University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department

BACHELOR THESIS

Retrieval-Augmented Generation for Enhancing Large Language Models with Contextual Knowledge: A Chat Assistant for Organizational Knowledge

Scientific Advisers:
Drd. Ing. Mihai-Valentin Dumitru
Conf. Dr. Ing. Matei Popovici

Author:
Ana-Maria Toader

Bucharest, 2025

Universitatea POLITEHNICA din București

Facultatea de Automatică și Calculatoare,
Departamentul de Calculatoare

LUCRARE DE DIPLOMĂ

Generare Augmentată prin Regăsire pentru Îmbunătățirea Modelelor Lingvistice Mari cu Cunoștințe Contextuale: Un Asistent Conversațional pentru Informații Organizaționale

Conducători Științifici:
Drd. Ing. Mihai-Valentin Dumitru
Conf. Dr. Ing. Matei Popovici

Autor:
Ana-Maria Toader

București, 2025

Abstract

In recent years, large language models have gained a significant growth in popularity for text generation tasks. This thesis focuses on the usage of pre-trained deep learning models, exploring retrieval-augmented generation as a method for improving the contextual relevance of large language model outputs by incorporating external knowledge sources during inference. To demonstrate this approach, an implementation of a chatbot is proposed, designed to interact with an organizational knowledge base and provide source-grounded question answering. Additionally, an overview of the concepts behind building such a system is provided, presenting the motivation behind it by highlighting the limitations of standard generative language models.

The proposed chatbot solution enables the citation of the sources used for response generation, allowing users to verify the factual accuracy of the information. The implementation is based on a modular architecture, consisting of components for query rewriting, information retrieval, reranking, and answer generation. This approach aims to provide flexibility and allow customization of independent modules. To evaluate this implementation, experiments have been conducted, and the performance has been assessed using a large language model as a judge to rate the relevance of the retrieved context to the user query, as well as the faithfulness and relevance of the generated responses.

Sinopsis

În ultimii ani, modelele lingvistice mari au dobândit o creștere semnificativă în popularitate pentru sarcini de generare de text. Această lucrare se concentrează asupra utilizării modelelor de învățare profundă pre-antrenate, explorând generarea augmentată prin regăsire ca metodă pentru îmbunătățirea relevanței contextuale a rezultatelor modelelor lingvistice mari prin incorporarea bazelor de cunoștințe externe în timpul inferenței. Pentru a demonstra această abordare, este propusă o implementare a unui asistent conversațional conceput pentru a interacționa cu o bază de cunoștințe organizaționale și a oferi răspunsuri fundamentate pe surse. De asemenea, se oferă o prezentare generală a conceptelor necesare pentru implementarea unui astfel de sistem, expunând motivația din spatele acestuia prin evidențierea limitărilor modelelor lingvistice generative.

Soluția de asistent conversațional propusă permite citarea surselor folosite pentru generarea răspunsului, oferindu-le utilizatorilor posibilitatea de a verifica acuratețea factuală a informațiilor. Implementarea se bazează pe o arhitectură modulară, compusă din componente pentru reformularea întrebărilor, regăsirea informației, reordonarea informației și generare. Această abordare își propune să ofere flexibilitate și să permită personalizarea modulelor independente. Pentru a evalua această implementare, au fost efectuate experimente și performanța a fost apreciată folosind un model lingvistic mare pentru a oferi un scor pentru relevanța contextului regăsit față de întrebarea utilizatorului, precum și fidelitatea și relevanța răspunsurilor generate.

Contents

Notations and Abbreviations

AI – Artificial Intelligence
DL – Deep Learning
DNN – Deep Neural Network
DPR – Dense Passage Retrieval
GenAI – Generative AI
GPT – Generative Pre-trained Transformer
KNN – K-Nearest Neighbor
LLM – Large Language Model
ML – Machine Learning
NLG – Natural Language Generation
NLP – Natural Language Processing
OCR – Optical Character Recognition
RAG – Retrieval-Augmented Generation

1 Introduction

Natural language processing (NLP) is a subfield of artificial intelligence (AI) that enables computers to process data in the form of human language. Among the many applications of NLP is the production of human-understandable textual data, a subfield known as natural language generation (NLG), which typically involves rule-based or machine learning models that convert structured data into human language. Generative AI (GenAI) is a wide field that encompasses AI models that learn complex patterns from large datasets and generate new content, rather than simply transforming structured data into language. GenAI for NLG aims to produce more creative and contextually appropriate language, through the incorporation of advanced deep learning models, such as transformers [1].

1.1 Context and Motivation

In today's rapidly evolving digital age, large amounts of information become available each day. With the overwhelming volume of existing information, the need for intelligent systems that can facilitate the processing and understanding of data has become undeniable. Large language models (LLMs) have greatly impacted the field of natural language processing, allowing the understanding and generation of natural language at unprecedented scales. The increasing demand for automated assistance has led to the development of chatbots, which often rely on deep learning models to provide real-time responses to user questions, facilitating efficient access to information. Figure 1.1 shows the number of weekly active users of such a system (ChatGPT[1]), showing a consistent growth that highlights the increase in popularity over time.

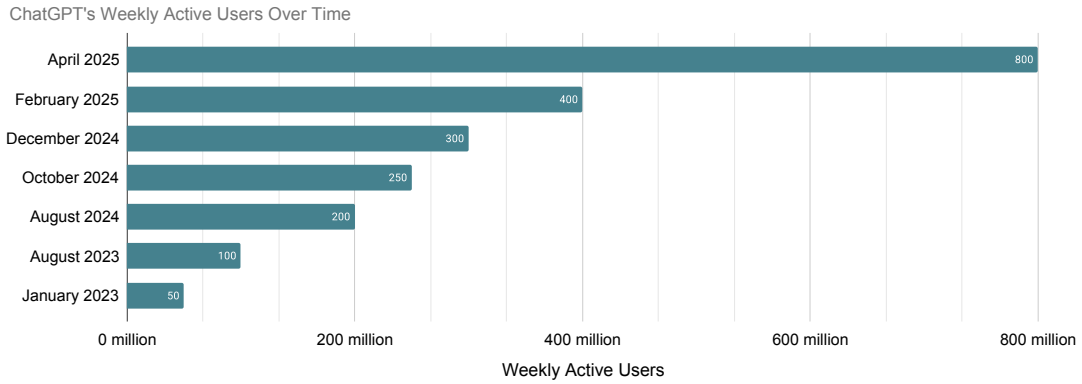

Figure 1.1: ChatGPT's weekly active users over time[2]

Modern chatbots use LLMs to generate responses based on the user query and the data the model has learned during training, typically from large corpora of text, which could include books, scientific papers, news articles, public forum comments, blog posts, and user reviews. Despite significant increases in the number of tokens in the pretraining data, with public datasets such as RedPajama-Data-v2[3] exceeding 30 trillion, LLMs still have limited access to data. Private data, as well as very recent data, are not included in the pre-training of these models. In addition to this, the context window in recent models has surpassed hundreds of thousands of

[1] https://chatgpt.com/
[2] Data from https://www.demandsage.com/chatgpt-statistics/
[3] https://huggingface.co/datasets/togethercomputer/RedPajama-Data-V2

tokens[1], allowing us to fit information from external sources into the prompt. To further improve these modern systems, retrieval-augmented generation (RAG) [2] was introduced, leveraging optimized search algorithms to retrieve information from external sources and use it to enhance the context of the LLM, enabling more grounded and informative responses. In essence, RAG first fetches the most relevant documents to an input query from a prebuilt vector index, a specialized database designed for efficient retrieval, typically via vector-embedding similarity metrics. It then incorporates these documents into the construction of a new prompt which is passed on to a language model to generate a contextually enriched response.

1.2 Problem

Many of the existing RAG chatbot solutions (e.g. Amazon Bedrock Knowledge Bases[2], Azure AI Search[3]) rely on cloud-based infrastructure for storing and accessing data, which may raise concerns about data privacy. Whether for organizations that handle sensitive or confidential information, or individual users concerned with the security and privacy risks of relying on a third-party cloud platform, the benefit of a local solution is recognizable.

In addition, commercial solutions often lack transparency and control over the technologies used. Due to the closed-source nature of most existing solutions, customization of such systems may be limited. Customization is particularly important for non-English language requirements, as it involves adapting to handle other languages, such as Romanian, by using multilingual models pre-trained on the specific language. Many off-the-shelf solutions, such as Microsoft Copilot[4] or Cohere Coral[5], are limited in the selection of models that can be used.

The evaluation of such systems is not straightforward, often requiring subjective means of assessing user satisfaction and relevance of responses. Due to the lack of a universally agreed upon metric to measure accuracy, properly configuring a chatbot proves challenging.

1.3 Objective

This work describes the development of a retrieval-augmented generation [2] chatbot capable of answering questions in Romanian, based on documents scraped from UNSTPB[6] websites. Given that the output of such systems may impact decision-making, this solution seeks to include linked source citations of the information used in the response, intended to support fact-checking and to improve access to organizational knowledge. At the same time, this thesis aims to provide a comprehensive overview of the underlying theoretical foundations necessary for building such a system. In order to offer a solution to organizations where information is often spread across numerous documents, either private or publicly available, a complete end-to-end pipeline is constructed that integrates data collection, data processing, indexing, query refining, retrieval, reranking, response generation and evaluation.

The main contributions lie in the design and implementation of a modular architecture, leveraging pre-trained transformer-based models [1] in multiple components, deploying and configuring a local OpenSearch[7] cluster for vector-based retrieval, as well as constructing a complete data collection and processing pipeline that incorporates web crawling, data cleaning, chunking, embedding, and indexing.

[1] https://platform.openai.com/docs/models/compare
[2] https://aws.amazon.com/bedrock/knowledge-bases/
[3] https://azure.microsoft.com/en-us/products/ai-services/ai-search
[4] https://developer.microsoft.com/en-us/copilot
[5] https://docs.cohere.com/cohere-documentation
[6] https://upb.ro/en/
[7] https://docs.opensearch.org/docs/latest

2 Background

In this chapter, I will provide an overview of the theoretical background required to understand the concepts specified in this thesis. I will present details about existing notions and architectures, as well as underline some of the motivations behind them.

2.1 Deep Learning

The amount of data available in today's modern world far exceeds human ability to process it. One of the goals of artificial intelligence is to develop intelligent systems capable of processing such large volumes of information. Deep learning (DL) is a subfield of AI which relies on applying neural networks with many layers to large volumes of data in order to build analytical models to solve complex tasks, such as image classification, machine translation or text generation [3].

Artificial neural networks are designed to mimic on a high level the human brain anatomy, modeling complex mathematical functions. They are composed of a sequence of layers of computational units called artificial neurons, first introduced by McCulloch and Pitts in 1943 [4]. Deep neural networks are a particular type of such networks with multiple hidden layers.

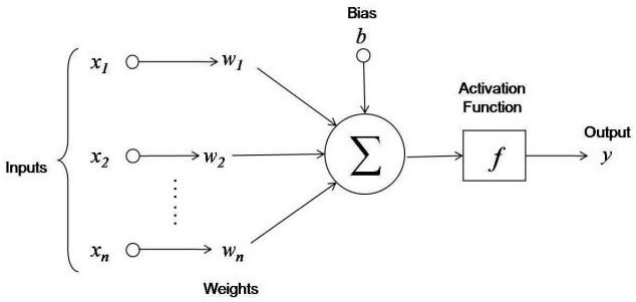

Figure 2.1: Graphical representation of an artificial neuron[1]

Figure 2.1 shows the structure of a modern artificial neuron, defined by a weight vector $(w_1, w_2, ..., w_n)$, a bias b that allows for a better function approximation, and an activation function f that introduces non-linearity, enabling the representation of more complex mathematical functions. Given an n-dimensional numerical input $(x_1, x_2, ..., x_n)$, the neuron models a mathematical function, computing the output y as:

$$y = f(\sum w_i x_i + b) \tag{2.1}$$

The structure of a deep neural network, as shown in Figure 2.2, consists of an input layer, where each neuron receives one input feature, an output layer which produces the final representation, and multiple hidden layers responsible for learning abstract representations from the data, mapping nonlinearities between the input layer and the output layer.

The learning process in an artificial neural network refers to improving its ability to predict and generalize based on patterns in the data it is exposed to. The training objective is to optimize the values of the parameters to make better predictions, minimizing a global error.

[1] https://blogs.cornell.edu/info2040/2015/09/08/neural-networks-and-machine-learning

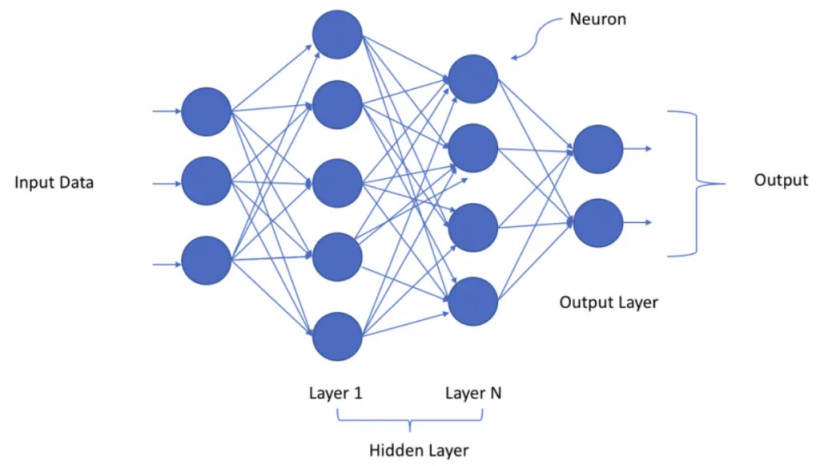

Figure 2.2: Graphical representation of a deep neural network[1]

Backpropagation [5] is the main weight optimization approach used in supervised learning, which is a machine learning paradigm in which the dataset is labeled with a true output value y for each input x. The model uses a predefined loss function $Loss = \mathcal{L}(y_i, \hat{y}_i)$ to quantify the error between the correct output y_i and the predicted output $\hat{y}_i$. The backpropagation algorithm consists of two phases:

- forward pass - in which the input is propagated through the layers to compute the predicted output of the model
- backward pass - in which the values of the model weights w_i are updated, typically based on a variant of the gradient descent algorithm [6], that uses the gradient of the loss function $\mathcal{L}$ with respect to w_i to adjust the weights, in order to minimize the loss function.

Deep learning provides an automated way of learning features from large amounts of data, in order to build meaningful, compact representations, overcoming the limitations of manual feature extraction methods, which are slow and require domain-specific knowledge.

2.2 Natural Language Processing

Natural language processing is a branch of artificial intelligence which focuses on enabling machines to understand, analyze, and generate human language [3]. Early NLP methods relied on manual feature engineering, followed by complex algorithms to process the extracted features. Advances in the field of deep learning triggered a shift to a data-driven approach, using deep neural networks.

An important objective in NLP is to provide an in-depth representation of language. Given that most of the existing data is unlabeled, deep learning allows for unsupervised feature learning, generating low-dimensional representations from high-dimensional data.

Many NLP models are sequence-to-sequence (seq2seq), where both the input and the output are sequences. Typical seq2seq models are based on an encoder-decoder architecture, with an encoder responsible for processing the input data and generating an intermediary value and a decoder which then consumes it in order to provide the final output sequence.

[1] https://product-leader.medium.com/deep-diving-into-neural-networks-and-deep-learning-154562ffe0e9

2.3 Large Language Models

Large language models are advanced pre-trained statistical models that use deep neural networks to process human language, solving complex natural language processing problems such as translation, summarization, and text generation. Such models have an increasingly large number of parameters, reaching hundreds of billions [7], and are pre-trained in a self-supervised manner on large corpora of raw text to predict the next token[1] given an input sequence. In this way, LLMs can learn generic representations of natural language without the need of labeled data. Many modern large language models are based on the Transformer architecture [1].

According to [8], based on the directionality of text processing, LLMs may be classified as:

- **Unidirectional**: These models are autoregressive, predicting the next token based solely on the previous tokens, without acces to "future" information.
- **Bidirectional**: These models build contextual representations of tokens, based on every token in the sequence, not limited to only previous ones. Due to this, they are not well suited for text generation, but are mainly used in tasks that require the understanding of context from both directions, such as semantic text similarity or named entity recognition.

In [7] a timeline of LLM releases is presented, as shown in Figure 2.3. The blue cards contain pre-trained models that use raw text to learn general patterns and representations to model natural language, with the objective of text generation. The models in the red cards are fine-tuned on specific data for various tasks, such as multimodal interaction (e.g., GPT-4o), or code generation and programming tasks (e.g., Wizard-Coder). The models above the horizontal line are open source, whereas the ones below it are closed source.

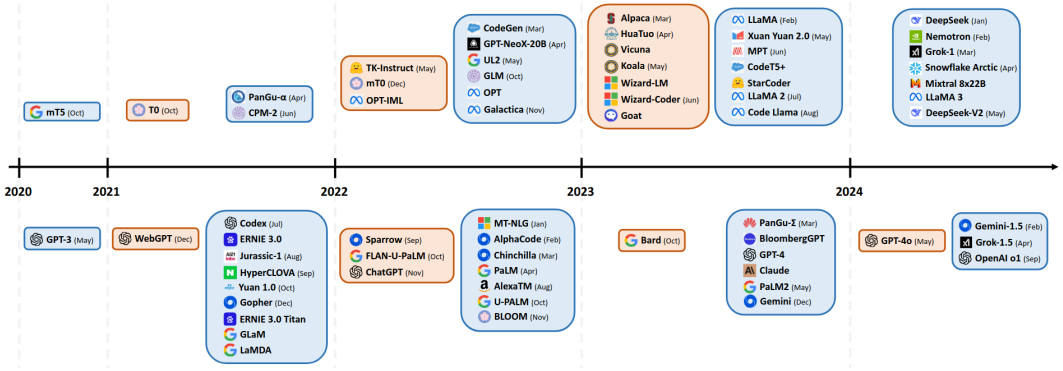

Figure 2.3: Chronological display of LLM releases, image adapted from [7]

The chart shows the fast pace at which new LLMs are released, illustrating the rapid evolution of this field. As scientific research advances, more methods of improving existing solutions appear, generating increasing interest in LLMs, which are gaining popularity worldwide.

2.3.1 Training Process

The training process of a large language model typically includes the following main steps [9]:

- **Data preparation:** Includes the collection, filtering, and deduplication of data.
- **Tokenization:** Consists of splitting the input text sequence into tokens[1].
- **Positional encoding:** Used as a way to preserve the order of elements in a sequence, without the need of recurrence.

[1]Tokens represent the atomic units on which language models operate. They may be words or fragments of words.

- **Pre-training:** Represents a self-supervised learning process in which the model is trained on vast amounts of data, with the objective of predicting subsequent tokens in an autoregressive manner, based on an input sequence. In this way, LLMs acquire a general understanding by developing internal representations.
- **Fine-tuning:** Represents a supervised learning process where the model is trained on labeled data to perform specific tasks. During pre-training, LLMs are optimized for next token prediction. Further training of the model on a set of `<instruction, response>` pairs is designed to enhance its ability to perform particular tasks, and has been shown to increase performance on unseen tasks as well [10].

2.3.2 Hallucinations

One of the core challenges in natural language generation is the phenomenon of hallucinations, described as generated content that is either unfaithful to the provided content, deviating from the source material, or simply nonsensical [11]. Without verifying the factuality of responses using trusted sources, it has potentially harmful influence over decision-making, as the output of large language models is used by people worldwide in an increasing variety of contexts.

A real-world example of the impact of hallucinations in large language models comes from the legal sector, in the Mata v. Avianca case[1], where a man sued Avianca Airlines claiming that he had suffered an injury on one of the airline's flights. The plaintiff's attorney unknowingly relied on ChatGPT-generated citations of relevant court decisions that later proved to be fabricated, drawing global attention, and highlighting the legal consequences of LLM hallucinations.

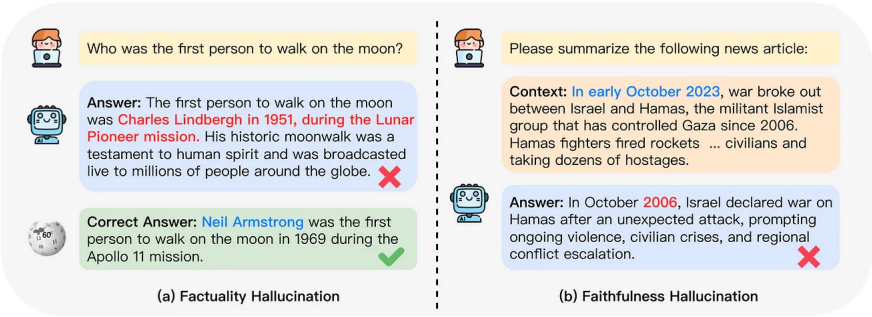

Figure 2.4: Hallucination taxonomy [11] examples: factuality and faithfulness[2]

A survey authored by Huang et al. [11] categorizes hallucinations into two main types, considering the alignment with the source: intrinsic and extrinsic. Intrinsic hallucinations refer to generated content that deviates from the sources, whereas extrinsic hallucinations describe content that is unverifiable using the provided context. A taxonomy is presented with two types of hallucinations, as shown in Figure 2.4:

- **Factuality hallucinations** occur when the model generates content that is either factually incorrect or unverifiable against real-world knowledge. The left-hand side of the image shows an example of a generated response that contradicts information available from trusted sources. This may be attributed to the model's lack of parametric factual knowledge about the required information.

[1] https://www.reuters.com/legal/new-york-lawyers-sanctioned-using-fake-chatgpt-cases-legal-brief-2023-06-22/

[2] https://blog.athina.ai/a-survey-on-hallucination-in-large-language-models-principles-taxonomy-challenges-and-open-questions

- **Faithfulness hallucinations** refer to model responses that are not aligned with user instructions, conflicting with the provided context, or even exhibit logical inconsistencies. The example shows the model being prompted with a summarization task, to which it provides a response that directly conflicts with the prompted context.

In an attempt to mitigate hallucinations, the proposed chatbot solution uses retrieval-augmented generation [2] to fetch contextually meaningful information from external sources and provide it to the LLM to enhance its context awareness. The main strategy involves ensuring contextual alignment between the generated response and the retrieved documents by explicitly attributing information to its source. This mechanism, denoted as source attribution [11], aims to identify the origins of information, requiring the model to cite specific passages in order to allow a faithful alignment with the context. This enables users to verify the information by tracing the responses to the original sources.

2.4 Word Embeddings

In order for computers to process human language, it needs to be encoded into numerical form. In NLP, words (more specifically tokens, which may be words or smaller parts of words) are transformed into numerical vector representations referred to as embeddings.

One early approach to this is one-hot encoding, where each word of a dictionary of size d is represented as a d-dimensional sparse vector, containing a single value of "1" and "0" in the rest. Despite its simplicity, this method has numerous problems, such as the high dimensionality of vectors for a large vocabulary, increasing the memory and time complexity of models, as well as the inability to capture semantic relationships between words.

In modern NLP applications, neural networks are used to obtain word embeddings in a continuous high-dimensional vector space. These capture semantic meaning and the relationship between words, supporting reasoning in vector spaces based on the offset between the embeddings. A classic example for this is presented in [12], where the neural network-based model Word2Vec was found to have learned the male/female relationship, such that if we denote the embedding vector of a word w as $\vec{E_w}$, $\vec{E}_{king} - \vec{E}_{man} + \vec{E}_{woman}$ results in a vector located close to $\vec{E}_{queen}$, as suggested in Figure 2.5. This suggests that the direction determined by the offset $\vec{E}_{man} - \vec{E}_{woman}$ encodes gender information.

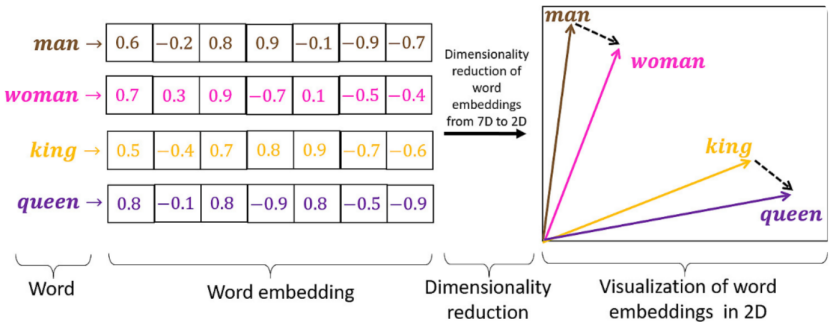

Figure 2.5: Semantic male/female relationship captured by word embeddings[1]

[1] https://airbyte.com/data-engineering-resources/sentence-word-embeddings

2.5 Transformers

An important contribution that led to remarkable advances in the field of natural language processing is the introduction of the Transformer model architecture in 2017, shown in Figure 2.6 [1]. The outstanding impact it had was mainly due to the fact that it allowed for parallelization at unprecedented scales for tasks such as language modeling or machine translation. It overcame the limitations of sequential computation imposed by the state-of-the-art approaches of that period, such as recurrent neural networks. Recurrent models process sequences one position at a time and generate for each input a hidden state, each based also on the hidden state at the previous step and the input.

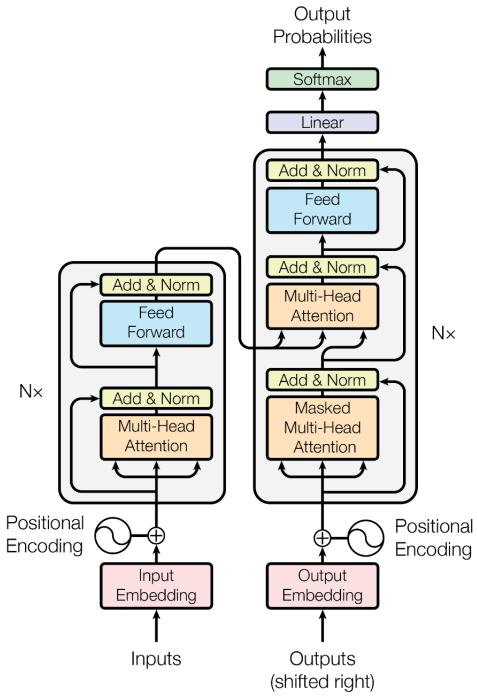

Figure 2.6: Transformer architecture [1]

Transformers are machine learning models based on deep neural networks with an encoder-decoder architecture. The encoder is used to generate a fixed-length vector representation from an input sequence, which consists of word embeddings augmented with positional encoding that provides information about the order of tokens, allowing the model to differentiate between identical tokens found at different positions. The decoder is responsible for decoding the fixed-dimensionality vector into a target sequence, generating symbols one by one.

Transformers leverage the mechanism of attention to eliminate the use of recurrence. Introduced by Bahdanau et al. in 2014 [13] as a way to improve the encoder-decoder architecture used in neural machine translation, it allows the model to focus on different parts of the input sequence without the bottleneck of forming a fixed representation that often failed to capture the whole meaning, especially when dealing with long sequences. Transformers rely on self-attention, in which each token attends to every other token in the input sequence, including itself.

The meaning of a word may differ depending on the context. Self-attention enables models to dynamically adjust the representation of a token based on the importance of the others. The attention mechanism is described as a mapping of a query and a key-value pair to an output [1]. Figure 2.7 shows a visual representation of the computations in a self-attention head. For

an input $x^{(i)}, i \in [1, T]$, where T is the length of the input sequence, learned weight matrices W_q, W_k, W_v are used to compute:

- Query vector $q^{(i)}$ - which is used to determine how much the current input token $x^{(i)}$ should attend to any other token in the input sequence (including itself).
- Key vector $k^{(i)}$ - which is used to determine the relevance of a token in the input sequence to the query vector of the current token.
- Value vector $v^{(i)}$ - which represents the actual value used to update the input embeddings.

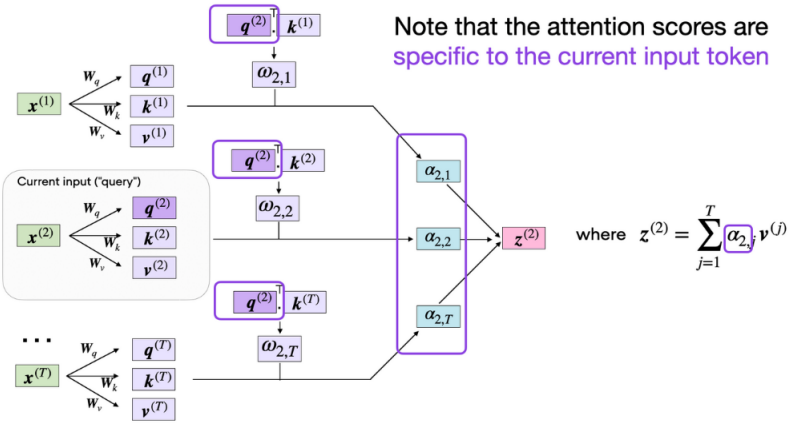

Figure 2.7: Visual representation of the self-attention mechanism computations[1]

The dot product between vectors $q^{(i)}$ and $k^{(j)}$, denoted by $\omega_{i,j}$, is scaled to prevent gradient instability during training, and normalized into a probability distribution to obtain the attention weight $\alpha_{i,j}$. The contextualized embedding vector $z^{(i)}$ is computed as a weighted sum of the value vectors $v^{(j)}, j \in [1, T]$, where the weights represent the attention scores $\alpha_{i,j}$.

The decoder part of the architecture uses masked self-attention, a variant of the self-attention mechanism that "masks" out all tokens that follow the current one, allowing the model to only consider known, previous tokens and disregard all "future" information. This is important to ensure that all predictions are strictly dependent on previously known outputs.

The transformer architecture uses multi-head attention, with one attention head responsible for computing one set of attention scores. By using multiple attention heads, vectors q, v, k are projected into multiple subspaces, allowing the model to focus on multiple parts of the input sequence in parallel. The outputs of all attention heads are concatenated and linearly projected for original dimensional alignment. The model is autoregressive, appending the previously generated output tokens to the input sequence to use for predicting the next token.

2.5.1 GPT

Generative Pre-trained Transformer (GPT) is a family of autoregressive models, introduced by OpenAI in [14] as a way to improve upon limitations of supervised learning techniques in natural language processing. Producing manually labeled data is often too resource intensive and challenging, restricting the applicability of supervised learning-based models in many fields.

GPT is based on the Transformer architecture [1] and was introduced as a semi-supervised deep learning model that aimed to learn a universal representation that easily adjusted to multiple tasks. The training framework consists of two phases:

[1] https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

1. **Unsupervised pre-training**. The model is pre-trained on large corpora of unlabeled text, with the objective of language modeling, which involves predicting the next token based on an input sequence of tokens.
2. **Supervised fine-tuning**. The model is fine-tuned on a smaller, labeled dataset, to adjust the learned parameters to a specific task, such as sentiment analysis, machine translation or question answering.

Using only the decoder part of the Transformer architecture, GPT takes in as input a prompt and generates in an autoregressive manner the predicted next tokens one by one. Focusing on text generation, the masked self-attention mechanism in the decoder is well-suited for the model, allowing each token to only attend to previous tokens, in an unidirectional manner.

2.5.2 BERT

Bidirectional Encoder Representations from Transformers (BERT) [15] is a transformer-based language representation model that introduced deep bidirectional context. The BERT transformer uses a bidirectional variant of the attention mechanism, in which each token can attend to every other token in the input sequence, without being restricted to follow the natural left-to-right order. Using only the encoder from the original Transformer architecture, it was designed to offer a deep understanding of natural language in order to be easily fine tuned for a wide variety of NLP tasks without significant architectural changes.

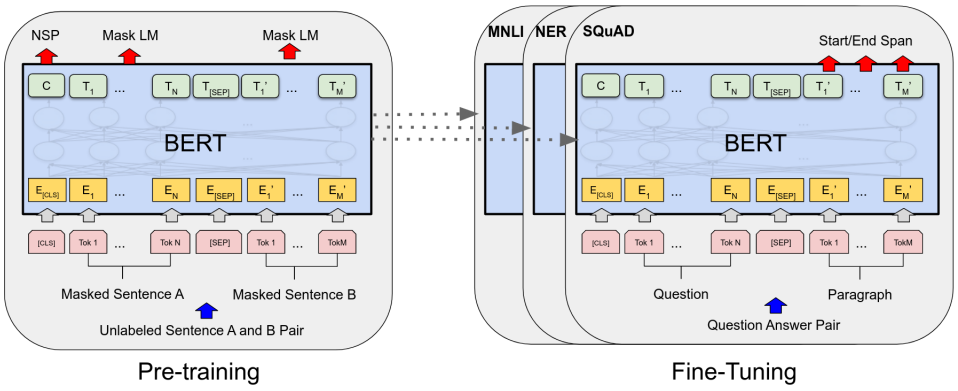

Figure 2.8: BERT pre-training and fine-tuning overview [15]

Figure 2.8 shows an overview of the two steps in the BERT training framework:

- **Pre-training** on two unsupervised tasks: masked language modeling (MLM) and next sentence prediction (NSP). In MLM a portion of the total tokens in the input sequence are "masked", and the model learns to predict them. NSP aims to encode the relationships between two sentences A and B by predicting whether sentence B follows sentence A. It generates labels in a self-supervised manner, using a balanced dataset containing true consecutive sentence pairs and randomly paired sentences.
- **Fine-tuning** the pre-trained model parameters by adapting its output layer to match the structure and objective of many possible supervised tasks.

BERT's input representation shown in Figure 2.9 is a sequence of tokens obtained from a single sentence or a pair of two sentences. Each input sequence is prefixed with a $[CLS]$ classification token, which produces a final hidden state used in classification tasks. Two sentences are separated by a $[SEP]$ token. The final representation of each token is obtained as a sum of the

corresponding token embedding, a segment embedding ($E_A or E_B$) indicating the token's source sentence, and a positional embedding E_i to indicate the i^{th} position in the token sequence.

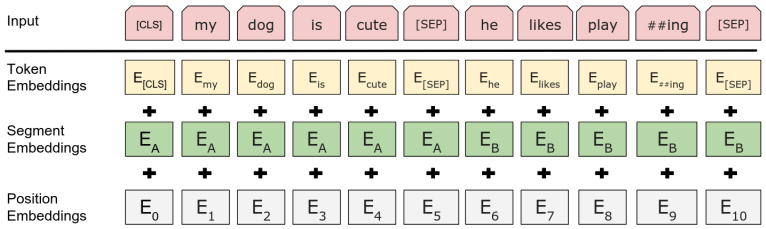

Figure 2.9: BERT input representation [15]

2.5.3 SBERT

Sentence-BERT (SBERT) [16] adds a pooling layer on top of BERT and was designed to output semantically meaningful continuous vector representations for sentences. This allows embeddings to be precomputed individually for each sentence, significantly reducing computational time for tasks such as information retrieval using semantic search. BERT requires a pair of sentences to be passed as input to a transformer network, whereas SBERT uses a siamese network structure consisting of two identical subnetworks that share the same weights to encode sentences independently in the same vector space.

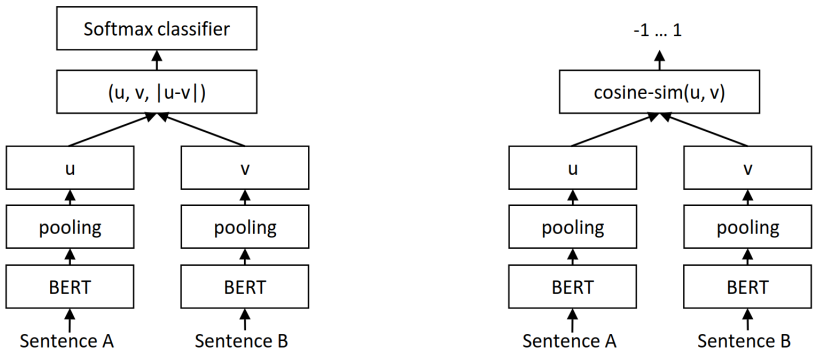

Figure 2.10: SBERT architecture overview with different objective functions [16]

The SBERT architecture may be used with many different objective functions. Figure 2.10 shows two such examples. Both variants have two input sentences A and B pass through a BERT model to produce a sequence of token embeddings. A pooling layer is added to obtain a single vector representation for each input sentence, u and v. Different pooling strategies may be applied, such as computing the mean value of all output vector embeddings. The left variant is used with a classification objective function, concatenating u, v and $|u - v|$ to form an input passed through a Softmax classifier to produce probabilities over discrete labels. The right variant is used with a regressive objective function, applying a cosine similarity metric to output a real value in the range $[-1, 1]$, representing a similarity score for the input sentences.

2.6 Retrieval-Augmented Generation

Large language models are pre-trained on massive volumes of data and have been shown to store factual knowledge [8]. Despite this, their knowledge base remains limited and fails to incorporate recent information, as well as private information that is not present in the training data. Retrieval-augmented generation was introduced in [2] as way to overcome these limitations.

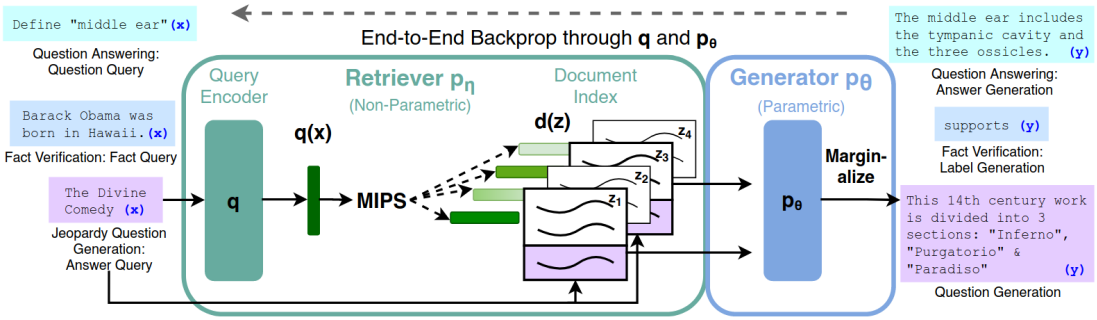

Figure 2.11: Overview of RAG approach [2]

The proposed architecture shown in Figure 2.11 combines a pre-trained generative model and a dense vector index accessed with a pre-trained retriever. The left-hand colored boxes show examples of inputs for different possible tasks: question answering, fact verification and question generation. The input sequence x is passed through a *Query Encoder* q, highlighted in the green block, which is a transformer, typically a BERT model. This generates a fixed-length continuous vector representation $q(x)$ which is differentiable, so that gradients can be used to update the weights during backpropagation. The vector $q(x)$ is used to perform Maximum Inner Product Search (MIPS) over a large vector index of pre-encoded Wikipedia passages, each represented as an embedding $d(z)$. This component describes the *Retriever* p_η, typically performing this search using an Approximate Nearest Neighbor (ANN) algorithm to efficiently output a distribution over candidate documents, and the top k most relevant passages are selected. Due to the fixed nature of the document embeddings used for retrieval, which are not updated during training, this memory is referred to as non-parametric. The k retrieved documents are then passed through a *Generator* p_θ, a seq2seq transformer that receives both the query and the additional context and produces a final response that is grounded by external knowledge.

3 Proposed Solution

In this chapter I will provide a detailed explanation of the proposed solution, presenting the implementation details and the motivations behind them.

3.1 Architecture Overview

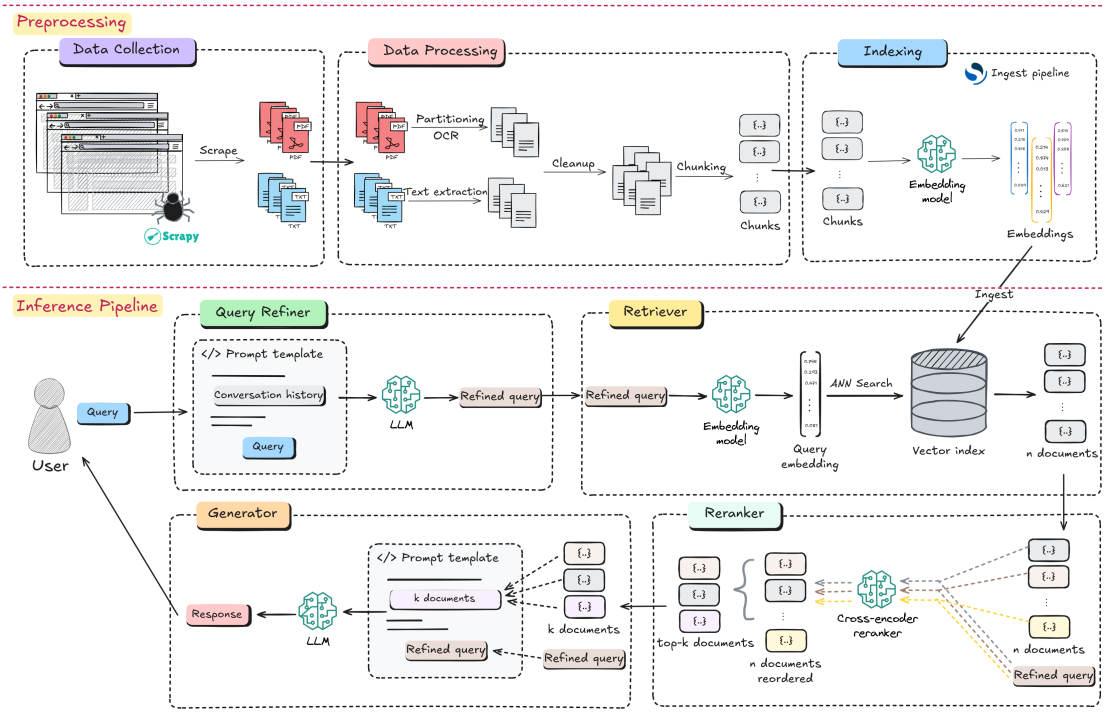

Figure 3.1: Architecture overview

The proposed retrieval-augmented generation chatbot pipeline shown in Figure 3.1 consists of two main parts: preprocessing and inference.

The preprocessing pipeline is executed once to prepare the knowledge base, or periodically to update it. It contains a data collection component, responsible for gathering relevant data from source websites, a data processing component that cleans up and splits the documents into smaller, semantically meaningful chunks, and an indexing component that embeds chunks into vector representations and ingests them into a vector store.

The inference pipeline is triggered by a user's input and begins with a query refiner component that aims to provide a more enhanced version of the user query for better retrieval, considering the conversational history. It is followed by a retriever that fetches the document passages from the vector index that are most similar to the refined query. A reranker component then combines information from the query and the retrieved documents to reorder them for a more fine-grained semantic similarity. The generator component integrates the top ranked passages and the refined query into a prompt, which is passed to a large language model to generate a contextually-grounded response.

3.1.1 Chatbot Workflow

The chatbot implementation is based on LangGraph[1], defining the system as a state machine, with nodes corresponding to functions that interact with pipeline components, reading and writing to a shared state, and edges that specify the logical transitions between them, as shown in Figure 3.2. LangChain[2] was used for the query refiner and generator components, offering integrations with OpenAI API[3]. The final processing node in the state graph is responsible for storing the user query and generated response to a list, removing old entries when a certain capacity is reached, comparable to a least-recently used caching mechanism.

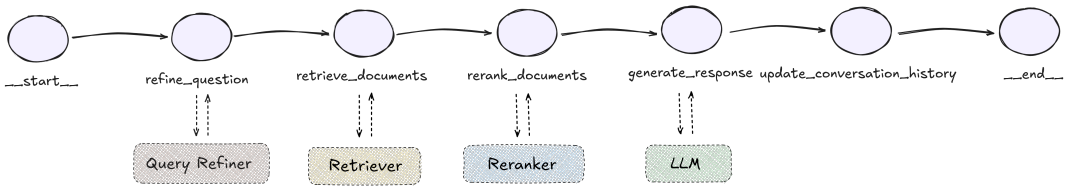

Figure 3.2: LangGraph RAG chatbot workflow

3.1.2 Software Components Overview

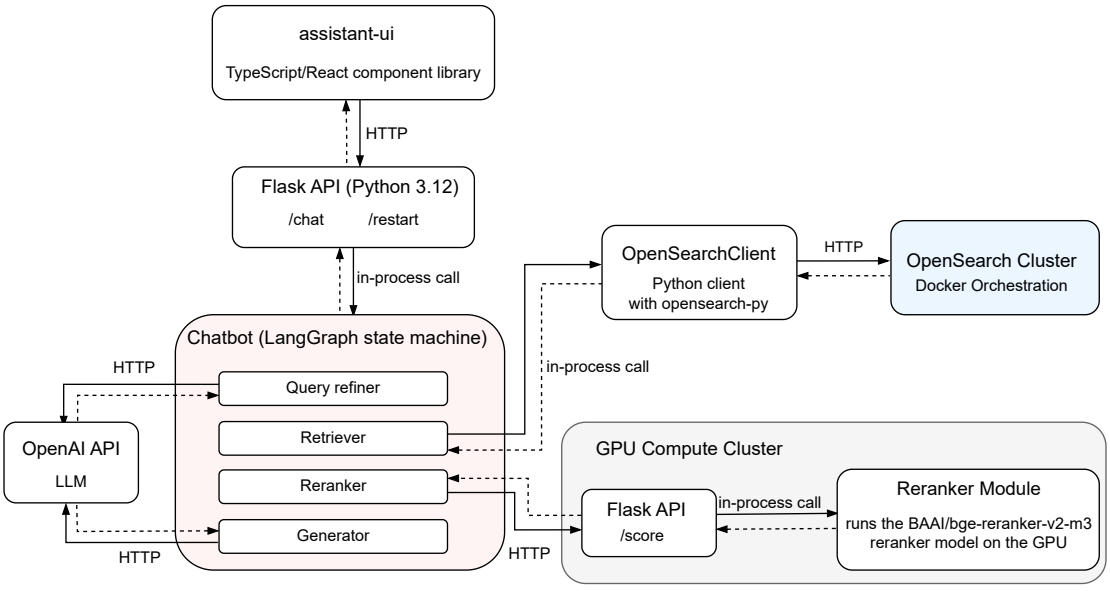

Figure 3.3: Software components overview

Figure 3.3 shows a diagram of the software components of the application. The independent components and external services (OpenSearch cluster in Docker, a reranker service hosted on a GPU compute cluster, and the OpenAI API endpoint) highlight the modularity of the architecture. The OpenAI API offers access to many high-performance models without requiring complex infrastructure and allows for rapid integration. The "GPT-4o mini" model was used during development and testing, but can be seamlessly replaced. Interactions between units are limited to HTTP requests and in-process Python calls, and components can be optimized, scaled, or replaced without disrupting the rest of the system, making this a modular, drop-in architecture.

[1] **LangGraph** is a Python framework for building AI agents https://www.langchain.com/langgraph
[2] **LangChain** is a Python framework for building LLM-based applications https://www.langchain.com/
[3] https://platform.openai.com/docs

3.1.3 Application User Interface

To provide an interactive user interface, I used the open-source TypeScript and React library assistant-ui[1]. It provides easy-to-use implementations that mimic ChatGPT's popular interface for a familiar playground. Flask[2] was used as a backend service to expose the chatbot over HTTP endpoints.

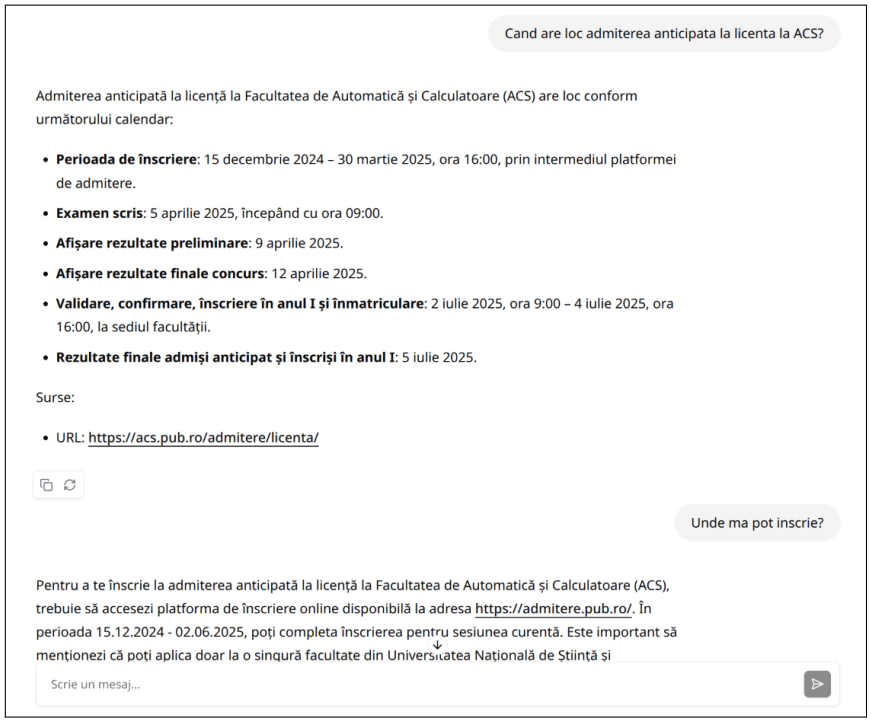

Figure 3.4: Chatbot user interface based on assistant-ui[1]

3.2 Data Collection

The performance of a RAG chatbot depends the data it has access to, according to its intended use case. This section provides an overview of the data collection process for the proposed RAG chatbot for organizational knowledge.

3.2.1 Implementation Details

In order to collect up-to-date organizational knowledge the Scrapy[3] framework was used to build a crawler capable of downloading PDF documents and extracting textual information from web pages. Consequently, a corpus of documents was compiled containing general rules and regulations, admission methodologies, as well as information from FAQ pages.

The crawler loads a list of manually defined URLs from a JSON file (see Appendix A), visits each URL, scrapes page contents and downloads *.pdf* files. The contents of the web pages are processed using the Trafilatura[4] library to extract the meaningful content from the HTML code, such as the title, body text, and dates, filtering out irrelevant information (navigation menus

[1] https://github.com/assistant-ui/assistant-ui
[2] **Flask** is a Python framework for building web applications https://flask.palletsprojects.com/en/stable/
[3] **Scrapy** is a web crawling framework https://docs.scrapy.org/en/latest/
[4] **Trafilatura** is a Python text processing library https://trafilatura.readthedocs.io/en/latest/

or widgets). The text from the web pages is saved as *.txt* files. Metadata about each document is saved to a *metadata.json* file with the following format:

```
[
 {
    "title": "Regulament_admitere_licenta_2025.pdf",
    "path": "Regulament_admitere_licenta_2025.pdf",
    "url": "https://admitere.pub.ro/docs/Regulament_admitere_licenta_2025.pdf",
    "filetype": "pdf",
    "tag": "[ACS REGULAMENT ADMITERE LICENTA 2025]"
 }
]
```

Listing 3.1: Metadata schema example for scraped documents

Metadata is important for tracking the source URL of each piece of information to support real-time fact checking. Fields such as `title` and `filetype` make it easier to filter information, allowing data management by enabling the identification of duplicate files.

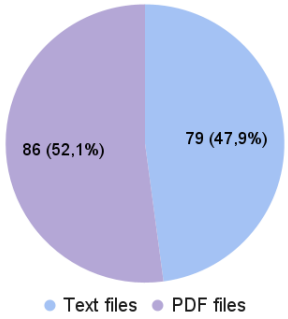

Figure 3.5: Distribution of file types in the collected corpus

After a manual examination of the target web pages, I discovered that documents such as admission methodologies that were updated and published annually were left accessible on the websites alongside the latest editions and not explicitly marked as outdated. In order to avoid downloading outdated PDFs, a "cutoff date" filtering mechanism was added, to only download documents last modified after a `TARGET_DATE`, a configurable parameter that was set as October 1^{st}, 2024. Some older documents were added manually, such as the ones containing general rules and regulations. The distribution of file types is shown in Figure 3.5.

3.2.2 Ethical Considerations

The data used for this project consists exclusively of publicly available documents and information and was collected and used for educational purposes. The proposed solution is designed to be easily adapted to use any other data sources, either privately owned or publicly available.

Upholding the ethical considerations of data sources is very important. To ensure the compliance with such standards, I configured the crawler to comply with the contents of the *robots.txt* file, a publicly accesible file on most websites that includes rules and instructions for crawlers about allowed resources[1]. This is a web standard designed to protect sections of a website from misuse by such programs.

One of the most important objectives of this project is to provide a solution to increase transparency by citing the source of the data used in the generation step of the chatbot. This allows for real-time, manual fact-checking to identify and disregard possible LLM hallucinations.

[1] https://developers.google.com/search/docs/crawling-indexing/robots/intro

3.3 Data Processing

After the data collection step presented in the previous section, the corpus of documents includes both *.pdf* files (downloaded from the target websites) and *.txt* files (contents of the scraped web pages). Each file is passed through a data processing pipeline consisting of two steps: cleanup and chunking. For PDF documents, an additional step is performed prior to these - partitioning.

3.3.1 PDF Partitioning

The first step in the PDF processing pipeline is the partitioning of each document into a list of structured elements such as `Title`, `ListItem` or `Table`, using the Unstructured[1] library's `partition_pdf` function. Many of the collected documents contain both textual and tabular data, with Figure 3.6 showing that 25% of the chunks in the index consist of tabular data. This is a significant portion, therefore it is important to process and extract the information available from tables, and the Unstructured library provides solutions for this.

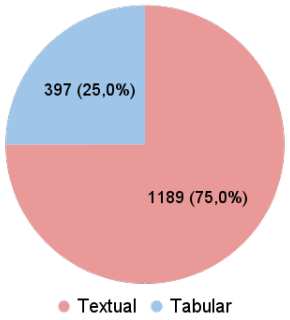

Figure 3.6: Distribution of textual versus tabular chunks in the index

The pipeline of the `partition_pdf` function begins with an analysis of the layout of each page of the document, processed as an image. For this, the high-resolution layout detection model YOLOX[2] is used, identifying text blocks, tables and images. Introduced in [17] as a high-performance object detector, YOLOX detects and classifies regions in a page, outputting a set of bounding boxes with labels indicating the category of each region (e.g. Title, NarrativeText, ListItem, Table). To illustrate the results of the layout detection model used, a two page PDF document was chosen from the ones collected in Section 3.2. Figure 3.9 shows the output bounding boxes and labels of the YOLOX model, given the document.

After the layout analysis yields these labeled bounding boxes, the contents of each region are extracted. For textual elements, if the document has an underlying text layer, direct text extraction is performed. If extractable text is not already available, Optical Character Recognition (OCR) is applied. Unstructured uses Tesseract[3] as the default engine for text extraction. Tesseract is an OCR engine designed to transform printed or handwritten text into machine-readable format. The Tesseract architecture, as described in [18], includes several steps. It first performs text line detection, splitting the image into lines of text which are then broken into individual words using spacing heuristics. Character classification is performed using a trained neural network and spelling is corrected by applying a dictionary lookup. For this particular use case, because the documents are in Romanian, the `tesseract-ron` data for

[1] **Unstructured** is a library that provides pre-processing tools for many document formats and images https://docs.unstructured.io/

[2] **YOLOX** is a document layout analysis model https://huggingface.co/unstructuredio/yolo_x_layout

[3] **Tesseract** is an open-source OCR engine that can recognize 100+ languages https://github.com/tesseract-ocr/tesseract

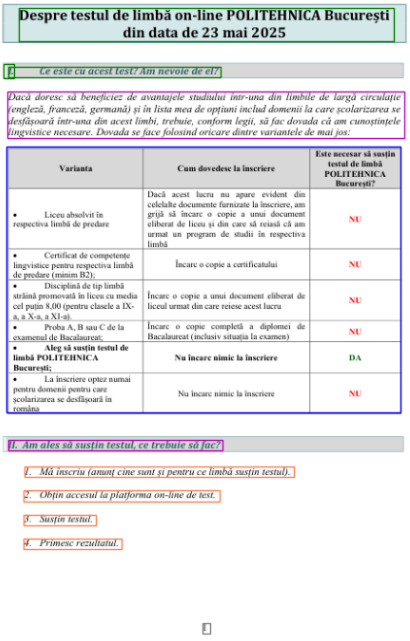

Figure 3.7: Page 1

Figure 3.8: Page 2

Figure 3.9: Visualisation of the YOLOX output bounding boxes with labels on PDF document[1]

the Romanian language were used.

The table detection process requires an additional step, in which the library uses heuristics or a specialized table recognition model (the exact model is not specified in the documentation) to identify the table structure. It uses information about the position of rows and columns to parse the content of the table into HTML format, with elements `<tr>` and `<td>` that preserve the grid structure.

3.3.2 Cleanup

The cleanup of textual content involves whitespace normalization, encoding standardization, and stripping of unwanted punctuation and symbols. Additionally, for tabular elements extracted from PDF documents, the HTML format is converted to markdown, for a cleaner and easier to process format.

3.3.3 Chunking

For chunking the extracted text, I used the spaCy[2] library, specifically the Romanian language trained pipeline `ro_core_news_lg`[3]. The pipeline performs sentence segmentation, using pretrained neural networks to predict whether a token should be labeled as a sentence beginning. The use of neural networks overcomes the limitation of detecting sentence boundaries using predefined rules. For example, following the simple rule that most sentences end with the "." punctuation mark fails when encountering an abbreviation, such as "e.g.".

[1] PDF sourced from https://fils.upb.ro/wp-content/uploads/2025/02/Test-limba-straina-23-05-detalii-S1-2025.pdf

[2] **SpaCy** is an open source library for Natural Language Processing https://spacy.io/api/doc

[3] https://spacy.io/models/ro#ro_core_news_lg

The identified sentences are used to split the text into chunks, taking into account the maximum sequence length (512 tokens) of the embedding model used, detailed in Section 3.4. Any text exceeding this context window will get truncated, leading to information loss.

Listing 3.2 shows the format of a resulted chunk which is ingested in the document index, as described in Section 3.4. The example shown represents the first chunk of text extracted from the document presented in Figure 3.9. Each chunk is represented as a `JSON` object, with an assigned unique `id`, computed as `<url_hash>-<chunk_number>`. The field `text` contains the extracted text which is provided as context to the LLM. Fields `url` and `page_number` represent the metadata designed to provide the source of the information used in the response generation process, enabling fact-checking by the user, as a measure of identifying factually unsupported content. Field `table_text` holds the markdown-formatted table processed from the document. Given that the contents of a table may exceed the maximum chunk size, it may be split into multiple chunks, which are all assigned the same `table_id`.

```
{
  "id": "4a8af42a6c47f11b1d13d802bee4da49f7d449950fa3f74b338baa449e1e595b-1",
  "text": "Despre testul de limbă on-line POLITEHNICA Bucureşti din data de 23 mai
      2025 L Ce este cu acest test? Am nevoie de el? Dacă doresc să beneficiez de
      avantajele studiului într-una din limbile de largă circulaţie (engleză,
      franceză, germană) şi în lista mea de opţiuni includ domenii la care
      şcolarizarea se desfăşoară într-una din acest limbi, trebuie, conform legii, să
       fac dovada că am cunoştinţele lingvistice necesare. Dovada se face folosind
      oricare dintre variantele de mai jos:",
  "url": "https://fils.upb.ro/wp-content/uploads/2025/02/Test-limba-straina-23-05-
      detalii-S1-2025.pdf",
  "type": "pdf",
  "filename": "Test-limba-straina-23-05-detalii-S1-2025.pdf",
  "page_number": 1,
  "table_id": null,
  "table_text": null
}
```

Listing 3.2: Example chunk from document in Figure 3.9

3.4 Indexing

Indexing refers to the process of organizing and storing a corpus of documents optimized for fast search and retrieval based on semantic similarity to a search query. Traditional relational databases store structured information and are designed to retrieve exact matches or perform rule-based queries. To enable semantic search, documents are split into chunks, which are then encoded into high-dimensional numerical vector representations known as embeddings, designed to capture semantic meaning and relationships between words, as illustrated in Figure 3.10. These embeddings are typically stored in vector databases or vector search engines, which are databases designed to store and search for this type of data.

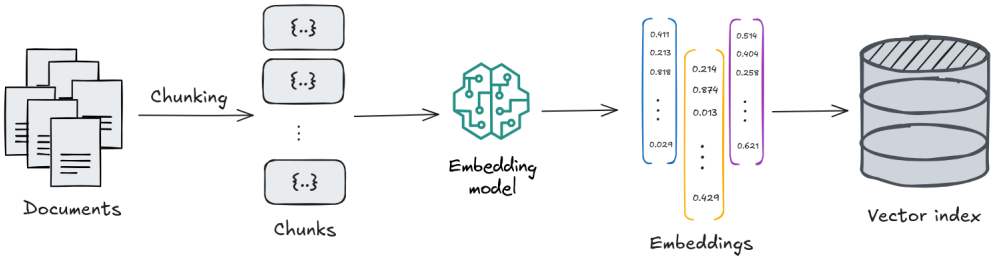

Figure 3.10: Overview of the embedding process

3.4.1 OpenSearch Motivation

To provide efficient and scalable semantic search, OpenSearch[1] was chosen as the underlying search engine. As opposed to other popular vector databases, such as FAISS[2] or Pinecone[3], which only provide vector search capabilities, OpenSearch provides both vector search and full-text search, a hybrid approach integrated into the same platform, making it a more versatile solution. Additionally, it was designed to easily scale horizontally, by adding additional nodes to the cluster, ensuring high availability. It is an open-source solution which may be used both locally or hosted in the cloud and it offers integrations with multiple existing NLP libraries.

3.4.2 OpenSearch Cluster Architecture

The proposed solution uses Docker containers to run OpenSearch locally as a multi-node cluster. The cluster architecture shown in Figure 3.11 is composed of two data nodes responsible for storing, searching, and performing all operations on the data, and one OpenSearch Dashboards[4] node that queries both data nodes and provides a user interface to analyze and provide visual representations of the data. One of the data nodes acts as a cluster manager, orchestrating all cluster-level operations, such as creating and deleting indexes. The nodes communicate over an internal Docker network. A custom Python OpenSearchClient provides an interface that allows communication with the cluster manager over HTTP, using the opensearch-py[5] client.

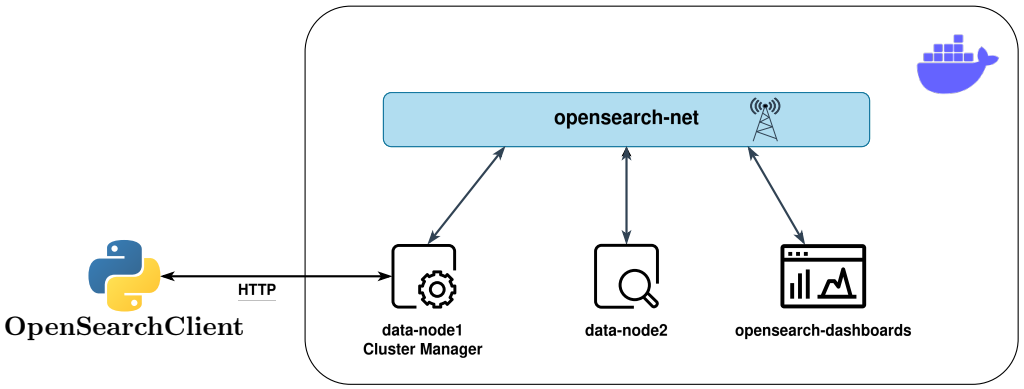

Figure 3.11: OpenSearch cluster architecture

OpenSearch is built upon Apache Lucene[6], a powerful search library that provides an implementation of the Hierarchical Navigable Small World (HNSW) graph data structure. An OpenSearch index is used to store and organize a corpus of documents. Indexes may be stored across multiple nodes, dividing responsibilities in multi-node clusters to increase performance. OpenSearch provides a redundancy mechanism by replicating indexes by default.

The chunked documents obtained after the processing pipeline described in Section 3.3 are ingested into OpenSearch using an ingest pipeline containing a text embedding processor.

[1] **OpenSearch** is an open source search engine https://docs.opensearch.org/docs/latest
[2] **Facebook AI Similarity Search (FAISS)** is an open source vector database https://faiss.ai/
[3] **Pinecone** is a cloud-native vector database https://docs.pinecone.io/
[4] **OpenSearch Dashboards** provides a UI for cluster data visualization and scaling https://docs.opensearch.org/docs/latest/dashboards/
[5] **opensearch-py** is a Python client for OpenSearch https://opensearch-project.github.io/opensearch-py/
[6] **Apache Lucene** is a high-performance search library https://lucene.apache.org/core/documentation.html

3.4.3 Text Embedding Processor

The text embedding processor uses the pre-trained Sentence Transformers[1] model paraphrase-multilingual-MiniLM-L12-v2[2] to compute embeddings for each chunk of text at ingestion time. This model is included in the list of pre-trained models supported by OpenSearch, making it easy to integrate and use it locally in the cluster. It is a multilingual model, with support for over 50 languages, including Romanian, making it well suited for this specific task. Based on the Sentence-BERT architecture described in Section 2.5.3, the model generates text embeddings in a 384-dimensional dense vector space, minimizing the distance between vectors with similar semantic meaning. It is significantly more lightweight than other alternatives, offering a trade-off between representation quality and scalability.

The paraphrase-multilingual-MiniLM-L12-v2 model is a multilingual version of paraphrase-MiniLM-L12-v2[3], which was optimized to produce sentence embeddings for tasks such as paraphrase detection, semantic similarity, and clustering. The base model is trained on datasets of `(anchor, positive, negative)` sentence triplets, using contrastive learning techniques to learn representations by minimizing the distance between positive (similar) pairs of sentences and maximizing the distance between negative (dissimilar) pairs. The multilingual model was obtained by applying multilingual knowledge distillation [19], a technique shown in Figure 3.12 that involves a teacher model, trained for English, a student model, and a set of translated sentences.

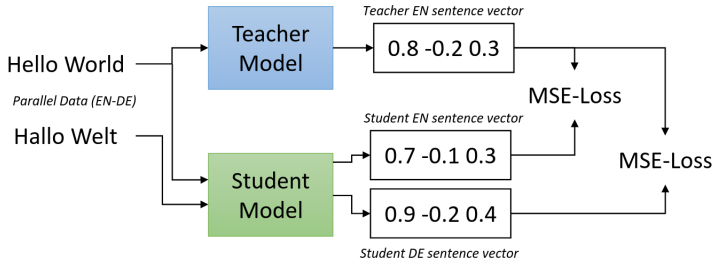

Figure 3.12: Multilingual knowledge distillation overview [19]

The student model (paraphrase-multilingual-MiniLM-L12-v2) is trained to minimize the mean squared loss between the output representations of the teacher (paraphrase-MiniLM-L12-v2) and its English and non-English representations. This is a flexible and scalable approach to learn a multilingual embedding space, in which similar sentences are located closer together.

3.4.4 Index Structure

To support semantic search, Apache Lucene provides a high-performance implementation of the HNSW data structure. This multi-layer graph-based index is designed for fast approximate k-nearest-neighbor (KNN) search. Each node in the graph corresponds to a vector embedding and may appear in multiple layers. Higher layers contain fewer nodes, more sparsely arranged, whereas lower layers contain more nodes, more densely grouped for a more precise search.

The index used to store the chunked documents was designed to encapsulate textual contents, metadata for filtering and vector embeddings to support semantic search. An overview of the fields of the index configuration is presented in Table 3.1.

[1] **Sentence Transformers** is a Python framework for NLP embeddings generation https://huggingface.co/sentence-transformers

[2] https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2

[3] https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L12-v2

Field	Type	Description
id	`text`	Unique identifier for the document passage.
embedding	`knn_vector`	384-dimensional vector embedding generated at ingestion time.
text	`text`	Textual contents of the document chunk.
url	`keyword`	Source URL of the document.
type	`keyword`	Type of the document (e.g. PDF).
filename	`keyword`	Name of the document.
page_number	`integer`	Page number (if the document is paginated).
table_id	`text`	Unique identifier for the tabular content (if applicable).
table_text	`text`	Markdown contents of the table (if applicable).

Table 3.1: OpenSearch index fields mapping

Documents are ingested in batches using the Bulk API[1], increasing performance by queuing multiple indexing operations into a single request, reducing overhead by limiting the number of requests performed.

3.5 Query Refiner

Interactions with chatbots are not limited to standalone questions, with users typically asking multiple questions, some of which may contain expressions with interpretations dependent on the conversational history. In order to allow the retrieval-augmented generation process to incorporate context from prior messages in the conversation, a *Query Refiner* component is introduced. The design draws inspiration from the mechanism described in [20] that uses a question reformulation strategy to rephrase the original user query into a *refined*, more explicit version. This procedure seeks to enable the use of RAG with follow-up questions.

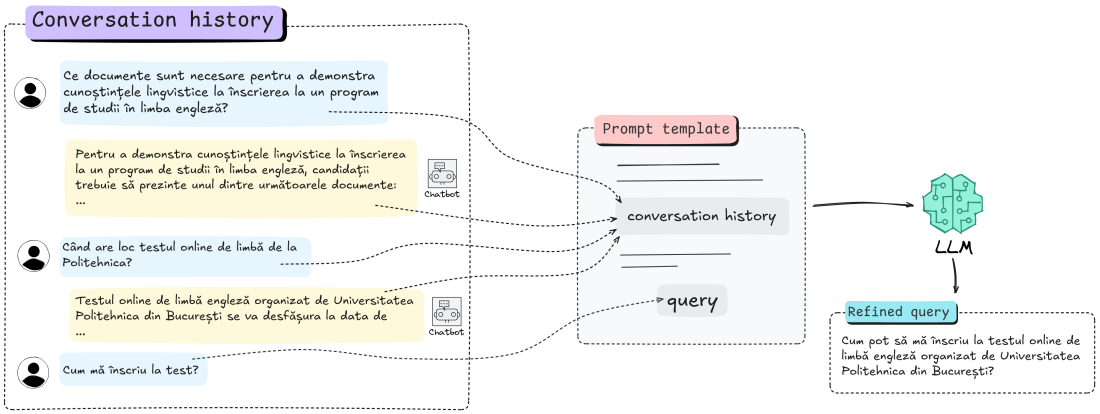

Figure 3.13: Query refiner overview

The *Query Refiner* component constructs a prompt containing the last n messages from the conversational history (see Figure 3.14, Appendix B), and inputs it to an LLM, the same model used for the generation, leveraging the model's pre-trained parametric memory to obtain a rephrased standalone question that contains context from the previous messages. This aims to eliminate possible ambiguities, allowing the refined question to be used in the retrieval process.

A common approach to enable follow-up questions is to append the conversation history directly to the LLM prompt to preserve the conversational context. However, this naive strategy

[1] https://docs.opensearch.org/docs/latest/api-reference/document-apis/bulk/

introduces several limitations, such as the increasing number of tokens in the input, with the prompt containing both the retrieved passages and the conversation history. The *Query Refiner* is designed as a separate component that provides an explicit query to enable a better retrieval.

```
"""
Ești un asistent specializat în reformularea întrebărilor din conversații.
Analizează întrebarea utilizatorului și conversația anterioară pentru a înțelege contextul.Identifică ambiguitățile și reformulează
întrebarea astfel încât să fie complet clară și înțeleasă fără a fi nevoie de context suplimentar.
Păstrează cât mai mult din structura și formularea originală, dar adaugă explicit detaliile și informațiile contextuale relevante.

Contextul conversației:
{conversation_history}

Întrebarea:
{query_text}
"""
```

Figure 3.14: Query refiner prompt template

3.6 Retrieval

The retrieval component is based on the Dense Passage Retrieval (DPR) architecture, introduced in [21] as a function that takes as arguments a sequence of text, query q, and a corpus of document passages $\mathcal{C}$ and outputs a k-dimensional subset of passages $\mathcal{C}_k$, with $|\mathcal{C}_k| = k$.

The first step to implementing the DPR is indexing the corpus of document passages in a low-dimensional vector space using an embedding model, presented in Section 3.4.3. The same embedding model is used at inference time to compute an embedding for the user query. The retriever component is designed to fetch the top k most similar passages to the given user query, which represens a Maximum Inner Product Search (MIPS) problem over the embedding space.

3.6.1 Vector Similarity

In order to retrieve the top k most similar passages to a given query, the retriever must use a measure to quantify the similarity between the embedding of the query E_q and the embedding E_p of each document passage p in the corpus $\mathcal{C}$. The distance metric used for calculating the similarity between vector embeddings in the OpenSearch index is cosine similarity, defined as:

$$cosine\_similarity(E_q, E_p) = cos\theta = \frac{E_q \cdot E_p}{||E_q|| \cdot ||E_p||}, \tag{3.1}$$

where E_q, E_p are the vector embeddings, θ is the angle between the vectors, $||E|| = \sqrt{\sum_{i=1}^{d} E_i^2}$ is the Eucledian norm of vector E, and d is the embedding dimension, in this case $d = 384$.

Cosine similarity was chosen over other functions because it uses the angle between the vector embeddings, which is more meaningful than the magnitude for text embeddings. Vectors corresponding to similar passages are located closer to each other in the embedding space, therefore a smaller angle θ corresponds to a higher cosine value $cos\theta$, yielding a higher similarity score.

3.6.2 Approximate K-Nearest Neighbor Search

OpenSearch supports semantic approximate k-nearest-neighbor search, using the HNSW graph-based index structure presented in Section 3.4.4. The search algorithm is used to find the top k most relevant passages to a user query, providing a higher performance than exact KNN search, significantly faster and more scalable, trading off search accuracy.

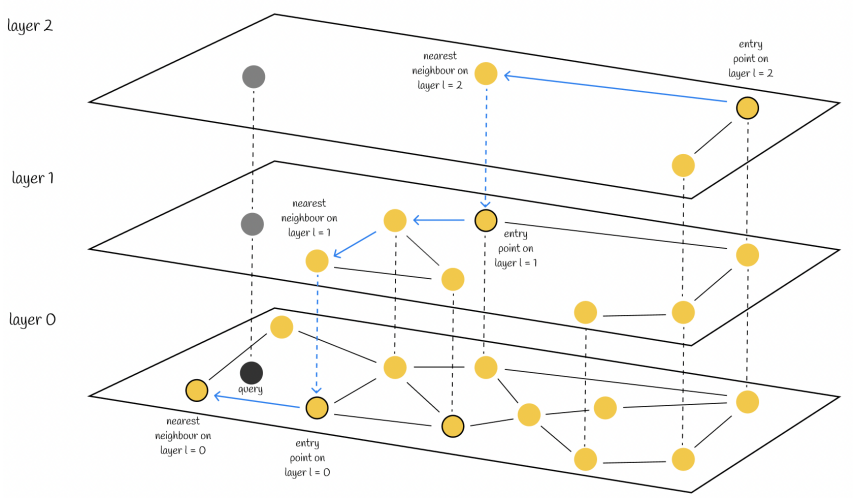

Figure 3.15: HNSW search visualization[1]

Figure 3.15 shows a visual representation of the approximate KNN search on a HNSW multi-layer graph index, a functionality provided natively by OpenSearch. The nodes correspond to the vector embeddings, and each edge in a layer links two nearest nodes. The search starts on the highest layer, with an entry point vector. A greedy search is then performed to find the nearest neighbors in the current layer. Once a local minimum is found, it is used as the new entry point, and the algorithm is recursively applied on the descending layers. The lower layers allow for a more precise search, with more densely linked nodes. The final k nearest neighbors are selected from the lowest layer.

3.7 Reranking

The dense passage retrieval step of the RAG pipeline is based on an embedding model that encodes text into high-dimensional continuous vector representations. Naturally, this process is prone to information loss, as the model aims to incorporate all of the meaning of text passages into fixed-size numerical representations. Despite the speed and scalability of the optimized search methods used, such retrieval systems have the limitation that the query and the passages are encoded independently, and search results may not be contextually relevant.

One way to address this limitation is to implement a two-stage retrieval process, with an additional reranking step designed to generate similarity scores of higher fidelity. This approach consists of a retriever that uses a bi-encoder model, followed by a reranker that uses a cross-encoder model [22].

3.7.1 Architecture Overview: Cross-Encoder versus Bi-Encoder

The retrieval step described in Section 3.6 is based on a bi-encoder model (see Figure 3.16) that uses two instances of a SBERT model to compute vector embeddings individually at ingestion time. This step is optimized for fast retrieval of the k most similar passages to a given query, based on the cosine similarity of the pre-computed embeddings.

[1] https://medium.com/data-science/similarity-search-part-4-hierarchical-navigable-small-world-hnsw-2aad4fe87d37

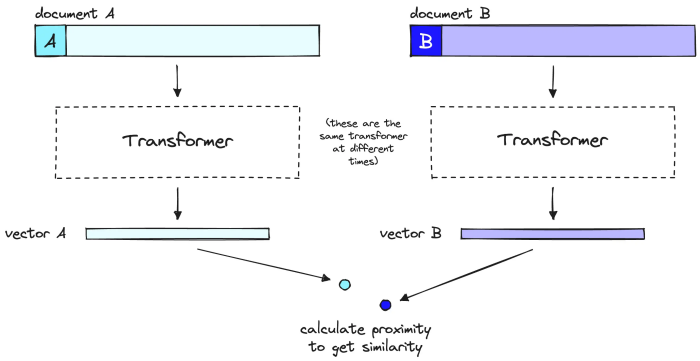

Figure 3.16: Bi-encoder model overview[1]

The reranker uses the cross-encoder model architecture [16] shown in Figure 3.17 to compute a similarity score for each `(query,document)` pair, to reorder the retrieved documents, analyzing them with additional context given by the user query. The query and document are concatenated using a classification token and a separator token, to form the input sequence "$[CLS]$ *query* $[SEP]$ *document*" which is passed to the BERT model to compute a similarity score mapped to the rage $[0, 1]$ by a sigmoid function.

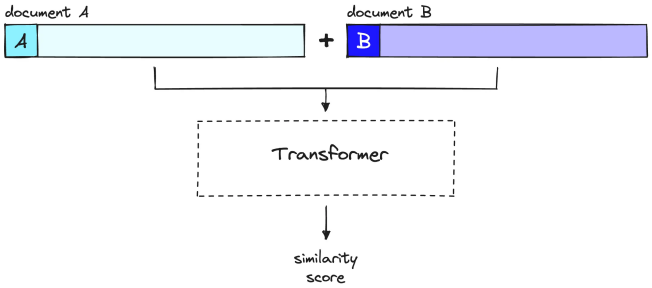

Figure 3.17: Cross-encoder model overview[1]

3.7.2 Model Configuration

The proposed solution uses the open-source BAAI bge-reranker-v2-m3[2] cross-encoder model, a lightweight multilingual reranker based on BGE-m3 [23]. The BGE-m3 model is an encoder trained so that one set of model weights supports three different functionalities (dense retrieval, sparse retrieval, and multi-vector retrieval) by using self-knowledge distillation, an approach in which relevance scores from multiple retrieval paradigms are integrated as teacher signals in a weighted sum. The training objective is to predict the weighed ensamble score, yielding a single model that provides all of the individual capabilities. The reranker is fine-tuned on multiple multilingual datasets to support over 100 languages, including Romanian, with a maximum input length of 1024 tokens. Having around 568 million parameters, with a size of 2.27GB, it is a lightweight model that can comfortably run on a single GPU on the university's compute cluster, which was set up as a Flask endpoint in order to be easily integrated into the pipeline. The FlagEmbedding[3] Python library was used to load and use the model from Hugging Face[4].

[1] https://www.pinecone.io/learn/series/rag/rerankers/
[2] https://huggingface.co/BAAI/bge-reranker-v2-m3
[3] https://github.com/FlagOpen/FlagEmbedding
[4] https://huggingface.co/

3.8 Generation

The generator component is responsible for producing answers to user queries that incorporate information from the retrieved context. It leverages the pre-trained large language models from the GPT family via the OpenAI API[1]. As described in Section 2.5.1, GPT models are well-suited for text generation and have shown strong performance in many different tasks, without the need of specialized prompting [24]. The purpose of this RAG system is to combine the extensive knowledge encoded in the parametric memory of LLMs during pre-training with the non-parametric memory provided by the retrieved documents incorporated into the prompt (see Appendix C) to enable contextually grounded responses.

Although the use of an LLM through a hosted API involves additional costs, this approach was preferred due to fast prototyping, rapid integration, and lack of need for advanced infrastructure. Adaptability is one of the main features of this system, and due to the fact that the model is accessed via an external API, it can be easily changed, or upgraded to different models.

In order to prioritize cost-efficiency and speed, the GPT-4o mini[2] model was chosen as the default large language model used in the generator component. This choice is easily adjustable, requiring no infrastructure reconfiguration and minimal changes to adapt to new requirements. The price of $0.15 per million input tokens places it among the more cost-effective options offered by OpenAI, offering a good balance between cost and performance. This makes it well-suited for a real-time chatbot application, that does not require complex reasoning. A more detailed list of features is shown in Table 3.2.

Feature	Value	Description
context_window	128,000 tokens	Maximum number of tokens in the input sequence.
max_output	16,384 tokens	Maximum number of tokens generated in one output sequence.
pricing_input	$0.15 per million tokens	Input cost.
pricing_output	$0.60 per million tokens	Output cost.
multimodal	Yes	Supports both text and images as input.
release_date	July 2024	Initial availability.
knowledge_cutoff	Oct 2023	The date at which the model's pre-training data ends.

Table 3.2: Key features of the GPT-4o Mini model[3]

3.8.1 Structured Output Validation

The Pydantic[4] library was used to validate the schema of the generated response. By ensuring that the answer follows a predefined structure (the LLMResponse object shown in Figure 3.18), it can be verified that the response includes source URLs.

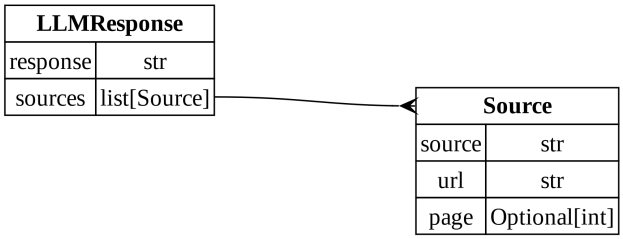

Figure 3.18: Diagram of the LLMResponse object used for schema validation

[1] https://openai.com/api/
[2] https://platform.openai.com/docs/models/gpt-4o-mini
[3] https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/
[4] https://docs.pydantic.dev/latest/

If the answer does not include a list of sources, the chatbot disregards it and instead produces a standard response to state that the information in the retrieved context was not sufficient to provide a grounded response.

3.8.2 Prompt Design and Structure

The prompt design aims to guide the language model toward generating structured, relevant and grounded responses, based on the provided context. The following recommended techniques [25] were used to shape the response generation prompt (see Appendix C), as well as the prompts used in the other application components:

- define a clear persona that the model should adopt
  Example: *"You are an assistant that answers questions based on the context from relevant documents."*
- include explicit instructions to clearly state the expected behaviour
  Example: *"Generate a formal response based on the received data. Consider the entire question, do not ignore details or parts of the question."*
- require the model to provide source attribution for all claims
  Example: *"For each statement, explicitly provide the source extracted from the document's metadata."*
- use delimiters to split the task into clear, distinct parts
  Example:
  "### Query:
  {query}

  ### Context:
  {context} "
- specify a well-defined structure the desired output
  Example: *"Format the response as JSON as follows: ..."*

4 Experiments and Results

The evaluation of a RAG system proves to be elaborate, with multiple aspects to consider: the retriever's ability to fetch documents relevant to the user query, the reranker's ability to reorder the retrieved passages, in a way that prioritizes the most useful and relevant passages, as well as the generative quality of the LLM.

Traditional metrics such as the F1 Score, commonly used in NLP tasks like text summarization or machine translation, require a reference-based evaluation method. Due to a lack of an annotated dataset containing ground truth answers and reference relevant contexts, an alternative evaluation approach must be adopted. Although this may be regarded as somewhat subjective, it is important to consider the nature of the task which involves open-ended question answering within the scope of a given corpus. This requires an understanding of the question and context, so an answer, although it should be grounded in the provided documents, does not have a single fixed correct representation. The proposed evaluation method represents a useful first step in obtaining an automated method to test the behavior of the system.

Although human evaluation may be considered a "gold standard" [26] for the evaluation of chatbot systems, this method is time-consuming, expensive, and not easily scalable for iterative workflows. To address the need for automated evaluation within the limitations of available computational resources and to provide metrics that quantify the performance of the system and allow further grounded development, the "LLM-as-a-judge" method was implemented. Human preference represents a meaningful indicator of the utility of a chatbot system, and the judge LLM approach, specifically using GPT-4[1] as a judge, has shown consistency with human assessments, surpassing 80% agreement with human evaluators [26]. This agreement score was computed using two different benchmarks: MT-Bench (consisting of 80 multi-turn questions) and Chatbot Arena (a platform where users compare two chatbot responses and vote for the preferred one), evaluating the agreement of LLM judges with expert human labelers on approximately 3000 votes for each benchmark independently.

The proposed automated evaluation method uses the single answer grading variant of the LLM-as-a-judge technique [26], prompting an LLM (GPT-4.1[2]) to grade a given answer or retrieved contexts, based on predefined criteria. A small amount of data is used to evaluate the system on the intended use-case, rather than large existing labeled datasets that are unrepresentative of the application scenario. This automated evaluation was designed for single-turn conversations, analyzing the system behaviour without the complexity of multi-turn context.

4.1 Multi-turn Conversation Example

To illustrate a multi-turn conversation example, Figure 4.1, Figure 4.2 and Figure 4.3 show snapshots from the chatbot user interface, with a series of follow-up questions. Each response contains a list of sources, with URLs and page numbers from which the information used to formulate the answer was extracted.

[1] https://openai.com/index/gpt-4/
[2] https://platform.openai.com/docs/models/gpt-4.1

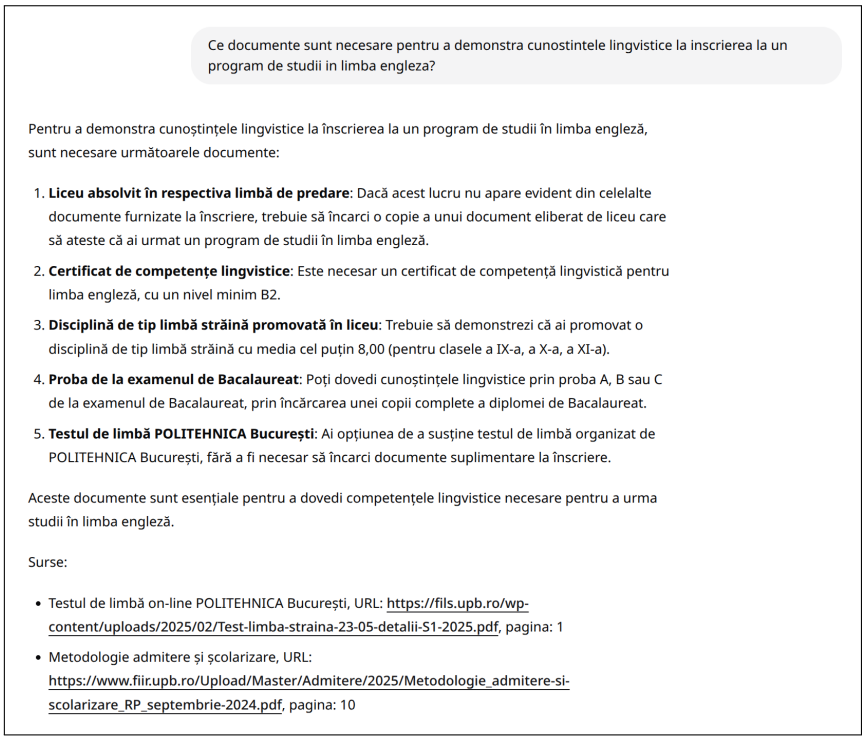

Figure 4.1: Initial query and chatbot response

The initial query is not affected by the query refiner component, given that there is no conversational history, and no additional information is provided to clarify potential ambiguities.

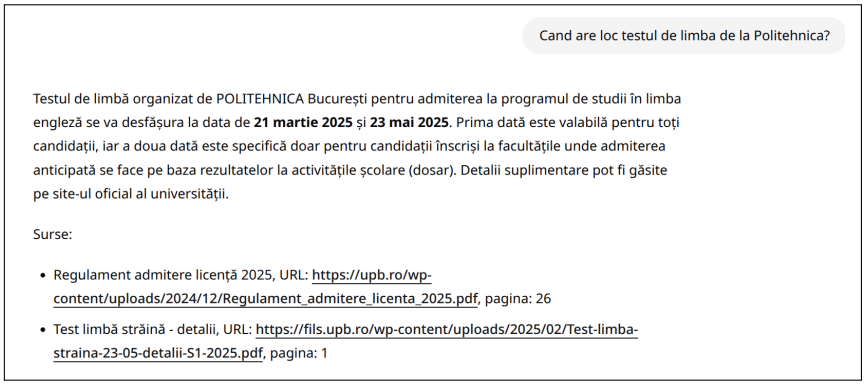

Figure 4.2: Follow-up question based on the conversation history from Figure 4.1
Original query: *"Cand are loc testul de limba de la Politehnica?"*
Refined query using the query refiner component: *"Când se va desfășura testul de limbă organizat de POLITEHNICA București pentru admiterea la programul de studii în limba engleză?"*

The first follow-up question shown in Figure 4.2 is reformulated using the query refiner component, based on the user and chatbot messages from Figure 4.1. The refined query aims to provide a more clear version of the original query, that may be understood without additional context. In this case, detailed information from the previous messages are included in the query, to enable a more contextually relevant retrieval process.

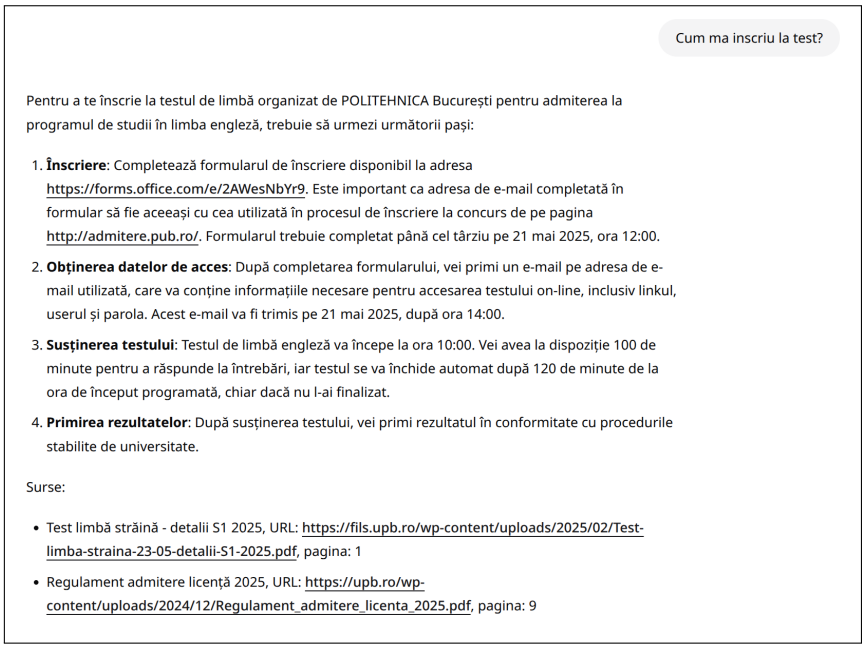

Figure 4.3: Follow-up question based on the conversation from Figure 4.1 and Figure 4.2
Original query: *"Cum ma inscriu la test?"*
Refined query using the query refiner component: *"Cum pot să mă înscriu la testul de limbă organizat de POLITEHNICA București pentru admiterea la programul de studii în limba engleză?"*

Manual experiments showed that without using the query refiner component to reformulate follow-up questions based on previous messages, the retrieved documents were not as relevant to the full conversational context, likely due to the lack of explicit information in the queries.

4.2 Test Data Generation

4.2.1 Initial Approach and Limitations

The first evaluation attempt was to generate a synthetic Q&A dataset based on a subset of documents in the corpus. The purpose of this was to create a small reference dataset that the system can be tested on. For this, a small portion of the documents was selected, and for each one GPT-4.1 was prompted to generate three `(question, answer, relevant context)` tuples. After a manual review, 20 pairs that were clear, diverse and grounded in the provided context were selected.

This initial approach was, however, disregarded, because it lacked credibility. Given that it relied on a single document per question-answer pair, it led to many false negatives. The main problem was that many retrieved passages were wrongfully penalized and marked as not relevant to the question, due to the fact that they were from documents that were not included in the synthetic dataset generation process. Generating such a dataset requires significant resources, and the limited context window of LLMs restricts their ability to process large corpora of documents at once, making this approach unsuitable.

4.2.2 Clustering-Based Curation from Real Conversations

To construct a set of evaluation questions, I leveraged a dataset comprised of real conversations between users and another chatbot with matching objectives and use-cases. ChatUPB[1] is a similar RAG-based chatbot developed by a team of doctoral researchers from the Faculty of Automatic Control and Computers of UNSTPB, which was presented during the PoliFest[2] educational fair. The underlying knowledge base of this system consists of a significant portion of the documents that I gathered and processed specifically for this application, as described in Section 3.2 and Section 3.3. The examined dataset consists of 530 real conversations with 941 user queries, collected at the event where the chatbot was first showcased.

The analysis performed on the collection of user-chatbot interactions first involved extracting the user inputs from each logged entry. These were then encoded using the sentence-level multilingual embedding model paraphrase-multilingual-MiniLM-L12-v2, as described in Section 3.4.3, to obtain vector representations in a high-dimensional continuous space. Using the scikit-learn[3] AgglomerativeClustering[4] implementation, I performed semantic clustering of the sentence embeddings, grouping similar user queries into the same cluster. This is a hierarchical clustering algorithm, in which all vectors are first assigned to individual clusters, which are then merged based on the cosine distance. A pre-specified number of expected clusters is not necessary, as the clustering stops when a fixed inter-cluster distance is reached. The results of this analysis revealed dense clusters corresponding to frequently asked user questions. After a manual review of the most dense clusters, I formulated a set of 40 questions representative to the real user queries and relevant to the domain. This approach aimed to curate common real-world user queries rather than randomly sample or synthetically generate new ones.

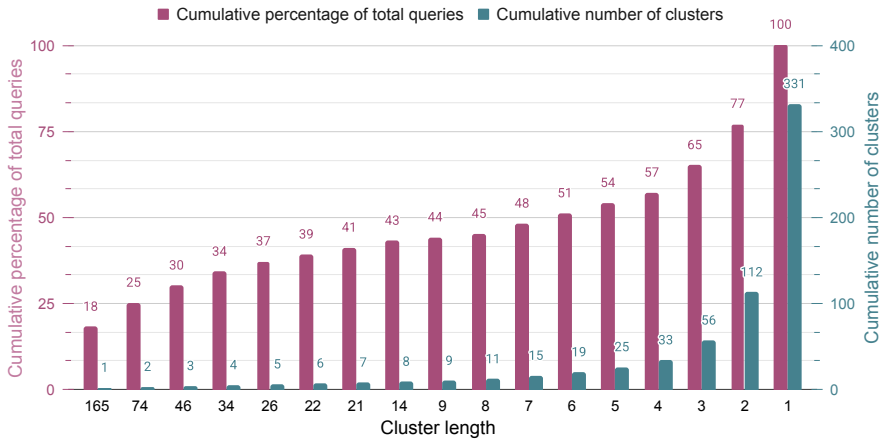

Figure 4.4: Cumulative distribution of queries and clusters by cluster length from the semantic clustering of the conversations collected from ChatUPB[1], used to construct the test queries

Figure 4.4 shows the cumulative query coverage by cluster length, indicating that a small number of the dense clusters account for a large percentage of the total queries examined. This suggests that the test questions are representative of the most common real user questions in the dataset, mainly about admission methodologies, event dates, rules and regulations.

[1] https://chat.upb.ro/
[2] https://polifest.upb.ro/
[3] **scikit-learn** is an open-source ML library for Python https://scikit-learn.org/
[4] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering

4.3 Evaluation

The proposed evaluation process aims to assess the performance of the system with regard to the relevance of the retrieved context to the user query, as well as the final generated output. For this, the LLM-as-a-judge method was employed, using GPT-4.1, which has shown a high agreement with human evaluation [26].

4.3.1 Retrieval Evaluation Metrics

To quantify the performance of the retriever and reranker components, the following metrics were used, as presented in [27]:

1. **MAP@k** - mean average precision score for each query, computed as:

$$MAP@k = \frac{1}{|Q|} \sum_{q=1}^{|Q|} AP_q@k \tag{4.1}$$

where $AP_q@k$ represents the average precision for the query q at cutoff k,

$$AP_q@k = \frac{1}{|relevant\ documents_q|} \sum_{i=1}^{k} Precision_q@i \times is\_relevant_q(i) \tag{4.2}$$

|Q| is the total number of queries, k is the number of retrieved documents for each query, $Precision_q@i$ is the fraction of relevant documents from the first i retrieved for query q, and $is\_relevant_q(i)$ corresponds to a binary value that reflects whether the retrieved document i was judged to be relevant or not to the user query q by the judge LLM.

A higher $MAP@k$ value indicates that the system is consistent in retrieving relevant documents among the top-k results.

2. **MRR@k** - mean reciprocal rank, computed as:

$$MRR@k = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \frac{1}{rank_q} \tag{4.3}$$

where |Q| is the total number of queries and $rank_q$ is the position in which the first relevant document appears in the list of retrieved documents for query q.

A higher $MRR@k$ value indicates that relevant documents appear closer to the top of the retrieval ranked list.

To assign a relevance score to each retrieved document passage relative to the a user query, two different prompt templates (see Appendix D and Appendix E) were used, adapted from RAGAs[1] and translated into Romanian for consistency. Adopting the strategy from the RAGAs framework implementation[2], the two prompts were passed to the same judge LLM to generate a score between $[0-2]$ that indicates whether a document is relevant (score 2), partially relevant (score 1), or not relevant at all (score 0) to a given user query. The final relevance score of a document i to a user query q, denoted as $is\_relevant_q(i)$, is computed as the mean of the two scores normalized to range [0-1], rounded to the nearest integer.

These metrics were chosen as a way to measure what can be described in high-level as *"How relevant are the k retrieved documents to the given user query?"*.

[1] **RAGAs** is an open-source evaluation framework for LLM applications https://docs.ragas.io/
[2] https://docs.ragas.io/en/latest/concepts/metrics/overview/

4.3.2 End-to-end System Evaluation Metrics

To evaluate the end-to-end RAG system, two methods were used, adapted from [28]: GPT Score and RAGAs implementations. For each method, two metrics were computed:

1. **Faithfulness** of a generated answer to the retrieved context.
2. **Answer relevance** to the given user query.

The **GPT Score** method consists of using an LLM (GPT-4.1) prompted with the definitions of the two metrics and evaluation instructions (see Appendix F and Appendix G) to generate scores for faithfulness and answer relevance.

The **RAGAs** scores are computed using the implementations from the RAGAs framework, for the two metrics as follows:

- The RAGAs faithfulness score[1]:

$$Faithfulness = \frac{|claims\ in\ the\ response\ supported\ by\ the\ retrieved\ context|}{|total\ claims\ in\ the\ response|} \tag{4.4}$$

  This measures how consistent a generated answer is to the retrieved context. The RAGAs Faithfulness[1] implementation uses a call to a judge LLM to identify all the claims in the answer and another call to jugde whether each claim can be inferred from the context.

- The RAGAs answer relevance score[2]:

$$Answer\ Relevance = \frac{1}{N}\sum_{i=1}^{N} cosine\_similarity(E_{g_i}, E_q) \tag{4.5}$$

  where N is the number of artificially generated questions g_i that capture the content of the answer to query q, E_{g_i} and E_q are the corresponding vector embeddings, and $cosine\_similarity(E_{g_i}, E_q)$ is the similarity metric used, as described in Section 3.6.1.

  This measures how relevant a generated answer is to the user query, with a score in the range [0-1]. Higher scores correspond to a high alignment with the input query, whereas lower scores may indicate incompleteness or redundancy. The RAGAs Answer Relevance[2] implementation uses a judge LLM to generate three synthetic questions for each answer. To compute the embeddings the same model was used as for the indexing step, paraphrase-multilingual-MiniLM-L12-v2, as described in Section 3.4.3.

For each call to the judge LLM, the RAGAs framework provides predefined prompt templates, and each of them has been adapted for Romanian to ensure consistency, using existing support from the framework.

The RAGAs implementation has shown high agreement with human labelers [28] using the WikEval[3] dataset that contains user queries with corresponding relevant contexts, as well as generated answers and ground truth evaluations from two human annotators.

4.3.3 Results

Reranker ablation study and k-value sweep

Table 4.1 shows the evaluation results on the test dataset described in Section 4.2. High faithfulness scores indicate that the generated responses are consistent to the retrieved context, and high answer relevance scores suggest that the answers address the query. Lower answer

[1] https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/faithfulness/
[2] https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/answer_relevance/
[3] https://huggingface.co/datasets/explodinggradients/WikiEval

relevance values may indicate the fact that the responses are either incomplete, or incorporate unnecessary information that might not directly adress the user input.

				Faithfulness		Answer Relevance	
	k	MAP@k	MRR@k	GPT Score	RAGAs	GPT Score	RAGAs
System witout Reranker	3	0.735	0.725	0.519	0.782	0.48	0.383
	5	0.691	0.702	0.567	0.818	0.497	0.359
	7	0.715	0.732	0.597	0.781	0.602	0.472
Complete system	3	**0.827**	0.841	0.637	0.739	0.627	0.494
	5	0.825	**0.85**	**0.842**	**0.864**	**0.792**	**0.615**
	7	0.815	0.847	0.727	0.812	0.705	0.571

Table 4.1: Evaluation results for the chatbot system with and without the reranker component across three different retrieval dimensions $k = \{3, 5, 7\}$

This evaluation was conducted as a reranker ablation study by comparing the performance of the chatbot system with and without the reranker across three different retrieval sizes $k = \{3, 5, 7\}$. For the complete system, the retriever fetched 15 documents, which were reordered by the reranker and limited to the top k. For each k value, the results are significantly higher for all metrics when using the complete system with the reranker, suggesting that this component is valuable and consistently contributes to improved response quality.

For the k-value sweep, analyzing the responses reveals a trade-off in choosing the number of documents to be retrieved. A lower k value may not provide sufficient information to generate a faithful and relevant response, leading to many generic outputs saying that an answer could not be generated due to insufficient information. A higher k value may introduce too much irrelevant information, while also being more computationally expensive due to a larger number of input tokens. Most of the results were higher for an intermediate value of $k = 5$, which has been set as the default.

Generator LLM comparison

	Average	Faithfulness		Answer Relevance	
	Response Time (s)	GPT Score	RAGAs	GPT Score	RAGAs
GPT-4o-mini	**6.68**	0.842	**0.864**	**0.792**	**0.615**
DeepSeek-V3-0324	20.37	**0.937**	0.752	0.764	0.513
Gemma-3n-E4B	8.82	0.839	0.664	0.63	0.512

Table 4.2: Evaluation results using different LLMs in the generator component

The complete system was evaluated with different large language models used as the generator, measuring the performance in faithfulness, answer relevance, and average response time, with a fixed value of $k = 5$ to isolate the effects of the models. All LLMs were run with temperature 0, with the aim of making the output more deterministic for a consistent evaluation. The GPT-4o-mini achieved the highest scores in most of the categories, as well as the lowest response times, making it the default choice for the chatbot system. DeepSeek-V3-0324[1] offers moderate performance, but it shows the longest average response time, and the costs are higher than for the GPT-4o-mini model, using an API with a comparable format to OpenAI API. Gemma-3n-E4B[2] underperforms in all metrics, but it has the advantage of being open-source, and may be

[1] https://huggingface.co/deepseek-ai/DeepSeek-V3-0324
[2] https://ai.google.dev/gemma/docs/gemma-3n

run locally to provide less latency and no recurring costs.

4.4 Negative Control Testing

Due to the fact that these metrics rely on a judge LLM, the outputs may appear credible even if something is not functioning properly. To verify the reliability of the evaluation results, a negative control test was performed as a *"sanity check"*. For this, the prompt passed to the judge LLM included random, out of domain statements that were not relevant to the context, alongside what was being evaluated, expecting very low scores as a result.

For **MAP@k** and **MRR@k**, where the LLM is prompted to judge the relevance of the *retrieved context* to the *user query*, the original *query* was replaced with a random question unrelated to the topic, with answers not found in the context. For **faithfulness**, where the judge LLM evaluates the informational consistency of the *answer* to the *retrieved context*, the original answer was replaced with a random, out of domain statement. This was also the case for the **answer relevance**, which measures how much an *answer* adresses the *user query*, and which should penalize incompleteness and redundancy.

		Faithfulness		Answer Relevance	
MAP@k	MRR@k	GPT Score	RAGAs	GPT Score	RAGAs
0	0	0	0	0	0.102

Table 4.3: Negative control test results for the complete system, with $k = 5$

As expected, most of the results shown in Table 4.3 are 0, which confirms that the evaluation scores poorly when it should, suggesting that the prompts passed to the judge LLM are adequate and contain clear instructions. The *Answer Relevance – RAGAs* metric scored slightly above 0, likely due to the implementation that uses vector embeddings cosine similarity. This may not yield a perfect 0 value for two different text sequences, despite being contextually unrelated, possibly due to the fact that in the multi-dimensional space, the embeddings may not always be orthogonal, leading to scores that are close to, but not exactly 0.

5 Conclusions and Further Work

5.1 Conclusions

Large language models are pre-trained on large volumes of data to learn complex representations of natural language and incorporate substantial amounts of knowledge in their parameters. However, this parametric memory is limited to the information available in the training data. In this thesis, I implemented a retrieval-augmented generation pipeline for a conversational chatbot, enhancing the context awareness of pre-trained large language models to enable source-grounded question answering. The purpose of this is to provide up-to-date knowledge without retraining the model. Most of the collected data used as a knowledge base is more recent than the knowledge cut-off date of the large language model used to generate the responses. The chatbot has shown to generate answers that incorporate this newer knowledge, with traceable sources to allow the user to manually fact check the information.

The evaluation consists of using an LLM as a judge to assess the relevance of the retrieved context to the user query, the faithfulness of the generated response to the retrieved context, as well as the answer relevance to the query. The results of an ablation study supports the importance of using a reranker component to reorder the retrieved passages. A k-value sweep was performed, varying the number of retrieved documents (k) to analyze how it affects performance. The results show that a smaller k value, despite being less computationally expensive, with fewer input tokens, may not include sufficient relevant information, leading to incomplete answers. Higher k values can introduce noise that leads to degraded performance, while also utilizing longer prompts.

5.2 Further Work

There are numerous directions to be explored which may lead to an increase in the performance of the proposed solution. Further work includes implementing guardrails to identify malicious intent or harmful content in the input and output, experimenting with different available moderation APIs.

Another valuable direction that may be worth exploring is prompt refinement, investigating more advanced prompting techniques, such as chain-of-thought, which encourages the model to endorse a step-by-step reasoning, or experimenting with few-shot prompting by adding examples of high quality question-answer pairs with source attribution to help the model mimic the desired behaviour.

Regarding hallucination mitigation, a secondary rule-based layer or large language model may be helpful to detect and flag claims from the generated response which are not supported by the retrieved context.

The document processing and retriever components exhibit a significant improvement potential, with multiple areas to be explored. Varying the chunking strategies and the chunk sizes, as well as implementing a hybrid search that combines semantic search with keyword search may lead to potential improvements. Various further experiments and tests may be done to determine better approaches, and in order to draw clear and grounded conclusions, the evaluation process must be refined, investigating additional strategies. A limitation of the current retrieval system may be the "one-size-fits-all" approach with a fixed k value, which may lead to either insufficient or excessive contexts. Building on this, an adaptive retrieval depth approach may be explored that dinamically adjusts the number of retrieved passages.

Further improvements regarding the user interface and overall experience when interacting with the chatbot may include customizing the interface to allow the filtering by document metadata, as well as implementing previews that allow users to expand and preview the cited source.

Appendix

A JSON File with Source URLs for Corpus Construction


```
{
    "acs": [
        "https://acs.pub.ro/admitere/viitorii-studenti/",
        "https://acs.pub.ro/admitere/licenta/",
        "https://acs.pub.ro/admitere/masterat/",
        "https://acs.pub.ro/admitere/doctorat/",
        "https://acs.pub.ro/educatie/licenta/",
        "https://acs.pub.ro/educatie/masterat/"
    ],
    "aero": [
        "http://www.aero.pub.ro/ro/admitere-licenta/",
        "http://www.aero.pub.ro/ro/programe-de-licenta/",
        "http://www.aero.pub.ro/ro/programe-de-master/",
        "http://www.aero.pub.ro/ro/admitere-master/"
    ],
    "chimie": [
        "https://chimie.upb.ro/admitere/programe-de-studii-de-licenta",
        "https://chimie.upb.ro/admitere/programe-de-studii-de-masterat",
        "https://chimie.upb.ro/admitere/programe-de-studii-de-doctorat",
        "https://chimie.upb.ro/educatie/programe-de-studii-de-licenta/",
        "https://chimie.upb.ro/educatie/programe-de-studii-de-masterat"
    ],
    "electro": [
        "https://www.electro.upb.ro/admitere-licenta/",
        "https://www.electro.upb.ro/planuri-de-invatamant/",
        "https://www.electro.upb.ro/admitere-master/",
        "https://www.electro.upb.ro/planuri-de-invatamant-2/"
    ],
    "energ": [
        "https://energ.upb.ro/admitere/licenta/de-ce-sa-ne-alegi",
        "https://energ.upb.ro/admitere/licenta/informatii-admitere",
        "https://energ.upb.ro/admitere/masterat/informatii-admitere",
        "https://energ.upb.ro/admitere/doctorat/informatii-admitere",
        "https://energ.upb.ro/educatie/studii/licenta",
        "https://energ.upb.ro/educatie/studii/master",
        "https://energ.upb.ro/educatie/studii/doctorat"
    ],
    "etti": [
        "https://etti.upb.ro/oferta-educationala/domenii-licenta/",
        "https://etti.upb.ro/oferta-educationala/admitere-licenta/",
        "https://etti.upb.ro/oferta-educationala/admitere-masterat/",
        "https://etti.upb.ro/oferta-educationala/programe-de-masterat/"
    ],
    "faima": [
        "https://faima.upb.ro/admitere-faima.php",
        "https://faima.upb.ro/licenta.php",
        "https://faima.upb.ro/master.php",
        "https://faima.upb.ro/doctorat.php",
        "https://faima.upb.ro/postuniversitar.php"
    ],
    "fiir": [
        "https://www.fiir.upb.ro/index.php/ro/licenta/admitere-licenta",
        "https://www.fiir.upb.ro/index.php/ro/licenta/planuri-de-invatamant",
        "https://www.fiir.upb.ro/index.php/ro/masterat/admitere-masterat",
        "https://www.fiir.upb.ro/index.php/ro/masterat/planuri-de-invatamant",
        "https://www.fiir.upb.ro/index.php/ro/scoala-doctorala/admitere-doctorat"
    ],
    "fils": [
        "https://fils.upb.ro/ro/admitere/",
        "https://fils.upb.ro/ro/admitere-master/",
        "https://fils.upb.ro/ro/admitere-doctorat/",
        "https://fils.upb.ro/ro/oferta-educationala/",
        "https://fils.upb.ro/ro/licenta/",
```

```
64          "https://fils.upb.ro/ro/masterat/",
65          "https://fils.upb.ro/ro/doctorat/"
66      ],
67      "fim": [
68          "https://fim.upb.ro/admitere-licenta/",
69          "https://fim.upb.ro/admitere-masterat/",
70          "https://fim.upb.ro/informatii-generale/",
71          "https://fim.upb.ro/programe-licenta/",
72          "https://fim.upb.ro/programe-master/"
73      ],
74      "fsa": [
75          "http://fsa.pub.ro/calendar-admitere",
76          "http://fsa.pub.ro/admitere-master",
77          "http://fsa.pub.ro/discipline-de-concurs"
78      ],
79      "isb": [
80          "https://isb.pub.ro/admitere-s3/",
81          "https://isb.pub.ro/admitere-master/",
82          "https://isb.pub.ro/licenta/",
83          "https://isb.pub.ro/master_isb/"
84      ],
85      "mecanica": [
86          "http://www.mecanica.pub.ro/new/index.php/admitere-2025/",
87          "http://www.mecanica.pub.ro/new/index.php/admitere-master-2025/",
88          "http://www.mecanica.pub.ro/new/index.php/plan-de-invatamant/",
89          "http://www.mecanica.pub.ro/new/index.php/plan-invatamant-master/"
90      ],
91      "sim": [
92          "https://www.sim.upb.ro/index.php/admitere-licenta/",
93          "https://www.sim.upb.ro/index.php/admitere-master/",
94          "http://sdsim.upb.ro/"
95      ],
96      "transport": [
97          "https://transport.upb.ro/admitere-licenta/",
98          "https://transport.upb.ro/admitere-masterat/",
99          "http://doctorat.transport.upb.ro/",
100         "http://transport.upb.ro/licenta",
101         "https://transport.upb.ro/masterat/"
102     ],
103     "unesco": [
104         "https://www.unesco.chair.upb.ro/programe-de-studii/",
105         "https://www.unesco.chair.upb.ro/admitere/"
106     ],
107     "upb": [
108         "https://upb.ro/regulamente-si-rapoarte/#1690451516639-650bb9a3-e876",
109         "https://upb.ro/regulamente-si-rapoarte/#1690451517189-68980d38-0362"
110     ],
111     "upit": [
112         "https://www.upit.ro/ro/academia-reorganizata/facultatea-de-stiinte-educatie-
                fizica-si-informatica/admitere-fsefi",
113         "https://www.upit.ro/ro/academia-reorganizata/facultatea-de-mecanica-si-
                tehnologie-2/admitere-fmt-2",
114         "https://www.upit.ro/ro/academia-reorganizata/facultatea-de-mecanica-si-
                tehnologie-2/admitere-fmt-master",
115         "https://www.upit.ro/ro/academia-reorganizata/facultatea-de-mecanica-si-
                tehnologie-2/admitere-fmt-doctorat",
116         "https://www.upit.ro/ro/academia-reorganizata/facultatea-de-electronica-
                comunicatii-si-calculatoare-2/fecc-admitere",
117         "https://www.upit.ro/ro/academia-reorganizata/facultatea-de-stiinte-economice
                -si-drept-1/admitere-2025-1",
118         "https://www.upit.ro/ro/academia-reorganizata/facultatea-de-stiintele-
                educatiei-stiinte-sociale-si-psihologie-2/admitere-licentamaster-fsesspl"
                ,
119         "https://www.upit.ro/ro/academia-reorganizata/facultatea-de-teologie-litere-
                istorie-si-arte/admitere-ftlia"
120     ]
121 }
```

B Query Refiner Prompt Template

```
"""
Eşti un asistent specializat în reformularea întrebărilor din conversaţii.
Analizează întrebarea utilizatorului şi conversaţia anterioară pentru a înţelege contextul. Identifică ambiguităţile şi reformulează întrebarea astfel încât să fie complet clară şi înţeleasă fără a fi nevoie de context suplimentar.
Păstrează cât mai mult din structura şi formularea originală, dar adaugă explicit detaliile şi informaţiile contextuale relevante.

Contextul conversaţiei:
{conversation_history}

Întrebarea:
{query_text}
"""
```

C Response Generation Prompt Template

```
"""
Eşti un asistent care răspunde la întrebări pe baza contextului din documentele relevante.
Generează un răspuns formal pe baza datelor primite. Ia în considerare întreaga întrebare, nu ignora detalii sau părţi din întrebare.
Pentru fiecare afirmaţie furnizează explicit sursa extrasă din metadatele documentului.
Dacă nu poţi formula un răspuns pe baza datelor primite, spune că nu ai suficiente informaţii pentru a răspunde la întrebare, nu încerca să inventezi un răspuns.

Documente relevante:
{relevant_docs}

Întrebarea utilizatorului:
{query_text}

Formatează răspunsul ca JSON astfel:
{{
    "response": "<Răspunsul tău detaliat formatat ca Markdown>",
    "sources": [
        {{
            "source": "<sursă>",
            "url": "<url>",
            "page": <pagina>  # pagina este opţională, poate fi null
        }}
    ]
}}

Respectă formatul JSON chiar dacă nu ai informaţii suficiente pentru a răspunde la întrebare.
"""
```

D Context Relevance Prompt Template Adapted from the RAGAs Framework - 1

```
"""
### Instrucţiuni

Eşti un expert de clasă mondială conceput să evalueze scorul de relevanţă al unui Context pentru a răspunde la Întrebare.
Sarcina ta este să determini dacă Contextul conţine informaţii corecte pentru a răspunde la Întrebare.
```

```
Nu te baza pe cunoștințele tale anterioare despre Întrebare.
Folosește doar ceea ce este scris în Context și în Întrebare.

Urmează instrucțiunile de mai jos:
0. Dacă Contextul nu conține nicio informație relevantă pentru a răspunde la întrebare, spune 0.
1. Dacă Contextul conține parțial informații relevante pentru a răspunde la întrebare , spune 1.
2. Dacă Contextul conține informații relevante pentru a răspunde la întrebare, spune 2.
Trebuie să furnizezi scorul de relevanță 0, 1 sau 2, nimic altceva.
Nu explica.

### Întrebare:
{query}

### Context:
{context}

Nu încerca să explici.

Analizând Contextul și Întrebarea, scorul de relevanță este
"""
```

E Context Relevance Prompt Template Adapted from the RAGAs Framework - 2

```
"""
În calitate de expert special conceput să evalueze scorul de relevanță al unui Context dat în raport cu o Întrebare, sarcina mea este să determin măsura în care Contextul oferă informațiile necesare pentru a răspunde la Întrebare.
Mă voi baza exclusiv pe informațiile furnizate în Context și în Întrebare, fără a utiliza cunoștințe anterioare.

Iată instrucțiunile pe care le voi urma:
* Dacă Contextul nu conține nicio informație relevantă pentru a răspunde la Întrebare , voi răspunde cu scorul de relevanță 0.
* Dacă Contextul conține parțial informații relevante pentru a răspunde la Întrebare,  voi răspunde cu scorul de relevanță 1.
* Dacă Contextul conține informații relevante pentru a răspunde la Întrebare, voi răspunde cu scorul de relevanță 2.

### Întrebare:
{query}

### Context:
{context}

Nu încerca să explici.

Pe baza Întrebării și Contextului furnizate, scorul de relevanță este
"""
```

F Faithfulness GPT Score Prompt Template Adapted from RAGAs

```
"""
### Instrucțiuni
Fidelitatea măsoară consistența informațională a răspunsului în raport cu contextul oferit.
Orice afirmații din răspuns care nu pot fi deduse din context trebuie penalizate.
```

```
Având un Răspuns și un Context, acordă un scor pentru acuratețe în intervalul 0-10.
Trebuie să furnizezi scorul de fidelitate din intervalul 0-10, nimic altceva.
Nu explica.

### Context:
{context}

### Răspuns:
{answer}

Nu încerca să explici.

Pe baza Contextului și Răspunsului furnizate, scorul de fidelitate este
"""
```

G Answer Relevance GPT Score Prompt Template Adapted from RAGAs

```
"""
### Instrucţiuni
Relevanţa răspunsului măsoară gradul în care un răspuns se adresează direct și este
    ↪ adecvat pentru o întrebare dată.
Ea penalizează prezenţa informaţiilor redundante sau a răspunsurilor incomplete în
    ↪ raport cu întrebarea.
Având o Întrebare și un Răspuns, acordă un scor pentru relevanţă în intervalul 0-10.
Trebuie să furnizezi scorul de relevanţă din intervalul 0-10, nimic altceva.
Nu explica.

### Întrebare:
{question}

### Răspuns:
{answer}

Nu încerca să explici.

Pe baza Întrebării și Răspunsului furnizate, scorul de relevanţă este
"""
```

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[2] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021.

[3] A. Torfi, R. A. Shirvani, Y. Keneshloo, N. Tavaf, and E. A. Fox, "Natural language processing advancements by deep learning: A survey," 2021.

[4] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[6] S. Ruder, "An overview of gradient descent optimization algorithms," 2017.

[7] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," 2024.

[8] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?," 2019.

[9] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," 2025.

[10] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, A. Castro-Ros, M. Pellat, K. Robinson, D. Valter, S. Narang, G. Mishra, A. Yu, V. Zhao, Y. Huang, A. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language models," 2022.

[11] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Transactions on Information Systems*, vol. 43, p. 1–55, Jan. 2025.

[12] T. Mikolov, W. tau Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *North American Chapter of the Association for Computational Linguistics*, 2013.

[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.

[14] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pretraining," *OpenAI Blog*, 2018.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[16] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," 2019.

[17] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," 2021.

[18] R. Smith, "An overview of the tesseract ocr engine," in *Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR)*, pp. 629–633, IEEE Computer Society, 2007.

[19] N. Reimers and I. Gurevych, "Making monolingual sentence embeddings multilingual using knowledge distillation," 2020.

[20] L. Ye, Z. Lei, J. Yin, Q. Chen, J. Zhou, and L. He, "Boosting conversational question answering with fine-grained retrieval-augmentation and self-check," 2024.

[21] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. tau Yih, "Dense passage retrieval for open-domain question answering," 2020.

[22] R. Nogueira and K. Cho, "Passage re-ranking with bert," 2020.

[23] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," 2024.

[24] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of artificial general intelligence: Early experiments with gpt-4," 2023.

[25] M. Hewing and V. Leinhos, "The prompt canvas: A literature-based practitioner guide for creating effective prompts in large language models," 2024.

[26] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging llm-as-a-judge with mt-bench and chatbot arena," 2023.

[27] H. Yu, A. Gan, K. Zhang, S. Tong, Q. Liu, and Z. Liu, *Evaluation of Retrieval-Augmented Generation: A Survey*, p. 102–120. Springer Nature Singapore, 2025.

[28] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "Ragas: Automated evaluation of retrieval augmented generation," 2025.