Generative enterprise search with extensible knowledge base using AI

Master Degree Thesis in
Data Science and Engineering

Author: **Domenico Bulfamante**

Student ID: S301780
Academic Advisor: Prof. Daniene Apiletti
Company Advisors: Dott. Tommaso Rosso, Dott. Petrea Tancau
Academic Year: 2022-23

Abstract

In the digital era, companies are inundated with huge amounts of data, making accurate and efficient access to information a relevant concern.

This master thesis delves into the integration of generative artificial intelligence (AI) in the realm of enterprise search in collaboration with Iriscube Reply, proposing a pipeline that synergies semantic embeddings and similarity search with generative AI capabilities. At the core of this approach is an information retriever that leverages semantic embeddings to understand the hidden relationships and meanings within the enterprise data. By doing so, it can effectively identify and return a set of documents most pertinent to a user's query.

Once these relevant documents are retrieved, they are fed into a generative AI system. Unlike traditional search mechanisms that simply return verbatim excerpts from documents, the generative AI system crafts coherent and contextually relevant answers to the user's questions. This enhances user experience by providing direct and clear answers and reduces the cognitive overload on the user, eliminating the need to sift through multiple documents for information.

During the component selection phase which forms the proposed pipeline, evaluations were conducted on the required resources to deliver a viable and scalable POC. Furthermore, a series of experiments were conducted to evaluate the effectiveness of the chosen component.

In conclusion, this thesis underscores the potential of integrating generative AI into enterprise search systems. By combining the advantages of semantic embeddings for information retrieval and generative AI for answer generation, enterprises may lower the barriers to understanding numerous internal processes and offer a more efficient, streamlined, and user-friendly search experience. As enterprise data volumes steadily increase, such advancements will prove essential to enhance decision-making and boost productivity.

Keywords: Natural Language Generation, Semantic Search, Information Retrieval

Contents

Summary

Thesis Objectives

In today's business landscape, enterprises are overwhelmed by many files and documents, making it challenging to locate the required information. The problem is compounded by copious amounts of documentation and dispersed data across numerous channels, which hampers decision-making while slowing down internal processes. The real challenge is to access high-quality and relevant information without navigating through vast amounts of often unrelated data.

Centralizing and standardizing the management of documents from different sources and formats would enable any employee to obtain the information they need by querying an intelligent system.

This master thesis, conducted in partnership with Iriscube Reply, explores the integration of generative AI by proposing a pipeline that synergizes semantic embeddings and similarity search with the capabilities of generative AI models. The critical element of this method is an information retriever that utilizes semantic embeddings to comprehend the connections and implicit significances among words. By following this approach, the system can identify and retrieve a set of documents that are most relevant to the user's query in an efficient manner. Subsequently, the relevant documents are inputted into a generative AI system. Unlike traditional search mechanisms, which return literal extracts of documents, the generative artificial intelligence system creates coherent and contextually relevant answers to the query. This feature enhances the experience by providing precise and direct solutions, reducing cognitive overload for the user, and eliminating the need to sift through multiple documents to obtain information.

This thesis aims to develop a POC (Proof of Concept) to overcome these challenges by implementing a web platform where employees interact with the proposed system through a chat. Using a chat system with artificial intelligence aims to provide a user-friendly interface, allowing individuals to make natural requests without using complicated query languages.

Research Area and Contribution

In this research, each element that forms the system has been analyzed and carefully chosen to secure a dependable and efficient platform that produces prompt and precise responses. The platform can be divided into two main components: a web application and the Retrieval Augmented Generation (RAG) system. The latter component can be subdivided into a text embedding model, document reader, vector database, and Large Language Model (LLM).

The text embedding model generates vectors that capture the semantic meaning of the text, aiding the identification of document chunks relevant to the user query during the semantic search process. The experiments aimed to compare multiple models to determine the most optimal one for retrieval tasks. In order to optimize and improve the retrieval of text chunks related to the query, a customized algorithm was developed within the document reader to segment the text accurately. After conversion to plain text by the document reader, the text is divided into self-informative segments, considering elements like headings, lists, and tables. The vectors produced through the text embedding model are then inserted into the vector database, a NoSQL database optimized for adequate storage and indexing of multidimensional vectors. Thanks to this database, a similarity search is performed between the query vector and the vectors representing the text chunks of the chosen document collection. One area of particular interest involved selecting an LLM capable of producing text based on questions and related data. The choice was guided by considering cost, performance, efficiency, and whether open or closed-source solutions were preferred.

Evaluating an RAG is a challenging task due to the two distinct components of the system performing different functions. This was done using the Ragas framework, which utilizes a large language model, LLM, to jointly assess information retrieval and response generation. The use of LLM in evaluating such systems enables a closer approximation to human evaluation than conventional metrics. Additionally, the framework is appropriate for assessing a system for a specific use case, using customized datasets extracted from actual samples that do not have ground truth.

As for the web app, the technology used for the front end is Angular; it provides robust single-page web application management. On the other hand, Flask was chosen for the back-end management, a Python micro-framework whose simplicity and flexibility accelerate the development of the prototype. Its compatibility with other Python libraries is essential, particularly in pre-processing documents and converting them into vectors.

Results

The evaluation results of the RAG have shown that LLMs when combined with an information retrieval system, can generate responses with high accuracy within specific contexts. Furthermore, these experiments have enabled selection of particular system settings, such as the maximum token number for each text chunk and the type of index to use for semantic search. Traditional methods could not have chosen these settings empirically.

Moreover, the experiments have confirmed the hypothesis that modern text embeddings have significantly improved in recent years, highlighting remarkable advancements in this field. Previous models were faster but significantly less effective, especially in the retrieval task, underscoring the importance of specific training.

The platform is designed to be scalable and easily expandable, allowing for a high level of customization in each component. Additionally, special precautions have been taken to ensure the security and privacy of the data used, a common concern when using generative AI models as a service.

In conclusion, the proposed platform has succeeded in its effectiveness and user-friendliness. Its increasing usage among Iriscube Reply employees makes it an ideal solution for internal use and as a product offered to third parties.

Introduction

Managing information in modern businesses has become increasingly complex due to the rise in digitization and subsequent increase in data. The digital landscape has led to a proliferation of files and documents, creating storage and efficient retrieval challenges.
Accessing high-quality information efficiently is crucial for optimizing a company's internal processes. Centralizing and standardizing document management from various sources and in diverse formats could provide each employee with the information they need by querying an intelligent system capable of organizing and understanding documents and providing accurate and comprehensive answers.
The purpose of this thesis is to develop a proof of concept (POC) that addresses these challenges by implementing a web platform where employees can interact with documents through a system that combines information retrieval with the generative capabilities of large language models.
Using an artificial intelligence chat system aims to provide a user-friendly interface, allowing individuals to make natural inquiries without using complicated query languages.

In chapter 1.1.1, all the knowledge needed to understand the architecture of the system is provided, i.e. the main milestones in NLP (Natural Language Processing), starting with the classical algorithms (BoW [20], TF-iDF [28]) and arriving at modern architectures based on transformers [36]. The latter has significantly progressed the field of NLP due to the systems' ability to capture intricate semantic connections amidst a wealth of text data. Previous models utilised less scalable and rigid techniques, unlike transformers, which employ attention mechanisms to assess the importance of each word in a text. This technique leads to an enhanced understanding of language through a deeper analysis and contextualisation of language. With the ability to handle longer texts and capture complex relationships, transformers have achieved unprecedented success in various NLP-related tasks such as machine translation, text classification and question-answering systems.
Another revolutionary aspect of transformers is their application in question-answering systems; before traditional systems were often limited to providing answers from a predefined database, transformers have now introduced the ability to generate dynamic answers. This new capability means they can create answers based on the specific context of the

question and the available information rather than simply drawing from a preset set of answers.
The transformer architecture gives rise to large language models, models trained on huge amounts of textual data that can understand and generate language that surprisingly resembles human language.
However, these models have limitations. Firstly, their knowledge is limited to the training dataset and may need to be more useful in specific contexts. Additionally, there may be instances of hallucinations and unreliable responses.

One solution to overcome these issues is combining large language models with information retrieval systems. Among these systems is semantic search, which can consider the semantic meaning of the query and the documents, making the search for relevant information more accurate. Furthermore, through semantic search, one can comprehend the information seeker's intention and the contextual importance of terms in the searchable data space, leading to more pertinent outcomes.
By combining large language models with semantic search, one can obtain a Retrieval Augmented Generation (RAG) [17] system. This comprises a retriever, which finds the most relevant information to the user's query, and a generative model, which utilizes the gathered information to generate a more accurate response.

Subsequently, the architecture of the POC, as mentioned earlier, was analyzed in chapter 2. Generative Knowledge base platform allows a collection of documents to be loaded, which are used as a source of information. Collections can be private or shared with other users for centralizing information sources. The documents are pre-processed to extract text segments, which are then converted into vectors by the text-embedding model (text-embedding-ada-002). These vectors are subsequently inserted into the vector database (Redis), a NoSQL database optimized to store efficiently and index multi-dimensional vectors.
After selecting the desired collection to analyze, the user engages in a conversation with the large language model, which activates a semantic search upon receiving a question. Utilizing the text embedding model, the question is transformed into a vector. This vector is then transmitted to the vector database, where a similarity-based search is conducted to retrieve the most relevant document segments, albeit from various sources. At this stage, the most pertinent documents and the initial query are inputted into the advanced language model (GPT3.5-Turbo) to produce a comprehensive and unbiased answer.

The technology used for the front end of the Generative Knowledge base platform is Angular, which provides solid management of the single-page web app. On the other hand, Flask was chosen for the back-end management, a Python micro-framework whose simplicity and flexibility accelerate the development of the prototype. Its compatibility with other Python libraries is essential, particularly in the pre-processing of documents and their conversion into vectors.

The Generative Knowledge base platform has been designed with scalability and extensibility in mind, allowing for each component's easy replacement or adaptation. This flexibility applies, for instance, to the text embedding model or the large language model, which can be updated or modified as necessary.

Finally, the chapter 3 presents the two types of experiments that were carried out. The first type evaluates various text embedding models. The selection of an appropriate model is vital for optimal system performance. The model generates vectors, which capture the semantic meaning of the text, facilitating the identification of documents that closely relate to the user's query during the semantic search.

The second set of experiments focuses on assessing the RAG system, which is challenging due to the system's two distinct components that perform varying functions. It is imperative to evaluate the system's performance individually and collectively. The Ragas [9] framework was employed to evaluate the RAG and offers a unique methodology that utilises a large language model to gauge both information retrieval and response generation.

A greater affinity to human evaluation can be achieved using a large language model than traditional metrics. Ragas enables the calculation of five measures: context relevancy, context recall, faithfulness, answer relevancy, and the Ragas score. Context relevancy and context recall serve as means to evaluate the effectiveness of retrieval systems, while faithfulness measures hallucinations. Answer relevancy indicates how pertinent the generated answer is to the posed question. Conversely, the Ragas score represents the harmonic mean of the four measures introduced. It offers an indicative value of the performance of the proposed RAG. Finally, the framework is suitable for assessing a system on a particular use case by employing custom datasets that lack ground truth.

1 | Background

The natural language is the primary medium humans communicate and express their thoughts. The term NLP (Natural Language Processing) is used to describe the field where the natural language is analyzed.
This chapter contains a brief introduction to the history of NLP in section 1.1.1, and then moves on to the remarkable contribution that the deep models gave in section 1.2 thanks to the exponential growth of the computational power of computers.
Moreover, the focus will be on generative question answering and how, in some use cases, it can be empowered by some information retrieval techniques in 1.5.

1.1. Introduction to Natural Language Processing

The final purpose that has always been a part of Natural Language Processing is to let a machine understand the human-written text.
The fact that our language is complex implies that the knowledge of the machine is divided into different levels, and it can be seen as a pipeline:

1. Lexicon: the dictionary of words used.
2. Morphology: the study of how the various parts of the words, the morphemes (root and affixes), or a unit minimum of first articulation, are combined to form a word (eat / eat-ing). The morphological analysis aims to break down a word into its morphemes, a problem similar to stemming that reduces a word to its basic form or lemma.
3. Syntax: the study of combining words for constructing a sentence at the constituent level, grammatical relations (the link between subjects and objects), and dependence.
4. Semantics: study the techniques for interpreting a sentence that requires knowing the meaning of each word and how these combine to give the sentence an overall meaning.

5. Pragmatic: Study how the meaning of a phrase is used to manifest a purpose or intention within a particular context or situation. The context allows the correlation between the sentence's literal meaning and the intention or concept that the user wants to convey. It includes social or cultural knowledge shared between interlocutors and spatial-temporal information about their situation or progress within the discourse.

Applications regarding the spoken language generally involve the analysis of an additional level, the lower of all that concerns the Phonetic and Phonological analysis or the sounds that constitute a language.
The primary challenge NLP methods encounter is the removal of ambiguity, an inherent property in a linguistic element that causes a structure to be ambiguous when it can have multiple interpretations. Ambiguity can manifest at various levels and requires elimination. At the morphological level, disparate meanings overlap in a shared definition. If they share the exact spelling, they are classified as homographs, but if they share the same pronunciation, they are known as homophones. If two different meanings share both aspects, they are referred to as homonyms. At the syntactic and structural level, ambiguity arises when a sentence's syntax can be interpreted in diverse ways, resulting in different meanings.

1.1.1. History

Alan Turing, a renowned mathematician, suggested the imitation game to measure a machine's ability to display intelligent behaviour [35]. The test requires a human to write a question and a computer to generate an answer. The machine must comprehend the meaning and interact in natural language for the machine to respond accurately. Since then, experts have investigated various methodologies to enable machines to understand and produce texts that resemble human language.

During the 1960s, research was primarily focused on generating rules to model human language manually. Data-driven approaches were deemed impractical due to the size of the necessary data, the high processing overhead, and the need for efficient learning algorithms.

Thus, the viable method for language models has manually defined rules that incorporate local linguistic dependencies for specific NLP tasks. Despite their usefulness, these approaches had notable limitations: the rules' effectiveness relied on the knowledge of their creators and updating and adapting them to new languages took much work.

In the late 1980s, performance was improved thanks to the adoption of statistical methods, aided by the increasing processing power of computers and access to extensive texts. Statistical methods have addressed the limitations of their predecessors, enabling long-term modelling of language dependencies, automating the training process and reducing the reliance on human input[18]. This transition from traditional to statistical techniques facilitated the development of new machine learning algorithms, such as decision trees [26], which have demonstrated their effectiveness in various natural language processing activities, including part-of-speech tagging [6].

In applying statistical techniques, language processing and generation activities utilized the concept of n-grams. An n-gram is a sequence of n consecutive elements in a text sample, expressed in words or characters. For instance, a bi-gram, consisting of two words taken from the phrase "machine learning and deep learning", would incorporate "machine learning", "learning and", and so forth. In a similar vein, a character-based bi-gram would be "ma", "ac", "ch", "hi", "in", "ne" and so on. The idea is to create a series of elements and use statistical techniques to determine the probability of their co-occurrence.

This concise yet dynamic definition has spurred the advancement of innovative text generation systems [19], information retrieval techniques [21], and text mining methods [31], among others. Applying n-gram definitions has given rise to a fresh approach to document representation that centres on individual words.

Text representation For a model to read texts, they must be encoded as a continuous vector of numerical values. Feature-based text representations encode basic text information into a representation of feature values. This process is called text vectorization. Early approaches were based on co-occurrence of words or precisely n-grams.

Bang-of-words text vectorization Given a document D that contains the words $(w_1, w_2, ..w_n) \in D$, the bag-of-words representation of D consists of a vector where each dimension is associated with a word w_i. Each vector element is set to 1 if the word appears in the document; otherwise, it is set to 0. Although recent systems have adopted these approaches, there are significant drawbacks. Firstly, the vectors' high dimensionality makes manipulating them complex.

Additionally, the vectors could be more sparse, indicating that many items are set to 0, rendering calculations arduous and necessitating substantial memory. Lastly, the vectors cannot encode contextual information; thus, words within the same context do not share similar vectors.

Vector Space Model The encoding used in the bag-of-words representation poses a fundamental issue since it merely indicates whether a word appears at least once in a document. Although easy to use, this approach fails to capture a word's frequency in a document. The *Vector Space Model* (VSM) [29] provides a solution to this problem.

Although initially designed for information retrieval, it has found extensive use in various NLP tasks. In the VSM, the frequency of each word is used to represent it rather than mere word presence. This approach has been further elaborated upon by incorporating TF-IDF (term frequency-inverse document frequency) [28]. It is a frequently applied statistic that seeks to indicate the significance of a term in a document with a collection or corpus. The inverse document frequency (IDF) aims to penalize commonly occurring words (such as those frequently found in the collection and, therefore, less distinctive).

While this approach is more precise than the bag of words, it still encounters the same problems of scarcity and absence of contextual information.

Latent Semantic Analysis The *Latent Semantic Analysis* (LSA) [7] approach relies on *Singular Value Decomposition* (SVD) [11]. This technique condenses matrix dimensions to reveal the hidden connections between data.

The matrix utilized is known as the term-document matrix, in which each row signifies a vocabulary term, and each column symbolizes a document. Furthermore, LSA vectorizes texts by presenting documents in a low-dimensional space based on the SVD of the term-document matrix, enabling the discovery of concealed relationships between terms and documents.

The method described tackles the issues of data sparsity and size by creating a space of reduced dimensions that covers the fundamental concepts of the initial matrix. Although LSA successfully reduces the dimensions, it is limited in capturing contextual details and word sequence, thus limiting its ability to identify the semantic importance and order of words. Consequently, LSA's capacity to capture the text's intricate meanings and contextual connections is restricted. It is vital to investigate alternative approaches that integrate context and sequential dependencies for thorough comprehension and analysis of language.

1.2. Deep NLP

With the rise of deep learning, the techniques and methodologies for comprehending natural language have undergone significant transformations. Deep learning models have shown remarkable effectiveness in different NLP tasks, particularly semantic comprehension. Unlike conventional methods, these models prioritize encoding textual information into latent vector spaces that capture contextual subtleties.

1.2.1. Semantic word embeddings

Conscious of the previous limitations of text vectorization methods, the next generation of text embedding is developed using deep learning's generalization capability. It is based on the *distributional hypothesis* [12], which assumes that words with similar meanings are likely to appear in similar text contexts. LSA was the earliest attempt to use this hypothesis for NLP, but more was needed to establish the semantic relationships between words. The advancement in computing power and the theoretical framework of hypothesis have enabled the creation of machine learning models that can portray words as vectors in high-dimensional spaces, leading to the mapping of words with analogous meanings to comparable vectors.

Early models, based on shallow neural networks, facilitated dense word representations. One of the most renowned and efficacious methods is Word2Vec [20]. Developed in 2013, Word2Vec utilizes a neural network model targeting to capture words' meanings and contexts in a vector space of high dimensions. The primary assumption is that each word relies on the surrounding words; this denotes the C context, which includes the k words encompassing the target word w. Word2Vec may exploit either of two model architectures:

- CBOW (Continuous Bag-of-Words): the model predicts the current word from a window of surrounding context C of k words. The order of context words does not influence prediction (bag-of-words assumption), and word repetition's eventual presence is ignored.
- Skip-Gram: the model uses the current word to predict the surrounding context window C of k words. The skip-gram architecture weighs nearby context words more heavily than distant context words.

The model consists of a single hidden layer that produces the vector representation of words; these vectors are initiated with random values and gradually updated during training. It is essential to note that the training procedure does not require annotations. Both training strategies are represented in Fig. 1.1.

These word embeddings contain important semantic information that enables the detection of relationships between words. To understand how similar are two words (w_i, w_j) the similarity between their vector representations $(\vec{w}_i, \vec{w}_j)$ can be measure. Different measures can be used like Euclidean, manhattan or cosine similarity to accomplish this task. Each has its advantages and disadvantages regarding semantic search. The cosine

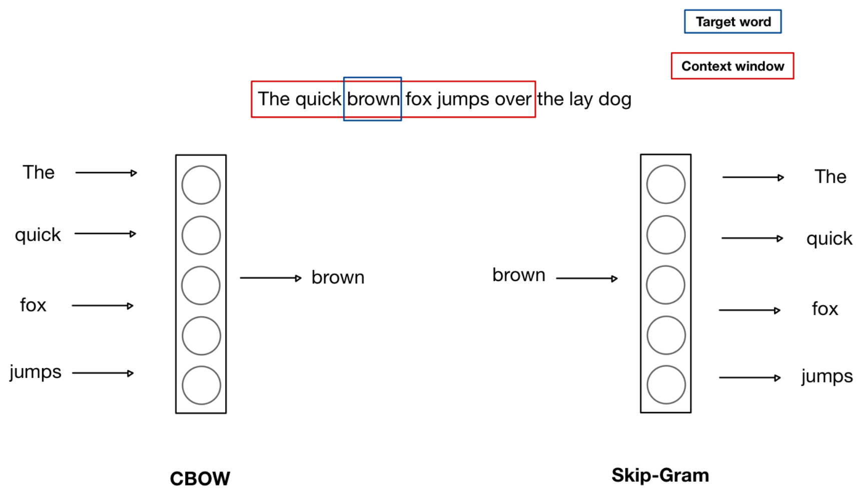


Figure 1.1: Training strategies for Word2Vec

similarity of two vectors calculates the cosine of the angle between

$$\cos(w_i, w_j) = \frac{\overrightarrow{w_i} \cdot \overrightarrow{w_j}}{\|\overrightarrow{w_i}\| \, \|\overrightarrow{w_j}\|}$$

This method is widely used because it remains constant regardless of the length of the two vectors, it is computationally efficient for scattered vectors, and the range of values it takes is from -1 to 1, which makes it easy to determine whether two words are similar (cosines close to 1), unrelated (cosines close to 0), or opposite (cosines close to -1).

Although word embeddings can address some of the previous vector representation issues, such as sparsity and lack of semantic meaning, there are still some limitations present:

- **Unique word embedding**: after training, a single vector representation exists for every word in the vocabulary. This is a big limitation since words change meaning depending on the context.
- **Out-of-vocabulary words (OOV)**: The model can only create vectors based on the words contained in the dictionary created during the training phase. In practice, the model is unable to generate representations for new words.
- **Rare words representation**: as vector representations begin with random values, rare words that occur infrequently in the training data may not have enough iterations in the neural network to stabilize their representation and, therefore, obtain a

meaningful representation.

- **Sentence and document representation**: pieces of text are represented by averaging constituent word embedding. This solution is not optimal when sentence order and structure are important.

1.2.2. RNN

Recurrent neural networks (RNN)[14] are models capable of recognizing and generating data sequences. They differ from traditional neural networks as the next element is assumed to depend on all the previous elements in the sequence. This sequential nature makes them ideal for text processing since they can consider previous words when generating the next.

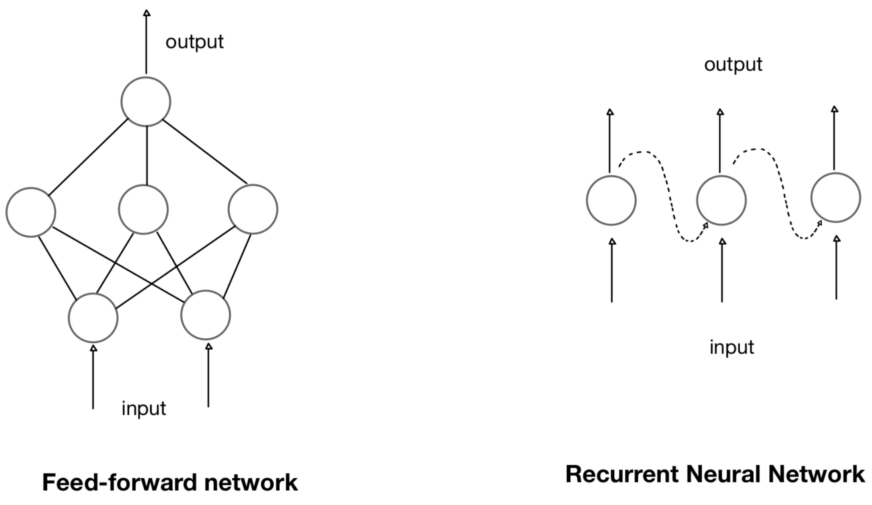


Figure 1.2: Feed-forward and recursive architecture visualisation

In Fig. 1.2, it is evident that the RNN, unlike the feed-forward neural network on the right, relies on the previous output to compute the subsequent value. This contrasts the feed-forward neural network, where input and output are independent.

Although this architecture can process long sequences of text, it suffers from certain problems, such as vanishing gradient, making it difficult for the network to learn long-term dependencies in the sequences. Some latter models that are variants of the classic RNNs, such as the Long Short-Term Memory (LSTM)[13] and the Gated Recurrent Units (GRU)[5] which, thanks to the 'gate' mechanisms, allow the gradient to flow better along the architecture and handle longer sequences. The gate determines whether to retain prior accumulated information for the generation of the following output.

These networks can achieve excellent results in various domains but require time-consuming training and are computationally expensive. Each iteration requires computing all the previous steps, which prevents parallelizing the training phase.

The last decade's challenge has been capturing the dependencies between words by attempting to accelerate or parallelize the training process.

1.2.3. Transformer

Until 2017, the best solution for encoding text sequences in fixed-size vectors were RRNs.

With Transformers [36], both issues have been resolved. Notably, the Transformers design does not require recurrence to capture long-range dependencies. This is due to its feed-forward structure which entrusts its 'memory' to a self-attention mechanism.

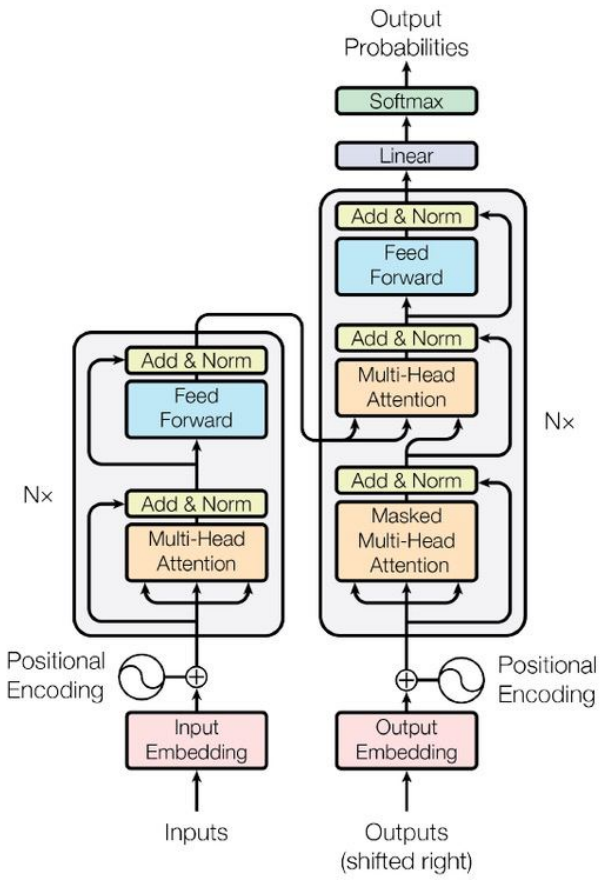


Figure 1.3: Transformer architecture [36]

The Transformer is a sequence-to-sequence (seq2seq) model that takes a sequence of items (words, letters, or time series) from one domain and converts it into a sequence from another. For example, we input a sentence to obtain another, as in translating from one language to another. In Fig. 1.3, we observe that the transformer comprises an encoder-decoder containing two primary blocks. On the left, there is the encoder block; on the right, there is the decoder block. Notably, only one encoder and one decoder are depicted for simplicity; typically, a model utilizing this structure employs a stack of both.

Attention mechanism enables the Transformer to grasp the dependencies between the sequence elements, weighing them in such a way as to give each of them a different importance. To do so, it uses learnable weights during the training phase.

Let us consider a sentence consisting of n words $S : w_1, w_2, \ldots w_n$, where each element w_i is represented through an initial x_i and a triplet of vectors q_i, k_i and v_i, respectively: "query", "key", and "value" vectors that are used when calculating the attention of that specific word. The triplet of vectors is obtained by multiplying the initial matrix of weights of W^q, W^k, and W^v with the vector x_i. To score each word that is part of the same sequence against the word itself w_i (from here the name of self-attention), the dot product is performed between q_i and k_i followed by a softmax operation. Finally, the scores are multiplied by the vector of values v_i.

To further enhance the self-attention layer, the multi-headed attention mechanism is employed. This consists of n weight matrix W^q, W^k and W^v, all randomly initialised. Each individual will acquire distinct relationships, resulting in varied representations combined through another layer of trainable weights W^0.

This architecture is therefore highly parallelizable, making training more efficient. On the other side, the computational complexity is of $o\left(n^2\right)$ for a sequence with a length of n, this generally limits the length of input sentences to 512 or 1024 tokens.

To comprehend certain variations of the transformer, it is essential to fully understand the two blocks constituting its encoder-decoder architecture.

Encoder Block

The encoder is accountable for processing the input and producing an intermediate representation for the decoder to generate the output. Analyzing the encoder block at a high level, it can observed that it is a stack of identical layers. Each of these layers can be further divided into two sublayers.

- Multi-head self-attention: this allows the encoder to focus on different parts of the input simultaneously, weighing words differently with the attention score according to their relevance.

- Point-wise Feed-Forward Networks: the output of the multi-head self-attention moves through this network, which consists of two dense layers and a ReLU activation function in between to process information using trainable weights.

In addition to these, there are also residual connections and normalisation layers. Residual connections help prevent the gradient from vanishing during training. Finally, normalisation layers stabilise learning by ensuring the output has a mean of zero and a standard deviation of one.

Consequently, the Transformer encoder produces a vector representation of d-dimensions for each position within the input sequence.

Decoder Block The decoder block has a different task from the encoder; specifically, it generates the output. Upon examination of Fig. 1.3, it is clear that in this case, in addition to the two blocks described above, there is a third block, the Multi-Head cross Attention Mechanism or encoder-decoder attention. This block highlights different parts of the encoder output, in fact, the query matrix Q of the decoder and K and V matrix of the encoder are used during the calculation. Another small variation from the encoder is the Masked Multi-head self-attention that masks the input up to the current position. This masked attention preserves the auto-regressive property, ensuring that the prediction only depends on those output tokens that have been generated.

1.3. Natural Language Generation

Natural language generation (NLG) refers to processes that lead to generating new written text in natural language with meaningful content. NLG enables models to generate comprehensible text miming what a human being would produce.

Natural Language Generation (NLG) aims to produce clear, informative, and appropriate text for a particular objective. NLG relies on models that blend information and language rules to process input data and generate useful and easily comprehensible output for users. NLG technology finds applications in various domains, including conversational automation and automatic content generation.

1.3.1. Generative Question Answering

Generative Question Answering (GQA) is a subset of natural language generation (NLG) that concentrates on generating answers to a particular question instead of choosing an answer from a predetermined list of options in the traditional question-answering technique. Indeed, this approach proves worthwhile when answers are not restricted to a fixed set of options, or when elaborate, well-defined answers are necessary. In this scenario, the models strive to comprehend the context and available information to respond to the user's question.

This, unlike classic Question Answering, enables dynamic interaction with the user, allowing for improved engagement.

1.3.2. GPT

Generative Pre-Trained Transformer (GPT)[27] is a language model based on the decoder block of the Transformers architecture. The architecture has been modified only by removing the encoder-decoder attention layer. The model inherits the autoregressive property from the decoder, which implies that the decoder looks at the sequence from left to right. The pre-training task is focused on next-word prediction, which enables the model to understand the sequential structure of the text. It's classified as an autoregressive model, wherein each generated word is fed back into the input to facilitate the continuation of the sequence generation.

As the name suggests, this system aims to generate text that is both coherent and relevant to its context. It is a useful tool for various tasks, including text generation and translation.

In its first version, GPT-1 was constructed with a stack of 12 Transformer decoder blocks, generating 117 million parameters. Pre-training was conducted on a dataset named BookCorpus[39], which comprises roughly 4.5GB of text derived from 7000 books. These two features give the model a comprehensive and abundant linguistic foundation capable of comprehending intricate language structures. The latest version of GPT-3 reaches a size of 175 billion parameters (requiring 800 GB of storage), allowing the model to produce sophisticated text almost indistinguishable from human-generated text.

GPT3.5

GPT3.5 is built on the same architecture and dataset as GPT3, with one notable difference. GPT3.5 underwent an additional post-training procedure using the Reinforcement Learning with Human Feedback (RLHF) methodology.

This methodology focuses on using human feedback to improve machine-learning algorithms. In RLHF, feedback is provided by humans directly, who attempt to rectify the actions of the model. Human feedback can be positive or negative reinforcement, altering model parameters and assigning labels to unmarked data. The primary responsibility of the human trainer is to classify and evaluate the model's prompts. RLHF also enables the removal of many toxic and incoherent outputs. This process adds an extra layer of security, allowing the model to reject requests for illegal information like crimes, weapons and adult content.

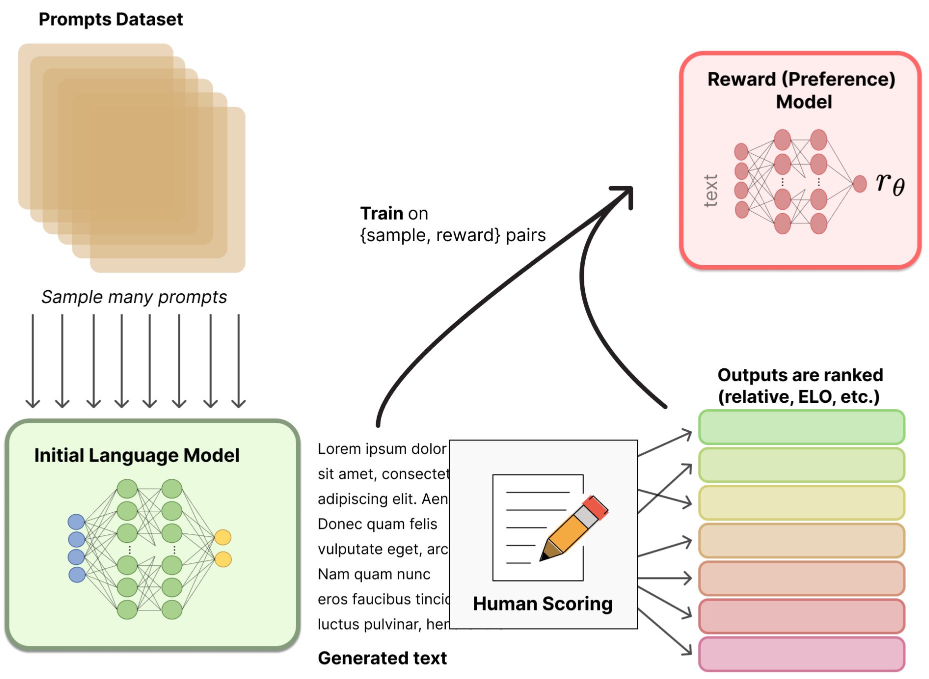

Prompts Dataset
Sample many prompts
Initial Language Model
Train on {sample, reward} pairs
Reward (Preference) Model
text
r_θ
Outputs are ranked (relative, ELO, etc.)
Lorem ipsum dolor sit amet, consectet adipiscing elit. Aen Donec quam felis vulputate eget, arc Nam quam nunc eros faucibus tincid luctus pulvinar, her
Generated text
Human Scoring

Figure 1.4: Illustration of Reinforcement Learning from Human Feedback process [16]

In this manner, OpenAI incorporated human expertise and knowledge into the model, allowing it to generate more accurate results.

GPT3.5 has been optimized for dialogue, enabling it to engage in conversation and respond to user input. The online conversation platform **ChatGPT** has enabled numerous users to experience the capabilities of GPT3.5 for themselves. Users can request any information within the model's training scope, including 570GB of text from Web Text, Common Crawl, Wikipedia, and several books.

GPT4

The last and most advanced model released by OpenAI is **GPT4** [24]. In this case, OpenAI has not released any information about the number of parameters or the training strategy, but there is a technical report. The report is anticipated to contain approximately 100 trillion parameters, which can be likened to the number of neural connections in the human brain. It is important to note that the number of parameters is not necessarily a marker of good performance; indeed, models larger than GPT-3 continue to underperform.

In this instance, the model's performance is considerably enhanced, with more precise, comprehensive, and dependable outputs than previous models. The performance was tested also by simulating examinations originally designed for humans. The model scored in the top 10% of test takers. Of particular interest is GPT4's multimodality, as it supports text input and output and images, video, and audio.

I would like to highlight that OpenAI offers access to its models via API or its website [23]. Despite their capabilities, these models have limitations:

- Suffers of hallucinations: the model "hallucinates" facts and makes reasoning errors giving false information. The model can be excessively gullible in accepting false statements from a user. This makes the model not fully reliable.
- Lack of knowledge: data used for training is up until September 2021. The model lacks precise knowledge of events or specific information after the mentioned date.
- Fail in hard problems: in the same way humans do, especially in problems containing mathematical calculations.
- Presence of biases: the input data can have various biases. Despite attempting to cleanse the data and tweak the outputs with the RLHF, these biases persist in the model outputs.

1.4. Information Retrieval

Information Retrieval (IR) is locating relevant material within vast collections to satisfy an information need. In computer and information science, this involves identifying system resources that align with an information need from a collection of such resources.

In the context of information retrieval, data can take various forms such as text, documents, images, audio, video, or any other digital content. IR aims to create algorithms, techniques, and systems for conducting efficient and precise searches within these data collections. A conventional information retrieval system consists of three key components: indexing, document representation, and query modelling. In the indexing phase, documents are analyzed and organized for more efficient searching. Document representation involves generating numerical or statistical representations that capture the crucial features of documents. Query modelling, meanwhile, aims to transform user queries into a format that can be analyzed together with indexed documents. Information retrieval forms the backbone of numerous systems and applications, including web search engines, document management systems, and other tools that facilitate swift and effective access to information.

The topic of interest to the user is referred to as the "information need". The information retrieval process commences when the user inputs a query into the system. Queries are formal declarations of informational needs, akin to search strings in web search engines. In IR, a query cannot distinguish a single object within the collection. Instead, multiple objects can correspond with the query, with varying degrees of relevance.

Unlike traditional SQL database queries, Information Retrieval (IR) search results may or may not align with the original query and are therefore typically classified. This classification of results is a significant distinguishing feature between IR and database search. Most IR systems generate a numerical score indicating the degree of alignment between each database object and the query, and objects are prioritized based on this value.

The algorithms selected by the system consider multiple factors, such as the frequency of the most frequently searched words in a document, the inclusion or exclusion of query-forming words in the title, synonyms of the searched words, and whether the document appears on certain blacklists. Finally, the search results are displayed to the user on a SERP (Search Engine Results Page), which is sorted in descending order from the most relevant (highest ranking) to the least relevant (lowest ranking).

1.4.1. Semantic Search

Semantic search is linked to information retrieval, which aims to locate the most appropriate result for a user's query. However, it surpasses conventional IR techniques by considering the semantic significance of the queries and documents, accomplished by implementing word embedding.

Semantic search aims to enhance the accuracy of searching by comprehending the searcher's intention and the contextual significance of terms in the searchable data space to produce more pertinent outcomes.

Vectors do not offer the same level of relevance as keyword searches for certain queries. Keyword search is still superior to vectors regarding single-word and exact brand match queries. However, vectors tend to perform better on multi-word queries, concept searches, questions, and other complex query types, such as:

- Document search: semantic search can assist in locating documents that pertain to particular subjects, even if they have not been indexed using the same keywords utilized in the search.
- Product search: semantic search can aid in discovering products that align with the user's requirements, even if they haven't been described with the identical keywords employed in the search.
- Information search: semantic search enables the location of precise information within a text, document, or website, without the need to search for specific keywords.

1.5. Retrieval Augmented Generation

To address apprehensions about hallucinations and unreliable responses from models such as GPT4, a possible solution is to include specific data in the generative question-and-answer system through a Retrieval-Augmented Generation (RAG) methodology. This involves incorporating a Retrieval component for a predetermined collection of documents to provide input to the QA system.

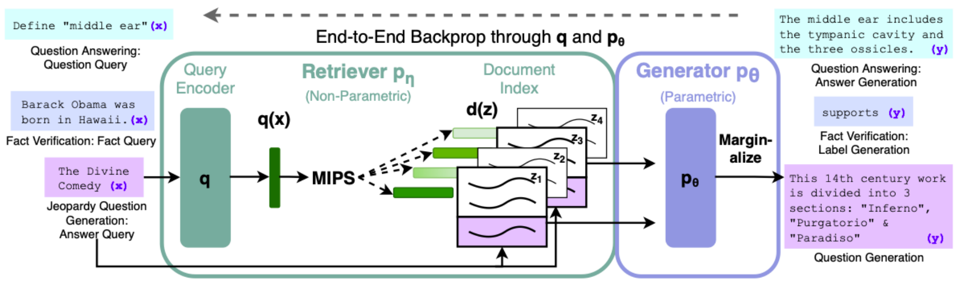

Define "middle ear" (x)
Question Answering: Question Query
Barack Obama was born in Hawaii. (x)
Fact Verification: Fact Query
The Divine Comedy (x)
Jeopardy Question Generation: Answer Query
End-to-End Backprop through q and pθ
Query Encoder
q
q(x)
Retriever pη
(Non-Parametric)
MIPS
Document Index
d(z)
z4
z3
z2
z1
Generator pθ
(Parametric)
pθ
Margin-alize
The middle ear includes the tympanic cavity and the three ossicles. (y)
Question Answering: Answer Generation
supports (y)
Fact Verification: Label Generation
This 14th century work is divided into 3 sections: "Inferno", "Purgatorio" & "Paradiso" (y)
Question Generation

Figure 1.5: Overview of the training approach used in [17]

So, a RAG behaves like a seq2seq model, which accepts a sequence as input and produces a sequence as output. The only difference is that before submitting the input to the model, the retriever finds information pertinent to that input and passes it along with the input to produce a better output. In this manner, non-parametric knowledge supplements the knowledge already present in the model.

The inclusion of external knowledge permits easy modification of information within the model. This approach also enhances the model's accuracy by providing a reliable means of verifying its knowledge. Consequently, the model can access relevant information to improve its accuracy.

As depicted in Fig. 1.5, [17] proposed a general-purpose fine-tuning method for RAG. The system utilizes a pre-trained generative model for parametric knowledge and a pre-trained neural retriever with access to Wikipedia. Subsequently, this model undergoes fine-tuning for performance enhancement throughout the entire process.

This method enhances the system's versatility, allowing it to circumvent the need for retraining when dealing with intricate and knowledge-based tasks.

2 | System Model

In this chapter, the Generative Knowledge base platform is explored, which enables Iriscube Reply employees to improve and speed up the search for information in a wide range of documents, making the search process more efficient and intuitive. The proposed system contains three main components:

- Knowledge Source: The primary source of information comprises various business documents, databases, and other information resources. Furthermore, this component contains the information retriever, responsible for retrieving documents related to the user's query.
- LLM (Large Language Model): An advanced language model interprets and comprehends user queries, utilizing retrieved information from the Knowledge Source to produce pertinent responses.
- Web Application: The interface acts as a bridge between the user and the system, aiming to be user-friendly and easy to use. It allows the user to ask questions and receive answers in real-time.

In the subsequent sections, these elements are examined in depth, considering the obstacles, remedies applied, and outcomes attained while creating and deploying the Generative Knowledge base platform within Iriscube Reply.

2.1. Knowledge Source

The Knowledge Source forms the foundation of the Generative Knowledge base platform, providing crucial information for the entire system. It is not a static entity but a dynamic combination of interconnected components that work in synergy to ensure the information retrieved is accurate, relevant, and timely.

2.1.1. Document Reader

The Document Reader converts the documents the user wants to consult. Furthermore, this component manages the processing and cleansing of the extracted data, before dividing it into smaller text blocks.

The documents used are the functional analyses of the projects the company deals with. The documents follow a uniform structure and are typically divided into chapters, with a tendency towards discursiveness. Information is often presented in tables or images, and in the case of the latter (which are not used by our template), there is always a detailed description of the figure's content.

The initial obstacle was selecting the appropriate Python library for converting from docx to strings. Several libraries encounter an issue as they cannot adequately convert tables into text and assign the correct class to headings in different sections.

The choice fell on docx2python[30], which allows these limitations to be overcome. This tool is particularly useful for reading tables and can convert docx documents into html format. The conversion process to HTML format can preserve the document's structure, including lists and headings. Following the conversion, unsupported characters are removed and the text is divided into chunks.

During the development process, various techniques have been employed to divide documents into smaller, manageable portions commonly referred to as 'chunks'. This division is necessary due to the technical and functional requirements of the system. Chunk sizes can exceed both the text embedder and LLM model's input restrictions when too big. However, if segments are too small, they lack adequate information context, potentially leading to inferior output quality.

Initially, a sentence-based division was chosen. However, this method proved inadequate as it disregarded other significant syntactic units including headings and lists.

In light of these limitations, a methodology of dividing the document into chapters was implemented, with each chapter or title comprising a separate segment. While this approach improved the quality of the results, it remained imperfect due to the highly variable segment sizes, with some sections consisting of only around 20 tokens whilst others consisting of up to 2000 tokens.

Algorithm 2.1 Docx documents splitting algorithm

```
1: Read the document and convert it in HTML format
2: Initialize the array of chunks
3: for every element of the array of pages do
4:    Create a temporary chunk element
5:    if is the element a table? then
6:       for every row of the table do
7:          if is the first row? then
8:             Create the headings element
9:          else
10:            for every cell in the row do
11:               Append at every cell the corresponding headings
12:               Append this text to the temporary chunk
13:               if is the limit of tokens reached? then
14:                  Append the chunk to the array of chunks
15:                  Create a new temporary chunk
16:               end if
17:            end for
18:         end if
19:      end for
20:   else
21:      for Every row on the page do
22:         if The row is a title? then
23:            if Is the temporary element not empty or reaches a minimum of fifty tokens?
               then
24:               Append the chunk to the array of chunks
25:               Create a new temporary chunk
26:               Append this row to the temporary chunk
27:            end if
28:         else
29:            if is the limit of tokens reached and the element is not inside a list? then
30:               Append the chunk to the array of chunks
31:               Create a new temporary chunk
32:            else
33:               Append this row to the temporary chunk
34:            end if
35:         end if
36:      end for
37:   end if
38: end for
```

Therefore, a more advanced segmentation 2.1 method has been adopted, that considers the sentence's syntax and the segment size. This approach considers lists, acknowledging their significant relevance in answering questions that demand the enumeration of particular elements. For instance, when a user requests a list of cases of a specific outcome in the presence of a bullet list, concise chunks may lead to incomplete retrieval. Therefore, short chunks are avoided and long chunks are subdivided into smaller ones. This methodology has demonstrated an optimal balance between ensuring semantic consistency, managing segment size, and preserving information integrity.

Recognition of the style is determined by the presence of HTML tags, primarily including title styles such as h1 or h2, which are then removed during the text cleaning process.

Conversion of tables to string format follows a different logic than plain text. To avoid producing unsatisfactory text that lacks semantic meaning and therefore goes unused, each cell in a row is provided with the corresponding field header. The cells are then combined with the other cells in that row, and multiple rows are merged until the token limit is reached. This helps ensure that the resulting text is effective and efficient.

2.1.2. Text Embedder

The text embedding model converts the chunks extracted by the Document Reader or the user's query into a vector representation. As mentioned in Chapter 1.1.1, significant advancements have been made in the past decade to produce models that can create vector representations of words or phrases, which encode semantic relations. This capacity for representing semantic connections is essential for determining the significance of a specific portion of text about the user's inquiry.

The following features were considered when choosing the model to be used:

- Number of input tokens: A model capable of handling many input tokens offers notable benefits. This minimizes the requirement for additional text segmentation, optimizing the amount of data stored in the database. Additionally, it hastens search operations while maintaining high accuracy and relevance in the responses.
- Supported Language: Given the project's nature and target audience, it was crucial that the chosen model adequately supported the Italian language. This ensures that linguistic and cultural nuances specific to Italian are captured and represented accurately.
- Cost and performance: Besides evaluating the model's pure effectiveness, practical aspects like its cost, speed and reliability were also considered. The balance between

cost and performance led to the selection of a model that provides the best value for the project's requirements.

FastText

FastText [2] is a machine-learning model for word embedding developed by Facebook. FastText differs from traditional word embedding models, like Word2Vec, in that it concentrates on words as distinctive units and considers n-grams of characters in words. This notion derives from the fact that English, along with other languages, is organized so that one can infer the connection between terms such as 'dog', 'dogs', and 'dogcatcher' from their spelling. As mentioned in 1.1, Morphology is the branch of natural language that studies words' internal structure and formation. In the paper, the method of subword embedding is proposed, which attempts to introduce morphological information into the skip-gram model in word2vec. The goal is to generate informative FastText word representations, even for unknown words, during training.

FastText follows an unsupervised approach akin to Skip-gram's for word embedding, incorporating character substrings. Compared to Skip-gram, FastText has a more extensive vocabulary, resulting in a model with more parameters. Additionally, calculating a word vector requires the addition of all the subword vectors, resulting in greater computational complexity. However, it can acquire superior vectors for uncommon and intricate words, including those not listed in the dictionary, by examining other words with a comparable structure.

Multilingual BERT

BERT (Bidirectional Encoder Representations from Transformers) [8] is a language model based on the encoder block of the Transformers architecture.

Unlike traditional language models that read the text in one direction, BERT reads text bidirectionally, capturing a deeper understanding of word meaning within a sentence. This methodology provides a richer context, which enhances the model's ability to comprehend the meaning of words within a sentence.

Another notable feature of BERT is its pre-training technique, which involves randomly masking particular words in the text and training the model to anticipate them via the surrounding context. This training method, the Masked Language Model (MLM), allows BERT to concentrate on context over word sequencing. Furthermore, another task has been integrated to enable the model to comprehend the connections between two sentences: "Next Sentence Prediction." This task requires the model to forecast whether sentence B follows sentence A.

Moreover, owing to its adaptable architecture, BERT can be effortlessly customized for a broad spectrum of NLP tasks like text classification, question answering, named entity recognition, and numerous others. This customization process is usually executed during the fine-tuning phase, when the model's weights are marginally adjusted, enabling it to perform optimally for that particular task.

M-BERT [25] (Multilingual BERT) is a variant of BERT that, during the pre-training phase, trained on corpora of 104 languages instead of exclusively on English. No other adjustments are made to the model's architecture. As a result, M-BERT uses the same architecture as BERT while being trained on a multilingual dataset. This approach enables M-BERT to identify the similarities and differences between languages, making it well-suited for zero-shot cross-language transfer tasks.

The fundamental concept driving M-BERT is that training the model on multiple languages allows cross-lingual knowledge to be utilized, enhancing performance on language-specific tasks. M-BERT can exploit shared meanings between languages, allowing inferences to be made in one language despite being primarily trained in another, mainly if there is substantial lexical overlap between the two languages.

E5

E5 [38] (EmbEddings from bidirEctional Encoder rEpresentations) is a general-purpose text embedding trained using contrastive learning.

Pre-trained language models like BERT and GPT can generate proficient text representations. Nevertheless, they are not optimal for retrieval and text-matching tasks, where single-vector text embeddings are more functional regarding efficiency and adaptability. To enhance sequence-level text representations, contrastive learning is the favoured approach for improving text embeddings from text pairs.

To achieve optimal performance, it is essential to consider both the quality and diversity of data used for training. This model was trained on CCPairs (Colossal Clean Text Pairs), a specially curated dataset that combines various web-based resources. Using this dataset ensures the model's adaptability to multiple tasks while producing consistently good results.

Each element of the dataset has a query-passage pair (q,p). The prefix "query:" precedes the query, and "passage:" precedes a sequence of words ranging from a short sentence to a lengthy paragraph. Contrastive pre-training aims to train the model to distinguish between relevant and non-relevant pairs. The scoring function is the cosine similarity between the two embeddings generated by a shared bi-encoder, whose size depends on the chosen E5 model. Our tests utilize the E5-large, based on the bert-large-uncased-whole-word-masked model.

Finally, the model is fine-tuned with annotated data to enhance its performance by incorporating human knowledge. The datasets utilized are NLI (Natural Language Inference) [10], MS-MARCO [1], and NQ (Natural Questions)[15].

This enabled the model to generate excellent general-purpose text embeddings without the need for fine-tuning, as it exhibited outstanding off-the-shelf performance in contrast to considerably larger models.

Text-emdedding-ada-002

Text-emdedding-ada-002 (TEA-002) is a general-purpose text embedding released by OpenAI in late 2022. This model can incorporate all the embedding models that OpenAI previously released and improve the performance in each task, such as search, similarity, and retrieval. In addition to enhancing performance, this model has expanded the input size and limited the output length, making the embeddings excellent for even very long text sequences while still achieving vectors with low dimensionality.

Since we are dealing with a closed model, details have yet to be released regarding the type of training and datasets used. The significant advantage of using this model is that it can be employed as a service, with dedicated APIs leading to zero infrastructure and maintenance costs. Furthermore, costs would be reduced over time, and there would be no dependence on potential malfunctions to OpenAI proprietary systems. The costs of this embedding model remain low at 0.00001€ per 1k token. If we estimate that a document page generally includes 8k tokens to reach 1€, we must convert 10.000 document pages.

Furthermore, converting documents into vectors would only incur a one-time expense. The only recurring cost would be related to the conversion of user queries.

2.1.3. Vector Database

The database is a fundamental component in the architecture of any computer system, particularly when it comes to machine learning or artificial intelligence models. Its effectiveness and efficiency directly affect the model's performance, particularly regarding response speed.

Vector databases are a type of non-relational database that utilizes vectorized data representation. In contrast to conventional keyword-based databases, vector databases enable searches based on the data's semantic meaning. Instead of seeking an exact match, they locate matches about the context or inherent sense of the data.

This attribute makes them notably fitting for similarity search or classification functions. In addition, vector databases can easily store associated metadata, such as file names, resource links, or contained images, due to their non-relational structure.

Redis is one well-known database for its exceptional performance. With its in-memory architecture, Redis offers rapid response times for both reads and writes, making it suitable for tasks that need processing large amounts of vector data, like similarity searches.

Redis supports two vector indexing types. Flat and Hierarchical Navigable Small Worlds (HNSW).

The flat index conducts a brute-force search, examining every vector in the database to locate the most comparable one to the query vector. Though simple, this technique can be sluggish when handling extensive data. On the contrary, HNSW (Hierarchical Navigable

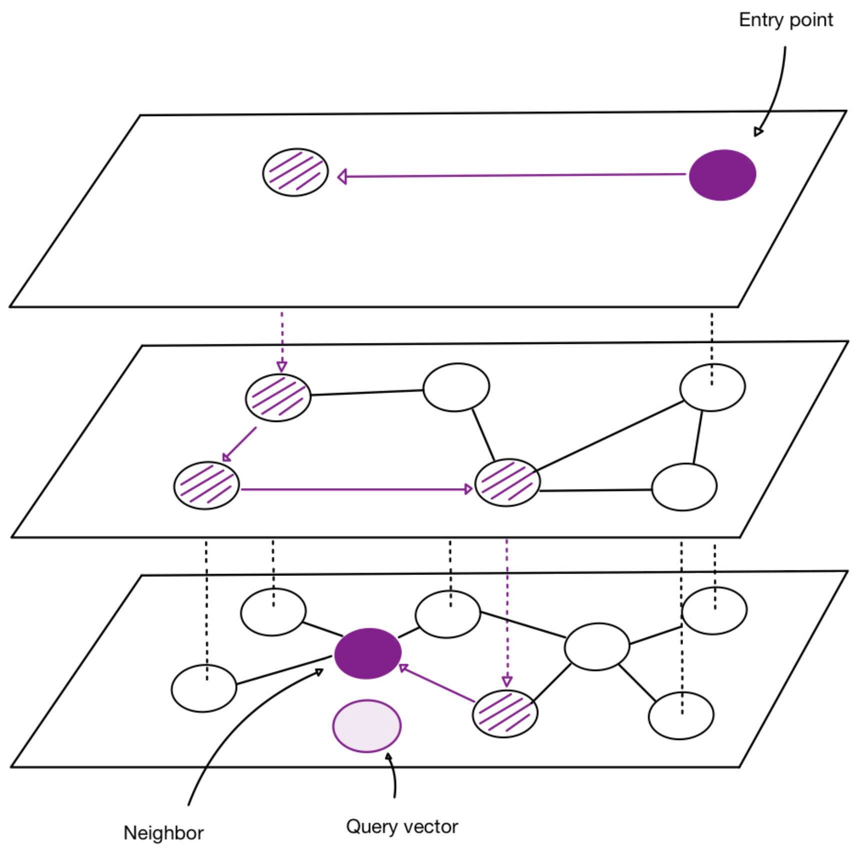


Figure 2.1: Search in Hierarchical Navigable Small World

Small World) constitutes an indexing and searching algorithm erecting a hierarchical vector architecture. This permits similarity searches to be executed much more efficiently than the flat indexing method, particularly in the case of large datasets. The HNSW traverses this hierarchical structure to promptly identify the most similar vectors, thus considerably diminishing the number of comparisons required.

Redis's in-memory performance is celebrated for its exceptional speed in read and write operations. This speed is especially beneficial for vector operations, such as similarity search.

Redis's scalability is another prominent feature when using it in this context. Redis can be easily scaled as datasets grow to manage larger workloads, ensuring steady response times and optimal performance.

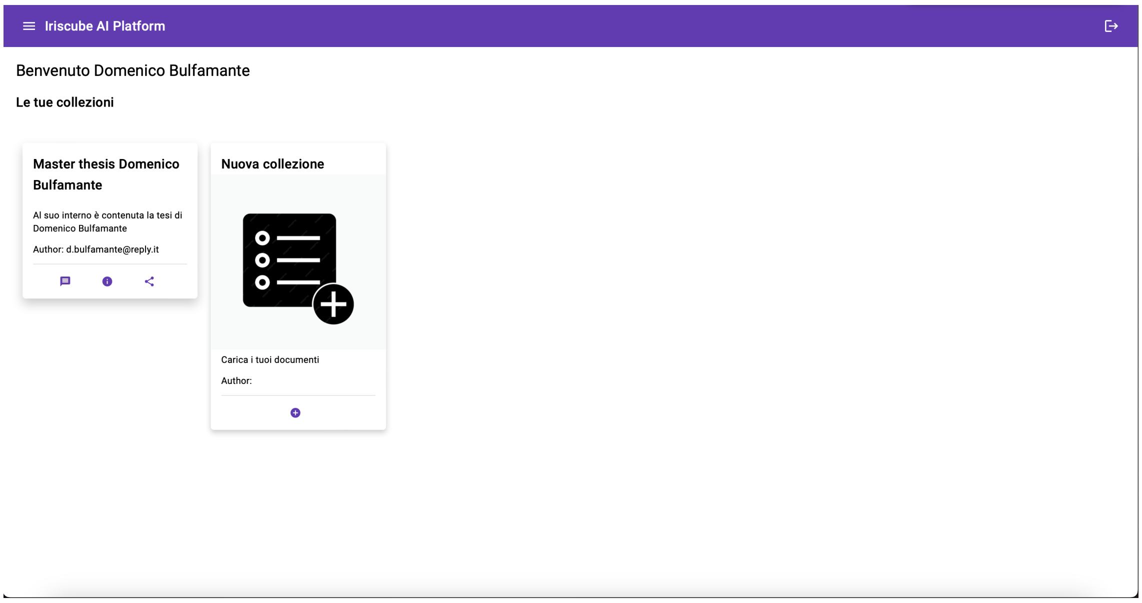
2.2. Application

2.2.1. Front-end

The web application was created using Angular, a widely used framework for developing single-page web applications. The preference for Angular was due to several inherent features of the framework. In particular, Angular offers a sophisticated dependency injection system that facilitates dependency management and provides the application with marked modularity, making it more adaptable to test and verification operations. In addition, Angular prioritizes an architecture underscored by componentization. This approach enables the creation of user interfaces that are both reusable and manageable. Furthermore, Angular boasts an incredibly adaptable built-in routing system, which facilitates navigation between different views and allows for the developing of highly complex single-page applications.

The application has two main features: the chat function and the creation and management of a collection of documents.

Creating a collection of documents is initiated by clicking on the "Crea Collezione" button shown in Fig. 2.2. At this point, the user can choose which documents to include in the collection and customise the collection by assigning it a specific name and an illustrative image. A noteworthy attribute of this system is the capability to specify the visibility of collections, allowing for the creation of both public and private collections. This function enables users to share collections without duplicating them, thus eliminating the redundancies of having the same collection multiple times.



Benvenuto Domenico Bulfamante

Le tue collezioni

Master thesis Domenico Bulfamante

Al suo interno è contenuta la tesi di Domenico Bulfamante

Author: d.bulfamante@reply.it

Nuova collezione

Carica i tuoi documenti

Author:

Figure 2.2: Home page section, where the list of user document collection are displayed

Fig. 2.3 illustrates the chat section, an interface through which users can ask questions about the selected collection. The decision to incorporate a chat feature is particularly astute in this context: it offers an intuitive and familiar interaction, simplifying the search for information and making it more accessible for the user. A vital feature of the chat-integrated bot is its capacity to understand the conversational context. This enables it to provide appropriate answers to subsequent or follow-up questions.

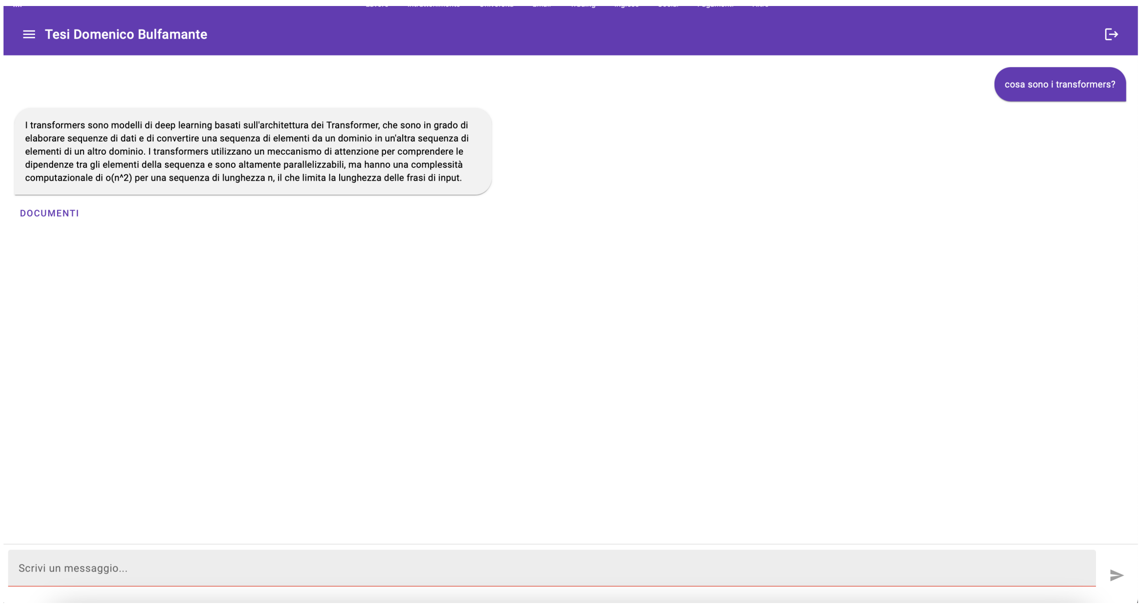
Registration and login processes were also incorporated, as illustrated in Fig. 2.4

In conclusion, a noteworthy function permits the user to recognize the source documents utilized to create a specific response. This is achieved through a straightforward interaction, by simply clicking at the "Documenti" that is present in Fig. 2.3. This aspect is of significant value concerning transparency and reliability: it enables the user to comprehend the derivation of the information presented, providing the opportunity to consult the authentic source directly for further examination or verification.

2.2.2. Back-end

The application's back-end compartment utilizes two separate databases: MongoDB and Redis. Although the explanation of Redis' associated features and motivations has been covered extensively in 2.1.3, it is necessary to go into more detail regarding why MongoDB was chosen.

MongoDB is a NoSQL database that is explicitly configured for a document-based ap-



cosa sono i transformers?

I transformers sono modelli di deep learning basati sull'architettura dei Transformer, che sono in grado di elaborare sequenze di dati e di convertire una sequenza di elementi da un dominio in un'altra sequenza di elementi di un altro dominio. I transformers utilizzano un meccanismo di attenzione per comprendere le dipendenze tra gli elementi della sequenza e sono altamente parallelizzabili, ma hanno una complessità computazionale di o(n^2) per una sequenza di lunghezza n, il che limita la lunghezza delle frasi di input.

Figure 2.3: Example of chat, on the right the user question and on the left the LLM response

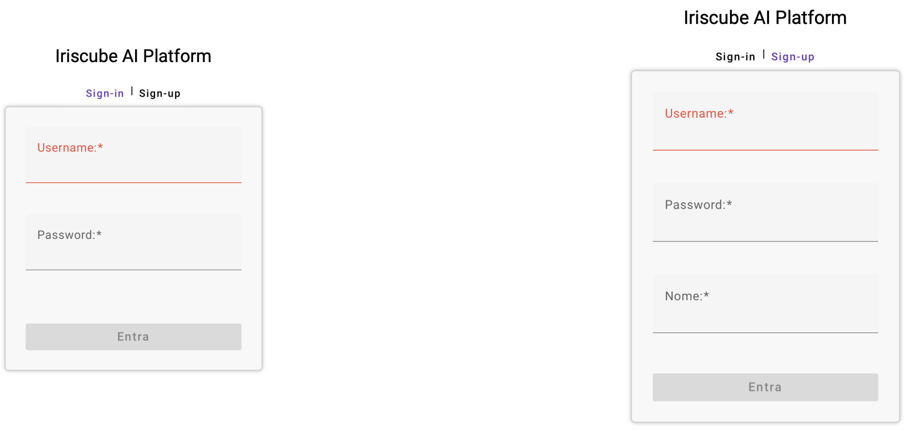


Figure 2.4: Login and registration process, respectively on the left and on the right.

proach. The adoption of MongoDB for registration and authentication processes was influenced by several inherent features, including its schema flexibility. Unlike traditional relational databases, MongoDB allows for easy modification and addition of new fields without imposing a rigid schema. This elasticity ensures a more straightforward adaptation to evolving application needs. In addition, MongoDB was intentionally developed for horizontal scalability, ensuring efficient handling of escalated workload due to user expansion. Its document-oriented architecture facilitates swift response times, particularly in use cases where read operations are more frequent than write operations, such as registration and login processes. Notably, the nature of the application does not require intricate transactions or complex relationships between tables, situations in which a relational database might be more suitable.

Regarding the exposure of APIs to the front end, the Python-based micro-framework Flask was utilized for web application development. Flask has several notable attributes. Its innate simplicity and lightness give developers enormous design freedom whilst not imposing a predetermined structure. Equally intuitive and quick to learn, Flask is an appealing option for those proficient in Python. Despite being a "micro" framework, Flask's high extensibility enables customization to suit project-specific requirements.

The APIs exposed are:

- login: allows users to authenticate into the application by entering username and password.
- registration: allows new users to sign up for the application by providing their details, such as their username, email address, and password. Once registration is complete, the user can log in to the application using their credentials.
- collection creation: enables users to create personalized document collections under defined parameters. These parameters include the name, description, image, and documents included in the collection. The visibility of the collection can be set as private or public.
- response generation: is used within the chat to generate the model response. When a user formulates a question or query, the system processes the request and provides a relevant response based on the documents in the selected collection. It also returns the documents used to generate the response.
- retrieving collections by user: allows the application's home screen to be populated with the collections accessible to the user.
- sharing a collection: allows users to share their collections with other users.

2.3. Large Language Model

Large Language Models (LLMs) represent a subcategory of deep neural networks, primarily based on the innovative transformer architecture. LLMs have transformed the field of

Natural Language Processing (NLP) due to their exceptional abilities in comprehending, producing, and manipulating natural language.

The Language Models are trained using the technique of "next word prediction." Essentially, a model is provided with a sequence of words and is then trained to predict the subsequent word in the sequence. This learning process enables the model to comprehend and recognize the subtle relationships and statistical patterns that exist within human language. The term "large" used to describe such models is entirely deliberate. It pertains to the considerable magnitude of data employed throughout the training phase, which often encompasses billions of words, as well as the intricacy and size of the model itself. LLMs possess language processing capabilities that often rival those of human beings concerning accuracy and fluency.

Their use in our application is focused on generating relevant responses based on text provided as context. Thanks to their deep language understanding, they can produce coherent and contextualized answers.

2.3.1. Finetuning vs prompt engineering

"Finetuning" and "prompt engineering" are two approaches used to adapt and optimize language models, particularly Large Language Models (LLMs), to specific tasks.

"Finetuning" represents an approach in which a model that has already undergone preliminary training on a large dataset ("pre-training" phase) is subsequently refined by further training on a smaller dataset that is specific to a particular domain or task. This methodology allows the model to assimilate the specifics of the new operational context while retaining the wealth of information acquired in the pre-training phase. While "finetuning" offers the advantage of being able to adapt the model to very specific scenarios, thereby greatly improving its performance, it also has critical issues. Indeed, training these models requires significant computational resources and, in the presence of limited data sets, can reduce overall performance and expose the model to the risk of overfitting.

In contrast, "prompt engineering" focuses on the curation and optimization of the prompt or the initial input in the form of an instruction or question given to the model to guide its response. This technique requires no additional computational resources for training and offers considerable flexibility, allowing the model to be adapted to different tasks without changing its structure. However, finding the ideal prompt may require iterative experimentation and may not guarantee the same effectiveness as 'fine-tuning' in some contexts.

In our case, prompt engineering was chosen for cost and versatility reasons: it would be inefficient to fine-tune each user-generated dataset, both in terms of time, because fine-tuning an LLM is not fast, and because we would have to save all the different models ad hoc.

Prompt engineering also allows for greater speed in adapting the model to its intended purpose. This is helped by the fact that these models are very large and already have extensive language knowledge and excellent word-processing skills. With this in mind, by providing the model with the necessary information through the prompt, a high level of performance was achieved, fully meeting the requirements of the project.

2.3.2. Model choice

As described in previous chapters, LLMs offer outstanding performance in natural language understanding and text generation. However, training such models is prohibitive, making the endeavour accessible only to a few large companies with considerable financial resources. Leading technology companies such as Google, Meta, Microsoft, and OpenAI have invested heavily in the research and development of these models.

To date, the LLM landscape is divided into open-source and closed-source models. Many open-source models are based on architectures such as LLama [33] or LLama 2 [34] released by Meta. These models have provided a solid basis for further experimentation and optimization to perform specific fine-tuning, such as Alpaca [32] or Vicuna [4]. These fine-tuned models perform very well but need to match leading closed-source models such as Bard or GPT4.

One of the main challenges of using LLMs is their size, which requires significant hardware resources, particularly in terms of GPU memory. Although techniques such as quantization exist to reduce the memory footprint of models, these solutions can compromise performance and response time.

Fortunately, there are now APIs available on the market that enable the use of models without the need to load them locally. This means that machine management and maintenance are already included in the cost, so there is no need to worry about them. Another advantage is the time required to start using the service. In this case, it is zero, unlike an open model. With an open model, the time required for fine-tuning must be calculated. The performance is also very high; the openAI models are the best on the market.

Privacy is maintained in this case because the services of OpenAI are used through Azure, avoiding severe security issues for the company. In this case, the model chosen is OpenAI's

GPT3.5-turbo. The GPT4 version is not used because it is 50 times more expensive, as shown in 2.1 and is not actually included in the company Azure enterprise account.

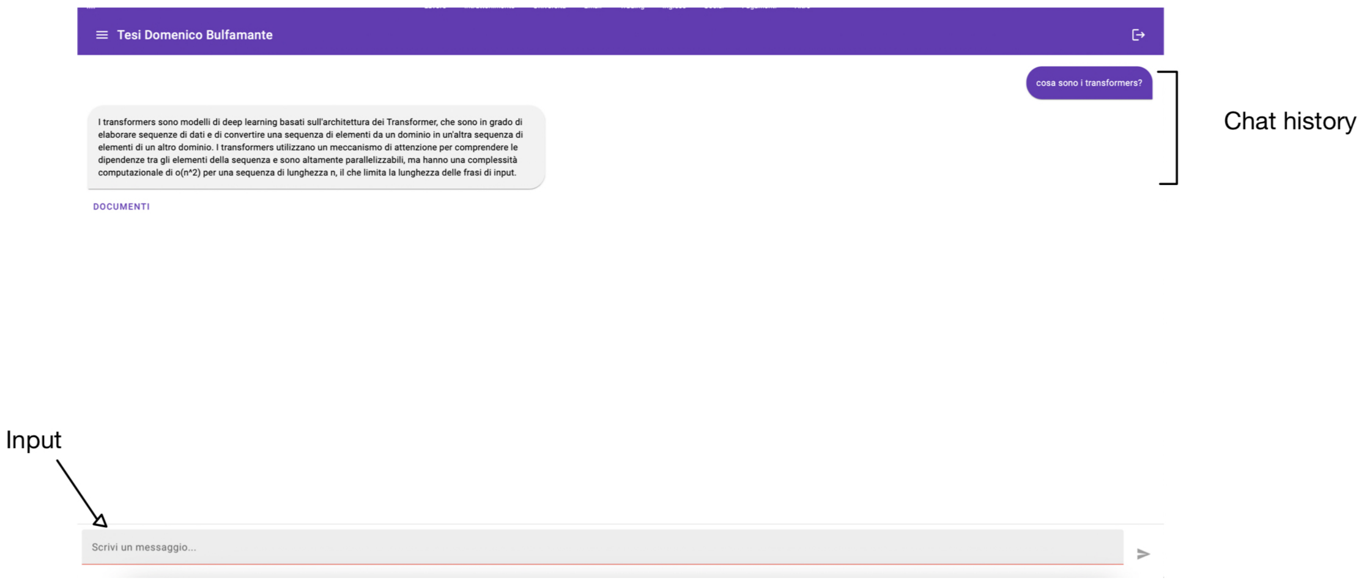
Model	Completion	Prompt	Context lenght	1M Tokens
gpt-4-0314	\$0.06/1 K	\$0.03/1 K	8192	\$1200.00
gpt-4-32k-0314	\$0.06/1 K	\$0.12/1 K	32768	\$1200.00
gpt-3.5-turbo	\$0.002/1 K	\$0.002/1 K	4096	\$4.00

Table 2.1: API pricing for GPT-4 (8K, 32K context) and GPT-3.5-turbo (ChatGPT API)

2.3.3. Conversational Memory

Large Language Models are stateless; they are not designed to remember previous output. Each input is treated as if it were independent of previous inputs. For example, GPT3 has required additional fine-tuning to handle chat, making it a chatbot that can consider not only the input and context it is given but also previous interactions.

Conversational memory allows a chatbot to answer multiple questions like a chat conversation.



cosa sono i transformers?

I transformers sono modelli di deep learning basati sull'architettura dei Transformer, che sono in grado di elaborare sequenze di dati e di convertire una sequenza di elementi da un dominio in un'altra sequenza di elementi di un altro dominio. I transformers utilizzano un meccanismo di attenzione per comprendere le dipendenze tra gli elementi della sequenza e sono altamente parallelizzabili, ma hanno una complessità computazionale di o(n^2) per una sequenza di lunghezza n, il che limita la lunghezza delle frasi di input.

Figure 2.5: Main components of a chat

In Fig. 2.5, we can see the two parameters: history and input. The history includes either the user's previous question or the model's previous answer. There is one case where the history is fundamental: when the user asks a follow-up question. This type of question is helpful to expand or clarify a previous answer.

The GPT3.5-turbo API is designed to receive the chat history parameter. In classic chatbot implementations, sufficient performance is obtained with this parameter. However, in the presented architecture, the information needed to answer the user question and chat history are added within the prompt. After testing, the model could no longer optimally handle the conversational memory because the inputs were processed independently. A possible cause of this problem is that the GPT model cannot process such large inputs simultaneously with the chat history. The information fed into the prompt is extensive, about 1000 tokens per input.

What is done is to create a new question that integrates the information from the previous chat. In fact, for each question, two API calls are made to the generative model:

- Summarise for the chat: the model is asked to create a new question that enriches our question using the chat history.
- Generate answer: The model is asked to generate the final answer using the new question and the information retrieved from the retriever.

This solution allows us to handle follow-up questions better. It allows the user to ask more specific questions more intuitively.

2.4. Deployment

The deployment process is a defining stage in an application's lifecycle, as it determines how and when users will access and interact with the application.

The application is hosted directly on a local machine in the company's offices. This selection offers numerous benefits, firstly in terms of security: accessibility of the application is restricted to the corporate network, creating an additional layer of protection against potential external threats. This approach also eradicates the supplementary costs of external hosting, allowing for more sustainable economic management.

Another significant consideration is ensuring uniformity between the development and production settings. Containerization is chosen to ensure the dependable and consistent operation of the application in both scenarios. This approach, which involves using technologies such as Docker, enables the application and its dependencies to be enclosed within an isolated environment, or "container." This ensures that the program runs identically in different environments, minimizing problems that may arise due to differences in configuration or software versions used.

Given the requirement to manage multiple containers, Docker Compose has been used as an adjunct tool to Docker. It permits an individual docker-compose.yml document to specify the complete application structure, streamlining service administration, network setup and environmental customization. This tool also eases scalability, dependency handling between services, and container communication.

Gunicorn is used to effectively handle HTTP requests, a WSGI HTTP server explicitly designed for Python. The pre-forked process model is one of Gunicorn's key benefits, enabling it to effectively process many requests concurrently without incurring the cost of creating a new process or thread for each request.

3 | Experimental setup and Results

This chapter provides an objective analysis of the experiments conducted to assess the efficiency of the system components. An in-depth examination of the setup, datasets, and performance metrics is provided. The aim is to present a logical flow and causal connections between statements, presenting the architecture used in the creation of the application in a comprehensive manner. The chapter's primary objective is to prepare the reader for accurate interpretation of results following an understanding of the research process.

3.1. Experimental settings

Most of the experiments related to the final architecture were carried out on a company computer, except for preliminary tests to determine a text embedding model. Such preliminary tests were conducted with Google Colab, a pre-configured Python environment offered by Google. This platform eradicates the requirement of installing dependencies or downloading models and datasets onto one's own device, as well as offering access to powerful GPUs for higher performance.

3.2. Dataset

In this section, the datasets employed for evaluating the text embedding models and analyzing the proposed architecture will be discussed.

Initial experiments were conducted to assist in selecting the most appropriate text embedding model. The aim was to showcase the substantial progress achieved by the latest models, especially compared to those used only four years ago. Our focus was mainly on the 'retrieval' task, whereby the primary goal involves identifying pertinent documents. During this procedure, the query embedding is compared with that of the documents, with the cosine similarity calculation for each query-document pair. Following this, the documents are sorted based on their score. However, our ability to conduct evaluations in this area was limited due to the need for Italian language databases. In summary, evaluating the retrieval-focused embedding models allowed us to obtain a comparative framework that accurately reflects our application scenario.

Subsequent experiments were designed to assess the architecture as a whole. It is important to emphasize that the obtained results are closely tied to the specific architecture; performance may differ by altering its components. There is no dataset that perfectly emulates our use case concerning Retrieval Augmented Generation (RAG). To attain an exact performance evaluation, datasets were curated based on actual questions and documents utilized by our platform's users.

Text embedding

In document retrieval, each dataset comprises a corpus, queries, and a mapping of each query to each document in the corpus. This mapping indicates the relevance of the document to the query.

The two selected datasets are specific to the argument retrieval task, which seeks to rank argumentative texts based on their relevance to a specific query.

ArguAna Counterargs Corpus [37] focuses on identifying the best counterargument about a given argument. This corpus contains pairs of arguments and counterarguments gathered from an online debate portal. The original test partition's arguments are used as our queries, named Q. The dataset comprises 10,080 samples of an average length of 1052.9 characters.

Task 1 of Touché-2020 [3] involves the acquisition of conversational topics. The test queries, Q, use the conclusion as the title and the premise as the presented topics, T. The dataset contains 382,594 samples with an average length of 1,720.1 characters.

To evaluate the performance of various models, the implementation available on MTEB [22] has been utilized.

Retrieval Augmented Generation

To evaluate the effectiveness of the implementation, custom datasets were utilized that contained the user's question, the model's answer, and the model's context to generate the response. The datasets consisted of around fifty questions classified into two sets for experimental purposes.

The questions were divided into two dataset categories to assess the difference between vector database index types. The first category included a significant collection of around 20 documents, while the second comprised just one document.

For each index category, two databases were created: one using the flat index and the other using the hnsw index.

To determine the appropriate number of tokens to include in each text chunk, different databases were created containing 300 and 600 tokens. Using a limit of 300 tokens for a single document resulted in 17 elements, while with 600 tokens, only 10 were found. When collecting 20 documents, a limit of 300 tokens yielded 432 elements, while 600 tokens resulted in 302 elements.

3.3. Evaluation Metrics

This section explores the various metrics used to fully understand the text embedding models in the retrieval task. The effectiveness of the architecture in producing responses that accurately and consistently align with the retrieved documents will also be measured. Additionally, the actual cost of using each model will be appraised to make a more informed decision about its usability. Lastly, the speed and performance of the final platform's architecture using collections of documents will be evaluated to gain a comprehensive understanding of the technical decisions made.

Text embedding

To assess the performance of text embedding models in the retrieval task, a selection of metrics has been identified to evaluate recommendation systems. These metrics measure how well the model has ranked the relevant elements. These ranked elements correspond to the pair (q,d), whereby q represents the query and d represents the document related to the context. A relevance score is assigned to each of these pairs, reflecting the degree to which the document is pertinent to the given query. In this scenario, text embedding models must be able to map every (q,d) pair to a value indicating the relevance of the document to the query.

Various metrics have been adopted to evaluate the effectiveness of this sorting and are frequently utilized in recommender systems. These metrics, usually identified by "@k", relate to the number of elements assessed in the relevance evaluation.

Precision@k is a metric used to measure the proportion of relevant items included in the top k recommendations. This metric helps to evaluate the quality of recommendations by measuring how many of the items suggested are actually relevant. The formula for precision@k is as follows:

$$\text{precision} = \frac{|\,\{\text{ relevant documents }\} \cap \{\text{ retrieved documents }\}\,|}{|\,\{\text{ retrieved documents }\}\,|}$$

Recall@k, on the other hand, measures the fraction of relevant items that are identified in the top k recommendations. It helps to determine how well the recommendations cover all the relevant items. The formula for recall@k is as follows:

$$\text{recall} = \frac{|\,\{\text{ relevant documents }\} \cap \{\text{ retrieved documents }\}\,|}{|\,\{\text{ relevant documents }\}\,|}$$

MAP@k is a measure that calculates the average precision for each relevant item position in the list of recommendations. It helps to determine how many of the suggested results are relevant and are positioned high in the list. The AP for a user can be calculated using the following formula:

$$AP = \sum_{k=1}^{n} p(k)\,\text{rel}(k)$$

Here, $p(k)$ represents the precision@k, n is the total number of items in the recommendation list, and $\text{rel}(k)$ is an indicator function that is worth one if the item at position k is relevant, otherwise zero. The MAP is calculated by averaging the AP over all users:

$$MAP = \frac{1}{|U_{\text{all}}|} \sum_{u=1}^{|U_{\text{all}}|} AP(u)$$

However, a limitation of this metric is that it only considers accuracy without assessing the relevance of an item. To overcome this limitation, the NDCG@k (Normalised Discounted Cumulative Gain) metric is introduced. The Cumulative Gain (CG) measures the sum of the gains for each item. In this context, the gain is defined as relevance, so it is worth one if the item at position k is relevant; otherwise, it is zero.

$$CG(k) = \sum_{i=1}^{k} G_i$$

Since the CG does not take order into account, the DCG (Discounted Cumulative Gain) is used to penalise highly relevant items placed at the bottom of the list:

$$DCG(k) = \sum_{i=1}^{k} \frac{G_i}{\log_2(i+1)}$$

However, the DCG depends on the length of the list. To solve this, the IDCG (Ideal Discounted Cumulative Gain) is introduced, where each relevant item is placed at the beginning of the list:

$$IDCG(k) = \sum_{i=1}^{|I(k)|} \frac{G_i}{\log_2(i+1)}$$

Here, $I(k)$ represents the ideal list of items up to position k, and $|I(k)| = k$. The NDCG normalises the DCG score for the IDCG, guaranteeing a value between 0 and 1:

$$NDCG(k) = \frac{DCG(k)}{IDCG(k)}$$

In the analysis, the value of k is set at 10, close to the number of documents returned by the system. In addition to these metrics, speed measures the time the model takes to generate the list, and cost evaluates the economic efficiency of adopting one model over another.

Retrieval Augmented Generation

To properly evaluate an RAG, one has to assess both the part that deals with document retrieval and the part that deals with document generation. This is a complex task as it requires the evaluation of these elements separately and together. To properly assess the RAG architecture, it is necessary to evaluate both the document retrieval and generation components. Such an approach enables the determination of the quality of the model and the identification of any areas for improvement that exist to achieve enhanced performance.

Many classical metrics exhibit a low correlation with human evaluation, particularly Challenges that arise when the model is applied to data that differs significantly from that used in classical benchmarks, leading to a noteworthy data shift. As a solution, employing an LLM for result evaluation has shown promise in recent times.

Ragas [9] is a framework that allows the proposed QA pipeline to be evaluated from different aspects.

- evaluates retrieval: context relevancy and context recall offer a way to assess the effectiveness of our retrieval system.
- evaluate generation: faithfulness that measures hallucinations and answer relevancy that indicates how relevant the generated answer is to the question asked.

The harmonic mean between the four is performed to obtain a single score, which gives the ragas score a single measure to evaluate the QA performance of our system. Most importantly, this does not require labelled data for some measures. One can utilize extracted data directly in a real-life scenario within the system.

To enhance understanding of the results, a description of each of the metrics follows:

- faithfulness: measures how accurate the response is with the factually generated context. This measure requires two steps. In the first, given the question and the generated answer, the framework uses an LLM to create statements that the generated answer creates. In the second step, given the list of statements and the context, an LLM verifies that the context supports the given statement. The number of correct statements is summed and divided by the total statements generated by the response.
- answer relevancy: measures how relevant the answer to the given question is. Given the generated answer, an LLM generates probable questions to this answer. Finally, the similarity between the user's question and the generated questions is calculated
- context relevancy: measures how much noise the defined context carries. Given the question, the LLM identifies which context sentences are needed to answer the question. The ratio of necessary to total context sentences generates the score.
- context recall: measures the retriever's ability to obtain all the information needed to answer. The ground truth answer (not available in our case) and an LLM are used to check that each statement contained in the answer is present in the context. If not present, it indicates that the retriever cannot locate the correct information.

In addition to these measures, the velocity with which the vector database can furnish the fitting context will be assessed. This will give us a gauge of the vector database's index type's impactfulness.

3.4. Results

This section will present and thoroughly examine the results of the conducted experiments. This will explicitly explain the selected components and parameters employed within the final architecture. Ultimately, the exceptional performance of the complete architecture will be showcased, demonstrating its ability to generate pertinent responses to the posed queries.

Text embedding

The experiments were conducted using the text embedding models previously discussed in section 2.1.2.

	p@10	**r@10**	**MAP@10**	**NDCG@10**	**eval time (s)**
FastText	0.30	3.73	1.81	2.34	17.08
MBert	2.01	20.13	9.97	12.35	173.81
E5	8.27	82.94	46.15	54.91	498.86
TEA-002	8.64	86.42	48.28	57.43	235.36

Table 3.1: Performances of text embedding models on ArguAna Counterargs Corpus

	p@10	**r@10**	**MAP@10**	**NDCG@10**	**eval time (s)**
FastText	0.21	0.01	0.01	0.08	866.43
MBert	0.18	0.03	0.21	0.97	7935.45
E5	13.67	10.27	5.67	23.39	20449.49
TEA-002	14.4	18.34	4.81	18.76	10290.32

Table 3.2: Performances of text embedding models on Touché-2020

The hugging face models, exept text-embedding-ada-002 (TEA-002), used for this experiments are: fasttext-it-vectors (FastText), bert-base-multilingual-cased (MBert) and multilingual-e5-large (E5); the given models can be found on hugging face As demonstrated by the newest models, impressive performance can be achieved, but it comes at the cost of sacrificing considerable speed. Specifically, the E5 and text-embedding-ada-002 models bear close similarities regarding their results, particularly when considering the benchmark metric of NDCG@10, as previously noted. The economic and speed considerations determined the selection of TEA-002 (text-embedding-ada-002); it enables us to create embeddings within half the time required by E5.

Furthermore, the OpenAI model is an API service, offering low-cost options (0.00001€ per 1K tokens) without the added expense of infrastructure and maintenance. Should we opt for the E5 model as our primary choice, we would be required to purchase a GPU or pay for a server that offers it. Additionally, the model demands 14.6 GB of VRAM, necessitating a minimum of 16GB of video RAM, which can cost around 1000€, such as an RTX 4080. If we consider an alternative to Colab for testing purposes, the cost would be

0.2£ per hour of use. The calculation was based on a T4 with 15GB of VRAM, consuming approximately 2.08 tokens per hour (tokens cost approximately 0.10£ on Colab). It is important to note that the evaluation remains objective and unbiased regarding cost.

index	token lim	ragas score	context rel	faithfulness	answer rel	time (ms)
HNSW	600	10.72	3.90	**91.25**	81.60	**1.03**
HNSW	300	15.84	6.02	89.58	82.37	1.24
FLAT	600	14.42	5.44	82.78	**83.39**	1.12
FLAT	300	**16.93**	**6.50**	90.21	82.06	1.42

Table 3.3: Results of test performed on architecture with single-document collection

index	token lim	ragas score	context rel	faithfulness	answer rel	time (ms)
HNSW	600	17.10	6.78	64.17	80.45	**1.30**
HNSW	300	18.53	7.24	**86.96**	81.69	1.32
FLAT	600	11.97	**7.51**	75.07	81.18	1.45
FLAT	300	**18.75**	7.34	86.25	**82.42**	1.60

Table 3.4: Results of test performed on architecture with multi-document collection

OpenAI's text-embedding-ada-002 model was chosen due to the platform's limited traffic and its excellent document retrieval performance.

Retrieval Augmented Generation

The results obtained are presented in the following tables. Table 3.3 displays the outcomes from the dataset with a single document, whereas Table 3.4 illustrates the findings from the dataset consisting of twenty documents. It is essential to note that the model was posed with identical questions in single and multi-document configurations. This means all questions were included in a single document, part of the twenty documents.

One clear finding is that, despite differences in dataset size, the time it takes to retrieve similar documents remains unchanged, mainly in milliseconds. This implies that to grasp the full potential of the HNSW index, it may be necessary to have a significantly more extensive collection of items.

Examining the other results, the most effective configuration with this relatively small number of documents is to employ a FLAT index with a split limit of 300 tokens. It is certain that conducting a greedy search to compare all documents and find the optimal match results in improved contextual relevancy. Indeed, this measures how relevant the context is to the query, and having the highest results guarantees the best results in terms of similarity.

It is worth noting that a limited number of tokens in the context can yield superior performance, which is an intriguing outcome. This may be because lengthy passages impede the optimal implementation of semantic meaning by the text embedding model, particularly if they contain multiple concepts.

4 | Conclusions

Corporate information management poses a challenge for many organisations in the era of digitisation. A system centralising and consolidating information from multiple sources has become increasingly necessary. This dissertation proposes a platform to address this requirement by utilising the potential of Large Language Models (LLM).

Integrating information retrieval techniques, LLM has demonstrated impressive accuracy levels in retrieving and analysing information from particular contexts, as evidenced by the results in table 3.3 and table 3.4. The faithfulness and relevance of the answers provided in the document-specific question set were notably high. This method allows for managing the information accessed by the LLM while maintaining high data security by storing it in a protected dataset instead of the model.

The chat-based interface of the platform enhances accessibility and ease of use, thus encouraging employee adoption. Creating and sharing private or public collections has enhanced the user experience, ensuring that information access is always current and centralised.

Another critical factor was the integration with OpenAI services via Azure. This expedited development and relieved the company from managing the models directly, allowing for concentration on the platform's principal features. Besides, the safety of the information treated by OpenAI models is ensured by keeping the company's data confidential, thanks to complying with Azure's GDPR security standards.

The results in tables 3.1 and 3.2 support the hypothesis that modern text embeddings have significantly improved in recent years. Previous models were much less effective, particularly in the retrieval task, underscoring the significance of specific training.

However, the study also has limitations. The first issue relates to the conversion of documents from Word to text. Currently, no Python libraries can extract all document contents effectively, including document comments, which are crucial for AFU. These comments often contain corrections or additional information. Furthermore, context relevancy significantly impacted the RAG test results, highlighting the requirement for further refining the text embedding procedure or considering alternative methods in creating information chunks.

In conclusion, the proposed platform has been successful. Its effectiveness and ease of use have been confirmed by the increasing use by Iriscube Reply employees. This work represents a significant step towards more efficient and intuitive business information management.

4.1. Future work

Within this section, a discussion of ideas for the improvement of the results obtained.

Important addition involves implementing an advanced **Document reader** capable of saving images. This would enhance the presentation of documents to the user. Integrating a description generated by an LLM would also enrich the content by incorporating information that may have been omitted otherwise. Optimizing the Document reader could include extending its compatibility with various formats, including Excel. Handling comments in Word documents can be challenging as some files are edited through comments rather than directly. A custom reader version would be necessary for this functionality as there is a lack of frameworks supporting such handling.

The selection of the **Vector database** is paramount to the model's overall performance. Examining and selecting the best-suited database could lead to faster and more precise document rendering. Various options are available in the market, including Picone or Milvus. Nevertheless, employing an external non-containerizable database such as Redis entails potential risks and has not been considered. The ideal course of action would involve the capability to modify files within a collection, which guarantees universal access to the latest version, obviating the need to remove and establish comparable collections.

It may be worth investigating alternative options, particularly open-source ones, and considering the solutions offered by **Text embedder** and **LLM**. Research into embedders ought to be extended whilst considering the requirement for a GPU or a lightweight model, such as the all-MiniLM, which also supports Italian. As for LLMs, a possible experimental avenue could involve utilizing LLama 2, renowned for its superb performance, albeit necessitating a specific configuration for deployment purposes.

Incorporating new features in the **web application** could boost its value to employees. Presently, efforts are being made to synchronize the application with email, enabling employees to streamline the retrieval of information frequently contained in email correspondence.

Finally, to assess the effectiveness of the architecture in a practical setting, incorporating a tool like LangSmith may be beneficial. By combining it with Ragas, real-time performance monitoring can be enabled, creating logs that can be accessed on a designated platform, thereby simplifying the identification and resolution of any issues.

Bibliography

[1] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, et al. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

[2] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[3] A. Bondarenko, M. Fröbe, M. Beloucif, L. Gienapp, Y. Ajjour, A. Panchenko, C. Biemann, B. Stein, H. Wachsmuth, M. Potthast, et al. Overview of touché 2020: argument retrieval. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction: 11th International Conference of the CLEF Association, CLEF 2020, Thessaloniki, Greece, September 22–25, 2020, Proceedings 11*, pages 384–395. Springer, 2020.

[4] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL `https://lmsys.org/blog/2023-03-30-vicuna/`.

[5] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Third conference on applied natural language processing*, pages 133–140, 1992.

[7] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] explodinggradients. ragas. `https://github.com/explodinggradients/ragas`, 2023.

[10] T. Gao, X. Yao, and D. Chen. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*, 2021.

[11] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. In *Handbook for Automatic Computation: Volume II: Linear Algebra*, pages 134–151. Springer, 1971.

[12] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[14] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[15] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[16] N. Lambert, L. Castricato, L. von Werra, and A. Havrilla. Illustrating reinforcement learning from human feedback (rlhf). *Hugging Face Blog*, 2022. https://huggingface.co/blog/rlhf.

[17] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[18] C. Manning and H. Schutze. *Foundations of statistical natural language processing.* MIT press, 1999.

[19] H. Masataki and Y. Sgisaka. Variable-order n-gram generation by word-class splitting and consecutive word grouping. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 1, pages 188–191. IEEE, 1996.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[21] E. Millar, D. Shen, J. Liu, and C. Nicholas. Performance and scalability of a large-scale n-gram based information retrieval system. 2000.

[22] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022. doi: 10.48550/ARXIV.2210.07316. URL `https://arxiv.org/abs/2210.07316`.

[23] OpenAI. URL `https://platform.openai.com/playground`. Open AI Playground.

[24] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023. URL `https://api.semanticscholar.org/CorpusID:257532815`.

[25] T. Pires, E. Schlinger, and D. Garrette. How multilingual is multilingual bert? *arXiv preprint arXiv:1906.01502*, 2019.

[26] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[27] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[28] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[29] G. Salton, A. Wong, and C.-S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[30] ShayHill. docx2python. `https://github.com/ShayHill/docx2python`, 2019.

[31] C. Y. Suen. N-gram statistics for natural language understanding and text processing. *IEEE transactions on pattern analysis and machine intelligence*, (2):164–172, 1979.

[32] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[33] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: open and efficient foundation language models, 2023. *URL https://arxiv. org/abs/2302.13971*.

[34] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[35] A. M. Turing. Computing machinery and intelligence (1950). *The Essential Turing: the Ideas That Gave Birth to the Computer Age*, pages 433–464, 2012.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[37] H. Wachsmuth, S. Syed, and B. Stein. Retrieval of the best counterargument without prior topic knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 241–251, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1023. URL `https://aclanthology.org/P18-1023`.

[38] L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

[39] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

List of Figures

List of Tables

Acknowledgements

I am deeply grateful to Dr Petrea Tancau and Dr Tommaso Rosso for their continuous support and guidance. Their involvement in various projects I supervised was crucial. I must also thank Prof. Apiletti for his remarkable flexibility and valuable advice. I extend my heartfelt appreciation to those who have contributed silently in the background. This thesis is a testament to your support, guidance and confidence in my abilities.