of splitting text that looks at charaters. Recursively tries to osplit by dufferent charaters to find one that works.
- **Language():** for CPP, Python, Puby, Markdown etc.
- **NLTKTextSplitter():** Implementation of splitting text that looks at sentences using NLTK (Natural Language Tool Kit)
- **SpacyTextSplitter():** Implementation of splitting text that looks at sentences using Spacy.

In our project, we utilize the 'RecursiveCharacterTextSplitter', with a chunk size of 1536 characters and an overlap of 200 characters. The data is subsequently split into 16 different chunks, each comprising 1536 characters.

The 'RecursiveCharacterTextSplitter' is particularly adept at dividing text based on character count. It employs a specified separator and determines chunk size by counting characters. Below is an example demonstrating the usage of 'RecursiveCharacterTextSplitter'. This approach divides the text into chunks approximately 1536 characters in length, with an overlap of 200 characters. The resulting chunks are stored in the 'docs' variable.

Listing 5.2: Using RecursiveCharacterTextSplitter [14]

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1536,
    chunk_overlap=200,
    separators=["\n\n", "\n", "(?<=\. )", " ", ""])
docs = text_splitter.split_documents(pages)
```

5.1.3 Embeddings and Vector stores

Having segmented our document into smaller, semantically rich fragments, the next step involves indexing these segments. This facilitates their easy retrieval for answering queries related to this corpus of data. To achieve this, we employ embeddings and vector stores.

Embeddings:

Embeddings offer a numerical representation of text data, encapsulating the semantic essence and interrelations between words. Consequently, texts with similar content are represented by closely resembling vectors. In our project, we utilize the OpenAI embedding model.

Listing 5.3: Using OpenAIEmbeddings [14]

```
from langchain.embeddings.openai import OpenAIEmbeddings
embedding = OpenAIEmbeddings()
```

Vector Stores/Databases:

Vector stores, or databases, are specialized data structures tailored for the efficient storage and retrieval of vector representations of data. In the domain of natural language processing (NLP), these stores are instrumental in housing embeddings, which are numerical representations of textual data.

For our purposes, we have chosen to use Qdrant:

Listing 5.4: Setting up Qdrant Vector Store [14]

```
from langchain.vectorstores import Qdrant
from qdrant_client import models

self.client.create_collection(
    collection_name = chatbot_name,
    vectors_config = models.VectorParams(
        size = 1536,
        distance = models.Distance.COSINE),
)

vector_store = Qdrant(
    client = self.client,
    collection_name = chatbot_name,
    embeddings = embeddings,
)
vector_store.add_documents(text_chunks)
```

In this configuration, the Qdrant vector store is set up to accommodate the specific requirements of our chatbot application, ensuring efficient management and retrieval of the embedded data.

5.1.4 Retrieval

The retrieval process is a crucial phase in information retrieval systems, focusing on locating relevant documents or information in response to a user's query. This process leverages vector stores or databases, which house embeddings of textual data, to efficiently find pertinent content.

Within LangChain, the retrieval mechanism is facilitated by the RetrievalChain class. This class is an integral part of the LangChain library and employs vector stores for effective retrieval operations. The following illustrates how retrieval is executed using LangChain:

Accessing/Indexing Data in Vector Store:

The retrieval process can encompass various methods, including:

- Basic similarity search
- Maximum marginal relevance (MMR)
- Including metadata in search

Here, we will delve into Basic similarity search and Maximum Marginal Relevance (MMR).

Basic Similarity Search:

Basic similarity search is an essential function in both information retrieval and natural language processing. It involves identifying documents or elements that bear a close resemblance or relation to a specified query. The process seeks to find items that mirror the essence of the query in question.

Listing 5.5: Basic Similarity Search Example [14]

```
question = "Who is the professor for Signals & Systems?"
answer = db.similarity_search(question, k=2)
```

Here, 'k=2' implies that the search will select the two documents most similar to the query.

Maximum Marginal Relevance (MMR):

MMR is a strategy in information retrieval aimed at enhancing the diversity of search outcomes. It seeks a balance between relevance and diversity, ensuring the results are not just highly pertinent but also encompass a broad spectrum of topics.

Sometimes, the objective is not to choose the most similar responses:

Listing 5.6: Maximum Marginal Relevance Search Example [14]

```
question = "Who is the professor for Signals & Systems?"
answer = vectordb.max_marginal_relevance_search(question, k=2)
```

In this case, ‘k=2‘ indicates that the search will select the two most diverse documents relative to the query.

5.1.5 Question Answering

RetrievalQA, or Question Answering, is a vital task in natural language processing (NLP) that focuses on deriving precise answers to user queries from a specified context or a collection of documents. The essence of RetrievalQA lies in its ability to amalgamate information retrieval and language understanding techniques to furnish accurate and contextually relevant answers.

The implementation of RetrievalQA in our project is demonstrated below, showcasing how this process is facilitated using the LangChain framework:

Listing 5.7: Implementing RetrievalQA [14]

```
from langchain.chains import RetrievalQA
from langchain.llms import ChatOpenAI
from langchain.vectorstores import vectordb

qa_chain = RetrievalQA(
    llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0),
    retriever = vectordb.as_retriever(),
    return_source_documents = True,
    chain_type = "refine"
)
result = qa_chain({"question": ""})
result['answer']
```

Listing 5.8: RetrievalQA Output [14]

```
Answer: The professor for Signals and Systems is Prof. Dr.-Ing. Alexander Lampe.
```

This example illustrates the RetrievalQA chain’s capability to efficiently process a user’s question and retrieve the most relevant and accurate answer. The integration of a large language model (LLM), in this case, the GPT-3.5 Turbo model, with a retriever component (vectordb), exemplifies the powerful synergy between language models and vector-based retrieval mechanisms in modern NLP applications.

6 Methodology

6.1 Business Understanding

Develop an AI chatbot where the owner uploads the information in the form PDF file to the portal and the new chatbot is created. When the user asks questions the information is fetched from the PDF.

- **Project Overview:** The primary objective of this internship project is to design and implement an AI-powered chatbot system that enhances user interaction and information retrieval. The central functionality of the chatbot involves the seamless integration of PDF files, where the owner can upload relevant information to a designated portal. The chatbot, in turn, is engineered to extract and present pertinent information from these PDF documents in response to user queries.

- **User and Owner Roles:** In this collaborative environment, distinct roles have been identified for key stakeholders:
  Owner: The owner of the system plays a pivotal role in contributing information to the chatbot. They are responsible for uploading PDF files containing valuable data related to user queries.
  Users: End-users engage with the chatbot by posing questions or seeking information. The chatbot's function is to intelligently comprehend these queries and retrieve accurate responses from the uploaded PDF documents.

- **PDF File Integration:** The integration of PDF files into the chatbot framework is a critical aspect of this project. Owners can easily upload PDF documents through a user-friendly portal, allowing the chatbot to access and extract relevant information. This feature is designed to streamline the process of updating and managing information within the system.

- **Business Impact:** The successful implementation of the AI chatbot with PDF file integration aligns with the broader business objectives of enhancing user experience and operational efficiency. The chatbot's ability to swiftly retrieve information from uploaded PDFs is anticipated to result in improved user satisfaction, increased productivity, and a more streamlined information retrieval process for the owner.

6.2 Architectural diagram for implementation in Project

6.2.1 Low-code application

Low-code applications are software applications that can be developed with minimal or no coding required. This is achieved through the use of visual development tools, drag-and-drop functionality, and pre-built components. Low-code applications are becoming increasingly popular as they offer several advantages over traditional development methods

Benefits of low-code applications:

- **Reduced development time and cost:** Low-code applications can be developed much faster and more cost-effectively than traditional applications. This is because low-code platforms abstract away the complexity of coding, making it easier for developers to get started and produce results quickly.

- **Increased productivity:** Low-code platforms can help developers to be more productive by providing them with several features that automate repetitive tasks. This can free up developers to focus on more complex and strategic aspects of the development process.

- **Improved accessibility:** Low-code platforms make it possible for people with limited coding experience to develop applications. This can empower non-technical users to create custom solutions that meet their specific needs.

- **Enhanced agility:** Low-code platforms make it easier for businesses to be agile and responsive to changing demands. This is because low-code applications can be developed and deployed quickly and easily.

Low-code applications are a powerful tool that can help businesses to develop custom applications quickly and cost-effectively. They are particularly well-suited for businesses that want to empower non-technical users to develop solutions that meet their specific needs.

As the low-code market continues to grow, we can expect to see even more innovative and powerful low-code platforms emerge. This will make it possible for businesses to develop even more sophisticated and custom applications without having to rely on expensive and experienced developers.

6.2.2 Architecture diagram

- **LCAP application:** The Low Code Application (LCAP) acts as the user's gateway to interacting with the AI chatbot system. It is meticulously designed to ensure a user-friendly and intuitive experience, specifically focusing on the question-and-answer functionalities. Here are the key elements related to the UI:

  - **User Actions:** LCAP provides a clean and intuitive interface for users to perform various actions seamlessly. The UI incorporates elements for uploading PDF files, initiating queries, and interacting with the chatbot. The user actions are streamlined to enhance usability and ensure that users can easily navigate through the application.

  - **PDF File Upload:** A dedicated section within the LCAP UI is allocated for PDF file uploads named **File Upload**. Users can effortlessly upload relevant documents containing information they want the chatbot to process. The upload process is

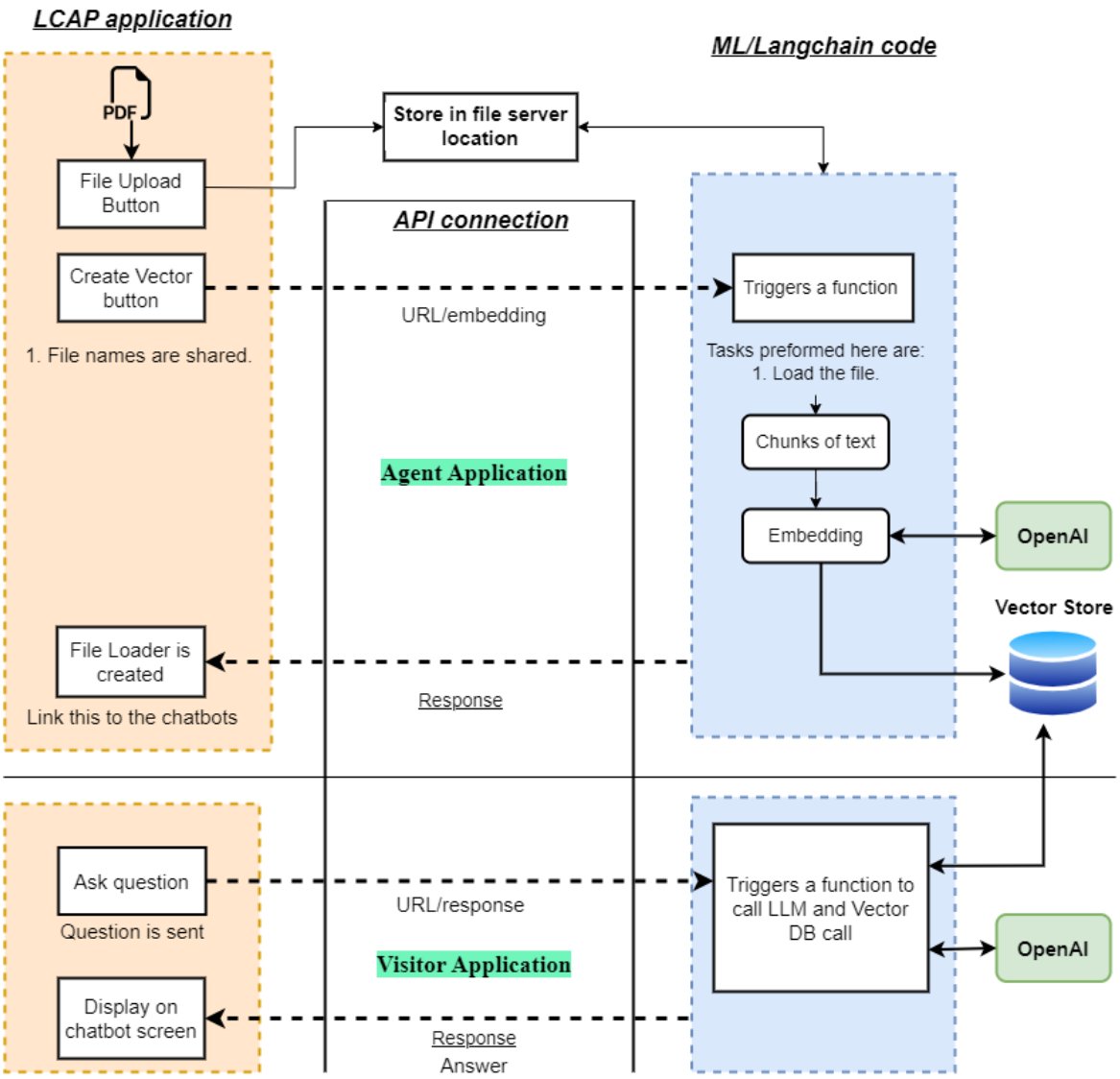

Figure 6.1: Architecture diagram

designed to be straightforward, allowing users to select and submit PDF files with minimal effort.

- **Question and Answer Interface:** The heart of the LCAP UI is the question-and-answer interface. Users can pose queries to the chatbot through a user-friendly input mechanism. The UI is optimized for natural language input, enabling users to communicate with the chatbot as if interacting with a human. Additionally, the interface responds in a clear and easily understandable format, fostering a conversational and engaging experience.

In summary, the LCAP Application not only serves as a functional bridge between users and the underlying components but is also carefully crafted to provide an exceptional user experience. The emphasis on an intuitive UI and seamless actions contributes to a positive and efficient interaction with the AI chatbot, making the process of querying and receiving answers a user-friendly and engaging experience.

- **API connection:** To establish communication between the LCAP application and the ML/Langchain code, a REST API approach is adopted. This ensures a standardized and interoperable connection between the front-end and back-end components. Fastapi, a modern, fast web framework for building APIs with Python 3.7+ based on standard Python-type hints, is employed to facilitate the creation and maintenance of the API.

- **ML/Langchain code:** The ML/Langchain code section is the core engine responsible for various actions, including Document Loader, Document Splitting, Embedding, Vector Stores, Retrieval, and Question An swering. This section utilizes the following technologies:
  - **Programming language:** Python is chosen as the primary programming language due to its versatility and widespread adoption in the fields of machine learning and natural language processing. Python's extensive ecosystem of libraries and frameworks, such as TensorFlow and PyTorch, facilitates the implementation of advanced machine learning algorithms and models.

  - **Database: Qdrant** Qdrant is employed as the database solution for efficient storage and retrieval of vectorized information. Qdrant is particularly well-suited for similarity search applications, making it an optimal choice for our information retrieval tasks. Its ability to handle high-dimensional vector data aligns with the demands of document embedding and retrieval within the chatbot system.

  - **Model: The GPT-3.5-Turbo** The GPT-3.5-Turbo model, developed by OpenAI, serves as the powerhouse for advanced natural language understanding and generation. This state-of-the-art language model is capable of comprehending nuanced queries and generating contextually relevant responses. Leveraging the power of GPT-3.5-Turbo enhances the chatbot's ability to engage in meaningful and context-aware conversations with users.

  - **API connection: Fastapi** Fastapi is employed to establish a robust connection between the ML/Langchain Code and the API layer. Fastapi is a modern web framework for building APIs with Python, and it was chosen for its speed, simplicity, and automatic generation of interactive documentation. The API layer facilitates smooth communication and data exchange between the user interface (LCAP Application) and the core ML/Langchain Code, ensuring a seamless end-to-end experience.

  - **Hosting: Kubernetes and Docker [16]** For efficient containerization and orchestration of the ML/Langchain Code, Kubernetes and Docker are employed. Containerization ensures that the code and its dependencies are encapsulated in isolated containers, providing consistency across different environments. Kubernetes, as the orchestration tool, manages the deployment, scaling, and operation of these containers. This combination of technologies enables a scalable and efficient hosting environment, ensuring consistent performance and facilitating easy management of system resources.

In essence, the ML/Langchain Code section is a sophisticated amalgamation of Python-based programming, Qdrant database for efficient data storage, the powerful GPT-3.5-Turbo model for advanced natural language processing, Fastapi for seamless API connections, and Kubernetes/Docker for scalable and efficient hosting. Together, these components empower the chatbot system to process documents, perform complex language tasks, and deliver accurate responses, contributing to a robust and intelligent conversational AI experience.

This architecture diagram illustrates the synergy between the LCAP application, API connection, and the ML/Langchain code, ensuring a cohesive and responsive AI chatbot system capable of extracting valuable information from PDF files and delivering accurate responses to user queries. The utilization of cutting-edge technologies enhances the system's capabilities, offering a sophisticated solution for document-based information retrieval.

7 Developing and Demonstrating the Chatbot Experience

7.1 Implementation Process

Agent Side Interface Development:

This section outlines the step-by-step procedure for creating a chatbot interface that allows for document upload and manipulation.

Step 1: Initiate Document Upload
Begin by locating and selecting the file upload button on the chatbot interface. This action is illustrated in Figure 7.1.

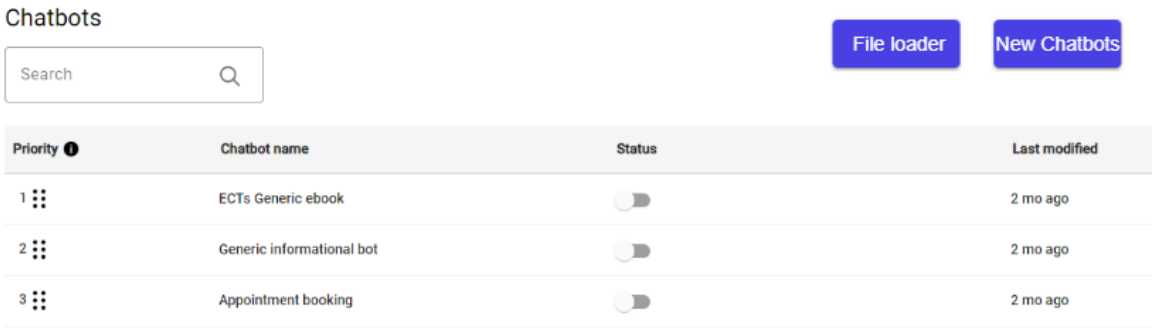

Figure 7.1: Initial chatbot interface with document upload option.

Step 2: Add and Manage Documents
Proceed by clicking the **Add steps** button to create a new document loader. You may rename the file for better identification. To add files, select the **browse button**, as depicted in Figure 7.2.

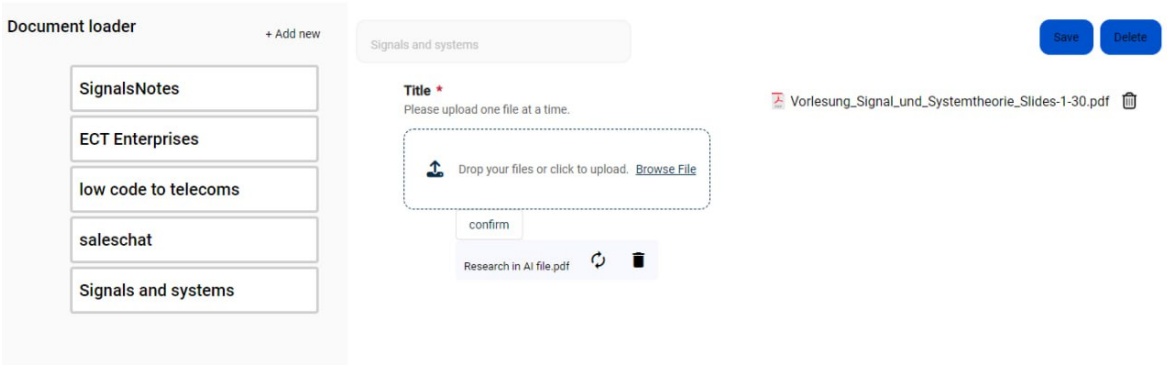

Figure 7.2: Adding a new document loader.

Step 3: Upload Multiple Documents

Multiple files can be added at this stage if necessary. The process for adding multiple files on clicking browse button and how these myltiple files can be listed is shown in Figure 7.3.

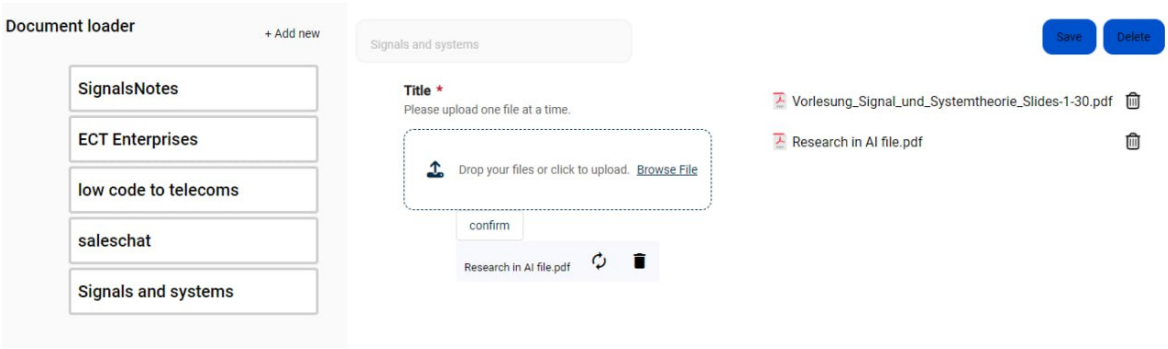

Figure 7.3: Interface for adding multiple files.

Step 4: Save and Process Documents

After adding the desired files, finalize the process by clicking the **Save button**. This converts the file data into vector format and stores it in the Qdrant database, as demonstrated in Figure 7.4.

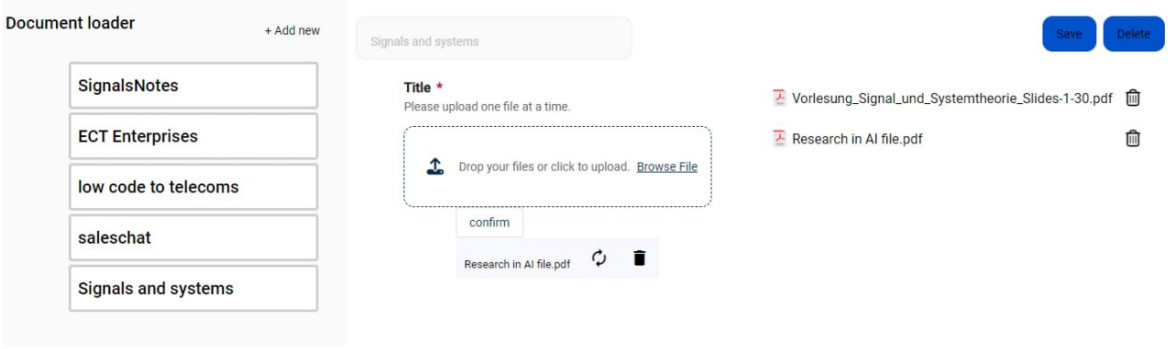

Figure 7.4: Saving the documents to the database.

Step 5: Verify Database Creation

Post-save, a database named "SignalsNotes" or "Signals and systems" appears in Qdrant. The creation of this database is shown in Figure 7.5.

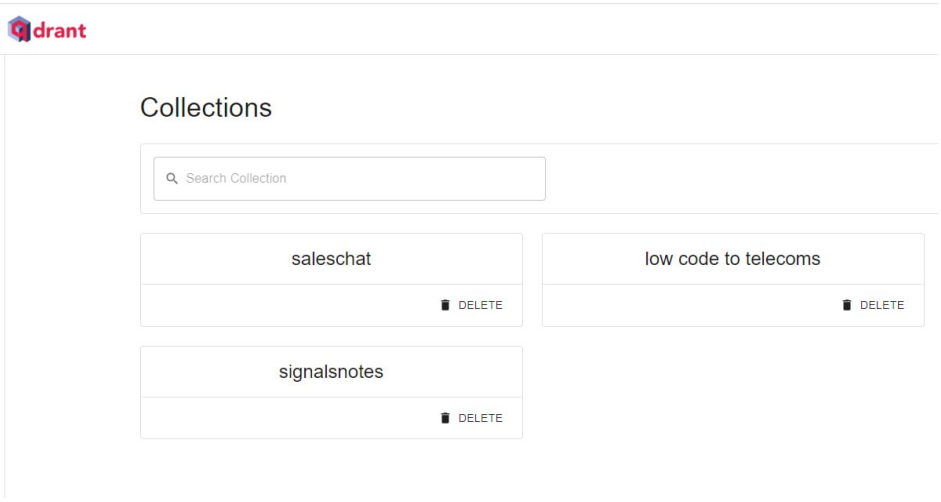

Figure 7.5: Qdrant database with the new entry.

Step 6: Inspect Vector Storage
Finally, verify the data and vector storage within the database. This verification is visualized in Figure 7.6.

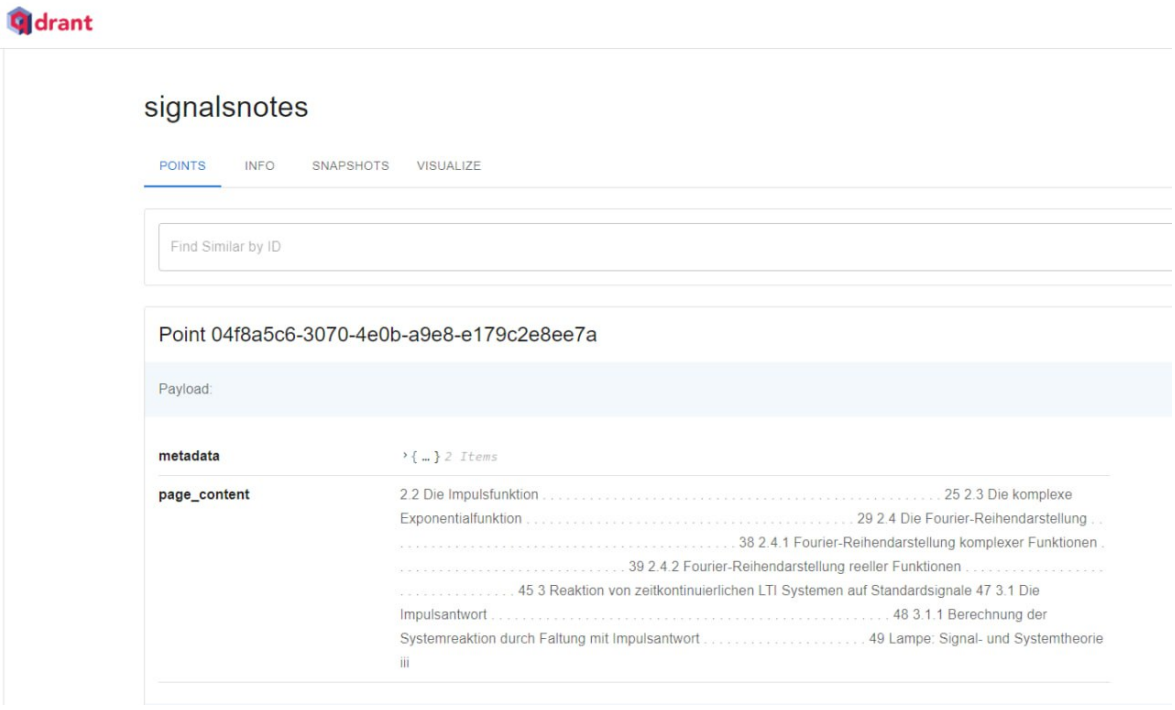

Figure 7.6: Inspecting the vector store in Qdrant.

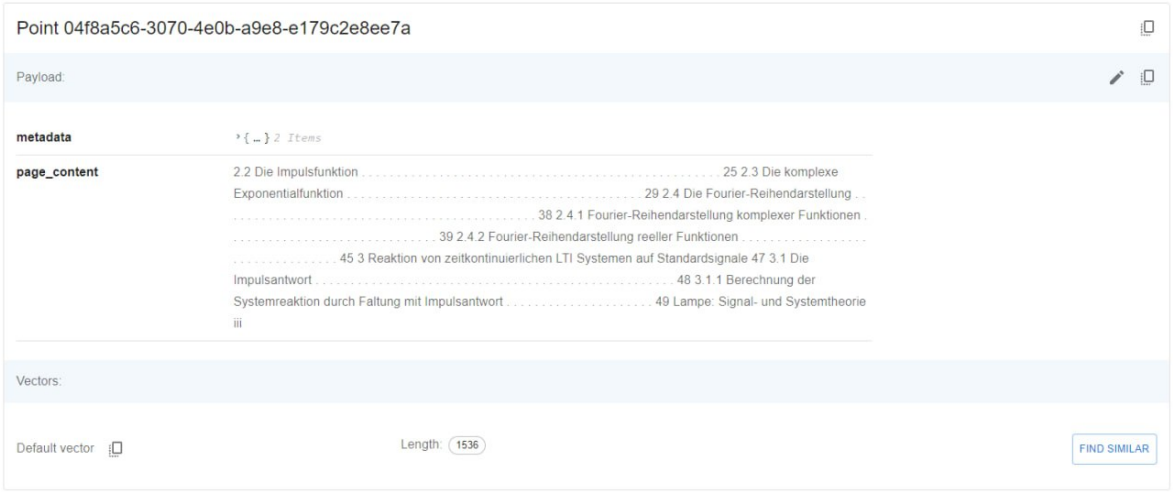

Figure 7.7: Inspecting the points in Qdrant.

Following these steps, document loaders are created to facilitate the inclusion of various document types into the chatbot system, enhancing the bot's functionality.

7.2 Visitor Side: Interaction with Emma the Chatbot

The visitor-side application facilitates interaction with a virtual assistant, Emma, designed to navigate through various topics. This section walks through the dialogue process and demonstrates how visitors can engage in conversations with Emma, particularly on subjects such as Signals and Systems, Cryptography, Modern Analysis, and Live Support.

7.2.1 Engaging with the Chatbot

Step 1: Introduction to Emma

Upon initiating a chat, visitors are greeted with an introductory message from Emma. The interface, showcasing the initial interaction, is presented below in Figure 7.8.

Visitors are encouraged to familiarize themselves with the chatbot flow and respond accordingly to establish a conversation.

Step 2: Navigating Through Topics

Emma provides a selection of topics from which visitors can choose. For instance, if a visitor selects "Signals and Systems," Emma is prepared to engage in a detailed discussion pertinent to that subject. The interface offering topic choices through a "Quick Reply" feature is depicted in Figure 7.8.

7.2.2 Initiating Subject-Specific Dialogue

Once a topic is selected, visitors have the opportunity to delve deeper, posing questions to learn more about the chosen subject. An illustrative dialogue might involve a query regarding the instructor of the "Signals and Systems" course.

> *"Who is the professor for Signals and Systems?"*

In response, Emma, the chatbot, furnishes pertinent information, facilitating an engaging and informative interaction, as demonstrated in Figures 7.9 and 7.10. These answer is fetched from the Qdrantdb based on the pdf file we uploaded.

Expanding the interaction, inquiries can be tailored to request responses in different languages, such as German, enhancing the chatbot's versatility. Figure 7.11 exemplifies such a multilingual request.

Further exploration might include asking about the tutoring staff for the subject, as depicted in Figure 7.12.

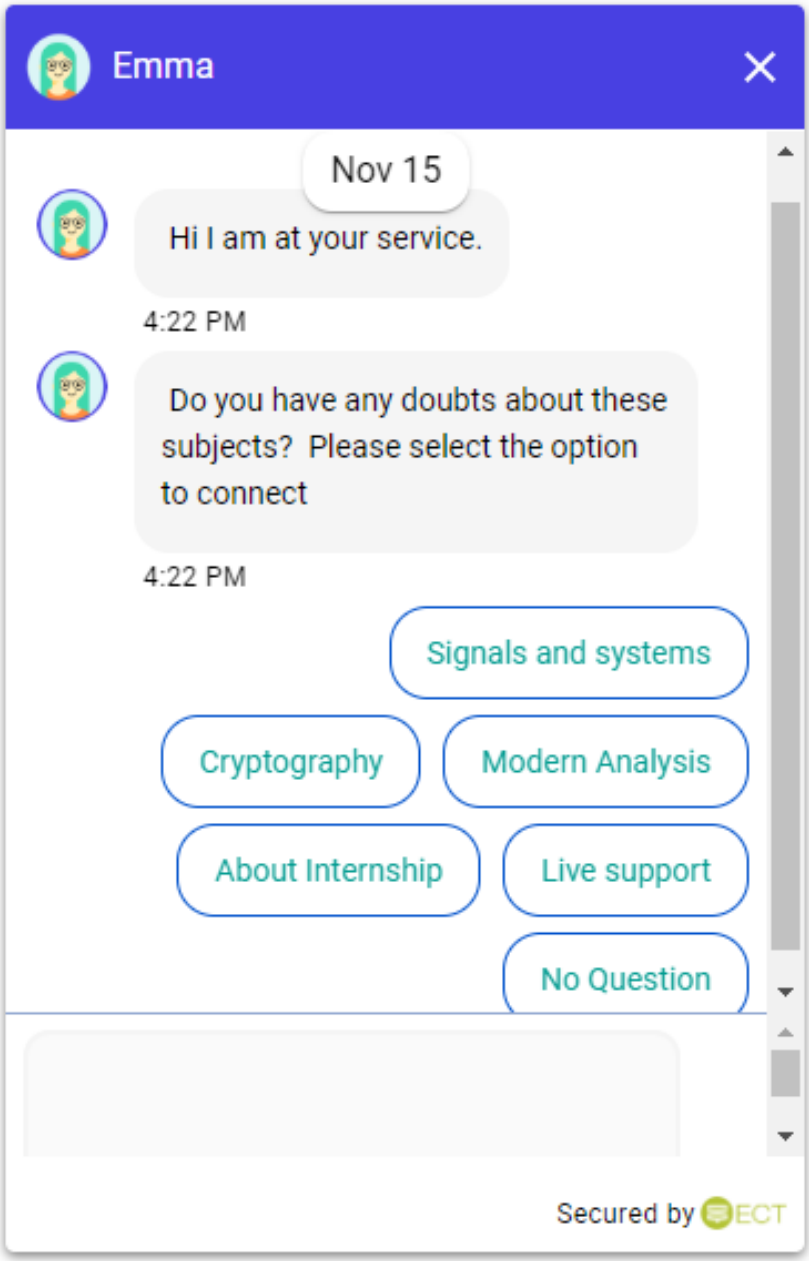

Figure 7.8: The initial interaction page with Emma, the chatbot.

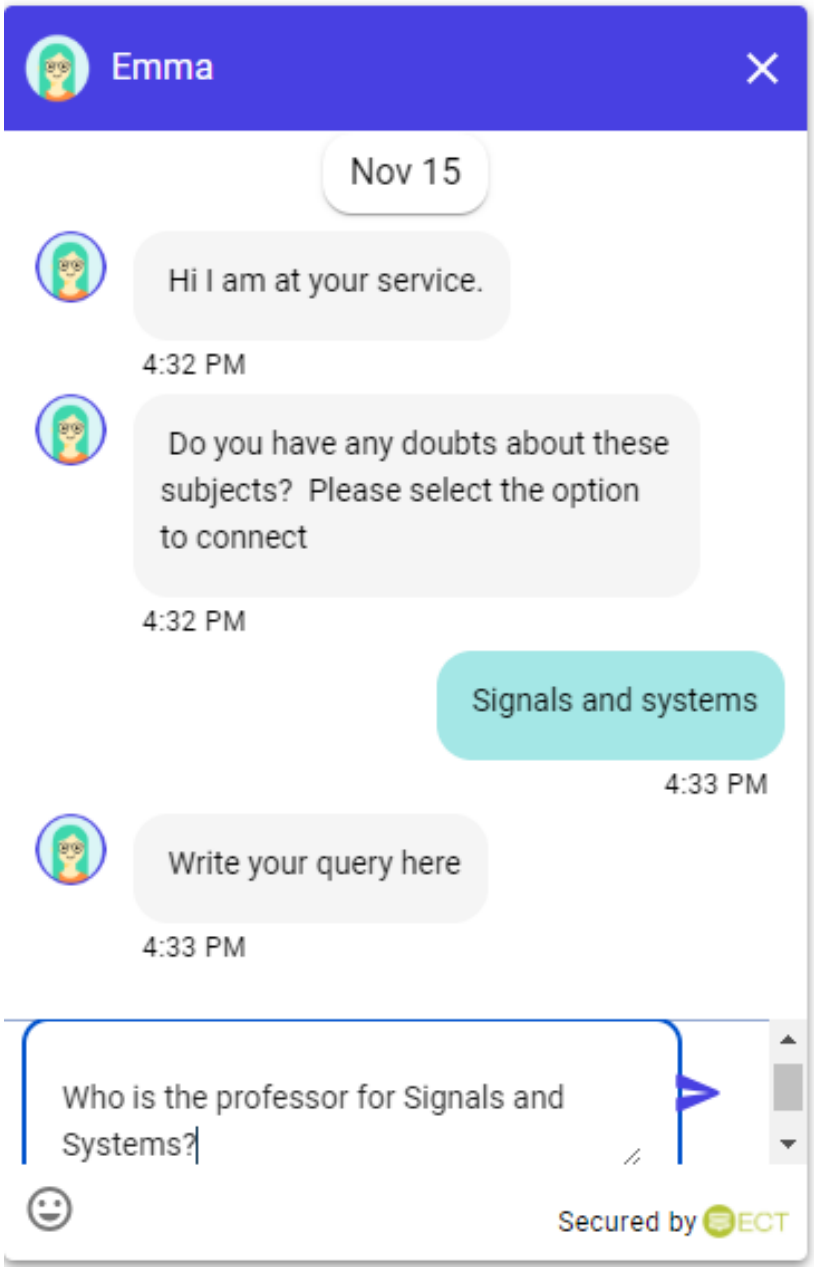

Figure 7.9: Question 1.

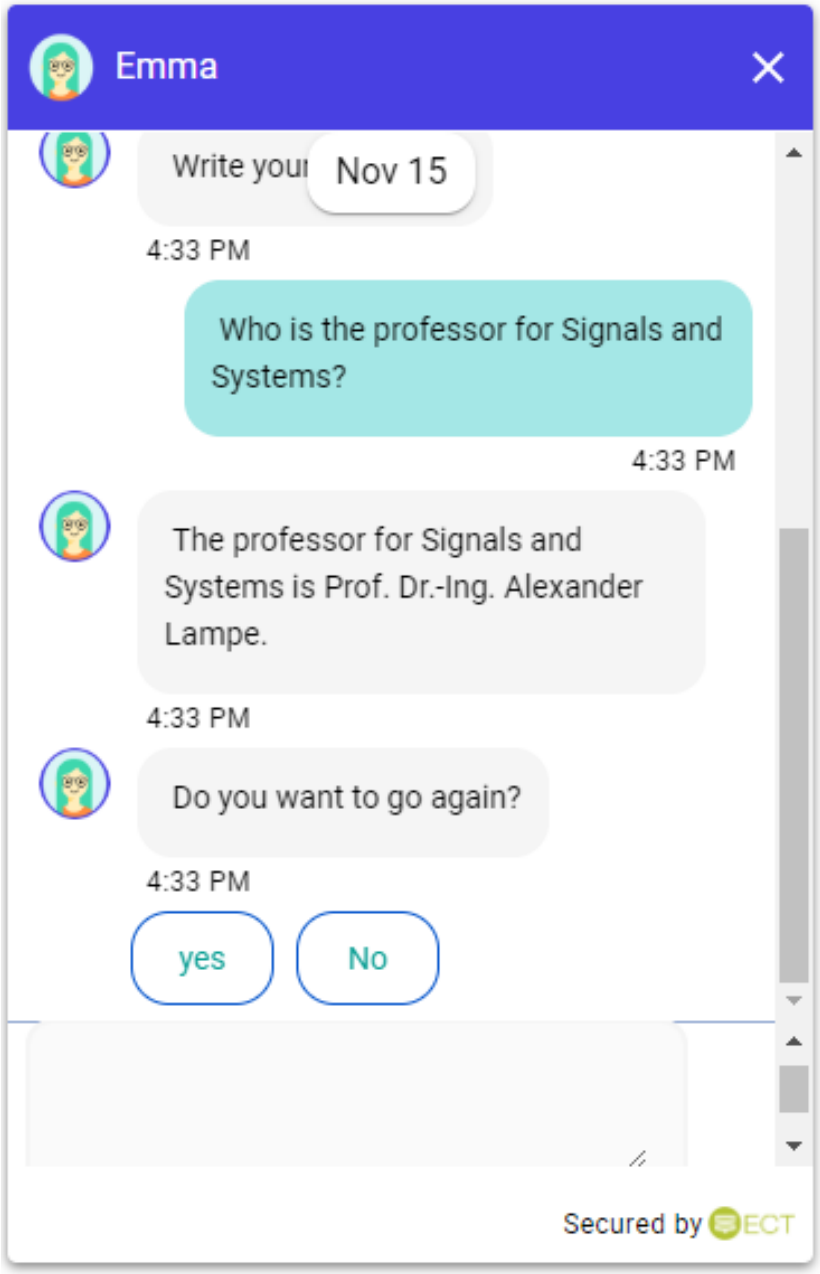

Figure 7.10: Response for question 1.

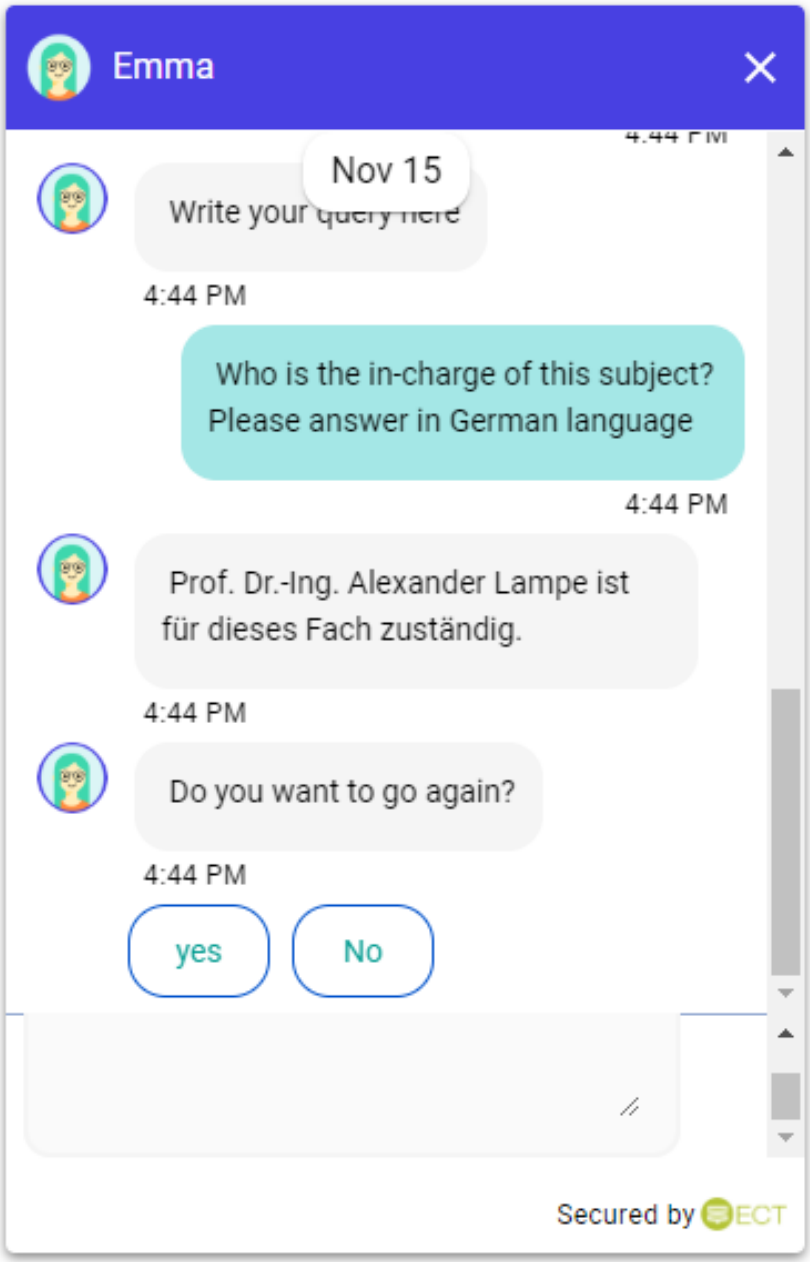

Figure 7.11: Question 2 and it's response.

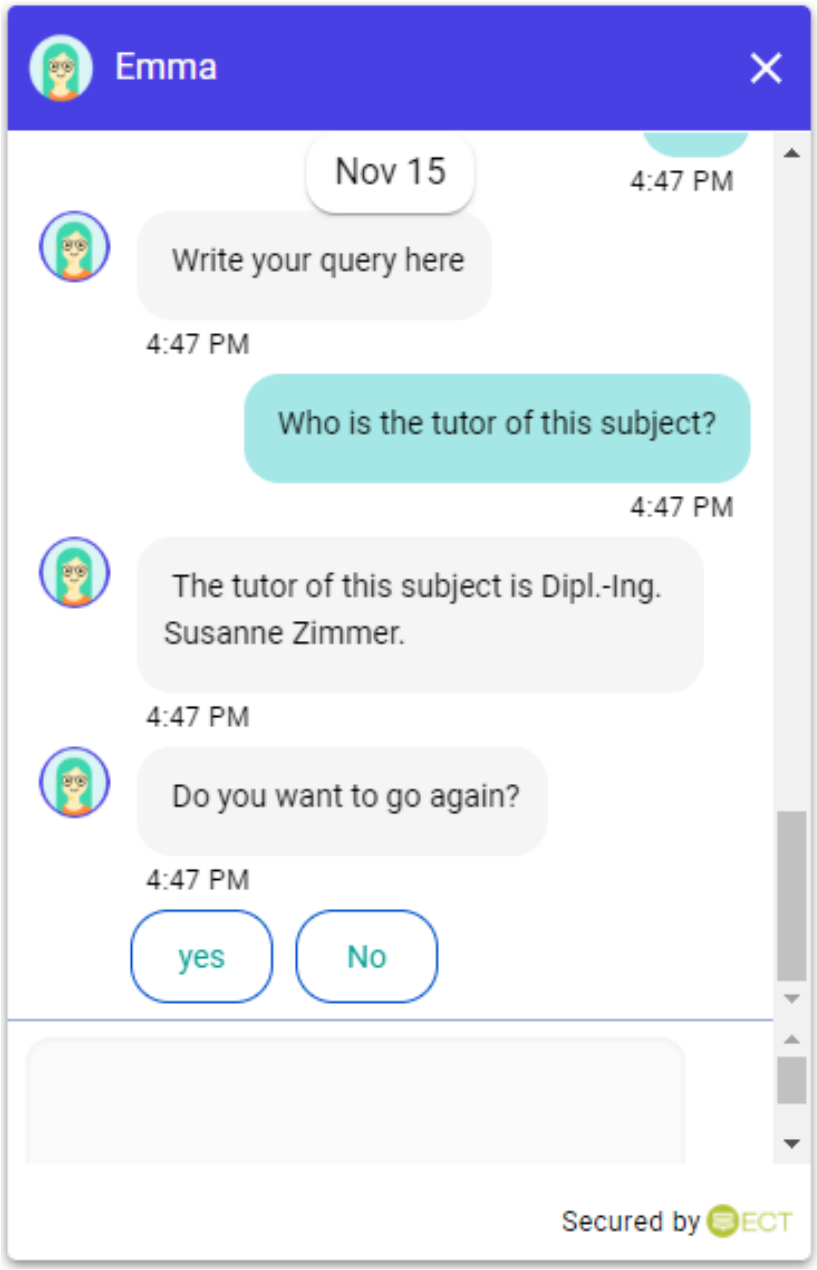

Figure 7.12: Question 3 and it's response.

8 Conclusion and Future Work

The development of an AI-based chatbot within the framework of ECT's Low Code Application marks a significant stride in the realm of conversational AI and document processing. This project, driven by the integration of advanced language models like OpenAI's GPT-3.5 Turbo and LangChain framework, has not only showcased the potential of AI in enhancing user experience but also opened avenues for innovative applications in information retrieval and chatbot technology.

Throughout the internship, the project navigated through various challenges, from effectively handling PDF documents to ensuring seamless communication between the user interface and backend processing. The successful implementation of features like document upload, vectorization, and intelligent response generation demonstrates the robustness and versatility of the system.

Moreover, the practical application of semantic search, vector databases, and retrieval-augmented generation techniques further underscores the project's contribution to the evolving landscape of AI-driven solutions. The chatbot, named Emma, embodies the culmination of these technologies, providing users with an interactive, informative, and engaging experience.

The internship has not only been a learning journey in terms of technical skills but also an insightful experience into the practical applications of AI in real-world scenarios. The project outcomes align with the broader goal of enhancing user interactions and operational efficiency, positioning ECT at the forefront of AI application in low-code platforms.

As we look to the future, there are multiple pathways for the project's evolution. These include the integration of more sophisticated AI models like GPT-4, advancements in multilingual support, and the exploration of multi-modal capabilities. The pursuit of these enhancements will ensure the chatbot's continuous improvement and adaptability to the dynamic needs of users and the ever-evolving technological landscape.

In summary, this internship report encapsulates the journey, challenges, and achievements of developing an AI-based chatbot in a low-code environment. It highlights the significant potential of AI in transforming how we interact with information and paves the way for future advancements in the field of conversational AI.

8.1 Future Work

The implementation of an AI-based chatbot within the ECTs Low Code application is an exciting and promising endeavor. As we progress with this project, several avenues for future work can further enhance the capabilities and usability of our chatbot development platform.

Integration of Advanced AI Models:
One of the key areas for future work in the development of AI-based chatbot implementation in ECTs low code applications is the ongoing integration of cutting-edge AI models. While we

currently rely on robust models such as OpenAI GPT-3.5 turbo and ada-v2 for embedding, it is imperative to remain at the forefront of AI and language model advancements. Therefore, our future endeavors should center on seamlessly incorporating state-of-the-art AI models, including but not limited to models like GPT-4, as they emerge. This strategic evolution is essential to ensure that our chatbots consistently deliver superior conversational quality and a deep understanding of user queries.

Multi-language Support:
Expanding the chatbot's capabilities to cater to a diverse range of languages is of paramount importance, as it enables us to connect with a more extensive and varied user base. In our future endeavors, we must focus on elevating the language model's proficiency in effectively engaging users in languages other than English. Achieving this objective will necessitate comprehensive training and fine-tuning of AI models for a multitude of languages, as well as the seamless integration of language detection and auto-translation features. This holistic approach will ensure that the chatbot provides a more natural and user-friendly multilingual experience.

Multi-Modal Capabilities:
Exploring the integration of multi-modal capabilities represents a promising avenue for future development, aimed at elevating the user experience. This innovation entails enabling chatbots to seamlessly process and generate a diverse array of content, encompassing text, images, and potentially other media formats. By embracing this enhancement, interactions with the chatbot can become more engaging and versatile, opening up new dimensions of user engagement and interaction.

Additionally, consider these future work areas for further development:

- User Profiling and Personalization,
- Continuous Training and Monitoring,
- Enhanced PDF Parsing and Understanding,
- User Analytics and Reporting

By directing efforts toward these areas, we can elevate the AI-based chatbot implementation in our low-code application, enhancing its potency, user-friendliness, and operational efficiency. These improvements will foster superior user experiences and bolster the productivity of the agents who utilize the system.

Bibliography

[1] European Computer Telecoms AG. *ECT Telecoms Homepage*. Accessed: 2023-11-17. 2023. URL: https://www.ect-telecoms.com/ (visited on 11/17/2023).

[2] Magdi Zakaria, AS Mabrouka, and Shahenda Sarhan. "Artificial neural network: a brief overview". In: *neural networks* 1 (2014), p. 2.

[3] Wikipedia. *Cosine Similarity*. Accessed: 2023-11-17. 2023. URL: https://en.wikipedia.org/wiki/Cosine_similarity (visited on 11/17/2023).

[4] StatQuest with Josh Starmer. *Cosine Similarity, Clearly Explained!!!* Accessed: 2023-11-17. 2023. URL: https://www.youtube.com/watch?v=e9U0QAFbfLI (visited on 11/17/2023).

[5] Wikipedia. *Dot Product*. Accessed: 2023-11-17. 2023. URL: https://en.wikipedia.org/wiki/Dot_product (visited on 11/17/2023).

[6] Wikipedia. *Euclidean Distance*. Accessed: 2023-11-17. 2023. URL: https://en.wikipedia.org/wiki/Euclidean_distance (visited on 11/17/2023).

[7] *Qdrant Documentation*. Accessed: 2023-11-17. URL: https://qdrant.tech/documentation/ (visited on 11/17/2023).

[8] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[9] Jay Alammar. *The Illustrated Transformer*. Accessed: 2023-11-17. URL: https://jalammar.github.io/illustrated-transformer/ (visited on 11/17/2023).

[10] OpenAI. *OpenAI Homepage*. Accessed: 2023-11-17. 2023. URL: https://openai.com/ (visited on 11/17/2023).

[11] OpenAI. *OpenAI API Pricing*. Accessed: 2023-11-17. 2023. URL: https://openai.com/pricing (visited on 11/17/2023).

[12] *Sentence Transformers Documentation*. Accessed: 2023-11-17. Hugging Face. 2023. URL: https://huggingface.co/docs/hub/sentence-transformers (visited on 11/17/2023).

[13] LangChain Contributors. *LangChain: A Framework for Language Model-Powered Applications*. Accessed: 2023-03-10. 2023. URL: https://www.langchain.com (visited on 03/10/2023).

[14] Sandeep Kasinampalli Rajkumar. *LangChain: Engage in Conversations with Your Data*. 2023. URL: https://medium.com/@sandeepkrajkumar/langchain-engage-in-conversations-with-your-data-1244d2a431e4 (visited on 11/17/2023).

[15] *LangChain: Chat With Your Data*. Accessed: 2023-11-17. DeepLearning.AI. URL: https://learn.deeplearning.ai/langchain-chat-with-your-data/lesson/1/introduction (visited on 11/17/2023).

[16] *Docker Documentation*. Accessed: 2023-11-17. Docker. 2023. URL: https://docs.docker.com/ (visited on 11/17/2023).