LUISS

Master of Science in Data Science and Management

Course of Data Science in Action

Transformative AI for Readers: Leveraging NLP, Transformers, and Retrieval-Augmented Generation to Build an AI-Driven Book Companion

Prof. Alessio Martino

SUPERVISOR

Prof. Irene Finocchi

CO-SUPERVISOR

Adib Menchali

CANDIDATE

Academic Year 2023/2024

Abstract

This thesis presents the development of an advanced question-answering AI chatbot, leveraging state-of-the-art information retrieval technologies and machine learning models. The chatbot utilizes the Haystack framework to retrieve answers from the Google Books API, ensuring access to a vast repository of textual data. Additionally, the system integrates a recommendation engine that harnesses transformers to compute semantic similarity between books, providing users with relevant reading suggestions.

The research showcases the capabilities of modern machine learning techniques, particularly focusing on the transformative power of transformers from Hugging Face. It delves into various natural language processing methodologies, including named entity recognition and retrieval-augmented generation (RAG). Furthermore, the thesis includes fine-tuning processes of transformer models to enhance their performance in specific tasks.

By combining advanced NLP techniques, information retrieval, and recommendation systems, this work demonstrates the potential of contemporary machine learning technologies in creating sophisticated, intelligent systems. The findings underscore the importance of these technologies in developing practical applications that can understand and respond to user inquiries while offering personalized recommendations.

Acknowledgements

I would like to express my deepest gratitude to my research supervisor, Professor Alessio Martino, whose continuous guidance and invaluable expertise have been indispensable in shaping this thesis. Your support and insights have profoundly influenced my academic journey, and for that, I am eternally grateful.

To all the friends I've met along the way during this international journey, thank you for making this experience both memorable and enriching. Your camaraderie made every challenge surmountable and every triumph sweeter. You made me feel at home, and I will forever cherish the memories we created together.

Finally, to the light of my life, my full-time supporter, my mother. You are the serene answer to all my doubts. Your unwavering belief in me has been the foundation of my strength and the source of my inspiration. Your presence in my life has made all the difference, and my success is a testament to your enduring support and love.

Table of Contents

1. Introduction

Artificial Intelligence has been emerging in so many different aspects of our lives completely reshaping the world and rebuilding our day-to-day tasks by integrating technologies like machine learning and natural language processing to drive efficiency and innovation. This technological constellation unlocks unprecedented possibilities, transcending traditional boundaries in leveraging data, knowledge extraction, and task automation.

At its core, AI is founded on formal reasoning, which aims to mechanize human cognition and replicate logical inference artificially. Formal reasoning, in this context, refers to using logical rules and algorithms to make decisions and draw conclusions, mirroring the way humans reason. This principle underpins the development of AI systems, enabling them to process information, learn from data, and perform complex tasks autonomously.

AI's propensity to appear intelligent has roots tracing back to the 1960s, notably exemplified by Joseph Weizenbaum's creation of the ELIZA chatbot (Weizenbaum, 1983). Though basic compared to today's standards, it showed how AI could simulate human-like interactions and understanding using simple patterns and pre-written responses. Its introduction sparked widespread interest in AI and raised deep questions about how humans interact with computers. The "Eliza effect" refers to users seeing AI systems as more human-like, even though they simply follow pre-defined rules and algorithms.

As AI permeates society, it challenges conventional views of human capabilities and machine limitations. Its fusion with disciplines like data science and natural language processing heralds a new era of innovation, shaping how we interact with technology.

This thesis embarks on a journey to explore AI's convergence with machine learning and natural language processing in the realm of book-related inquiries. Through the

development of an AI-driven question-answering system and a novel book recommendation engine, we aim to uncover AI's potential in enriching our interaction with literary content. By leveraging formal reasoning principles and state-of-the-art methodologies, we seek to bridge the gap between human intellect and artificial intelligence.

In the realm of book-related domains, existing AI applications are harnessing the power of integrations with APIs such as the Google Books API to enhance user experiences and access to literary content. These integrations span across various platforms, each offering unique features and functionalities:

Goodreads, a popular platform for readers to discover, review, and recommend books, integrates with the Google Books API to access its wealth of user-generated data. AI applications leveraging this integration can tap into Goodreads' repository of book ratings, reviews, and reading preferences to provide personalized reading suggestions based on individual tastes and interests.

WorldCat, the world's largest network of library content and services, offers access to millions of bibliographic records. Integration with the Google Books API enables AI applications to tap into WorldCat's extensive database of library collections and holdings. AI-driven search tools can utilize this integration to facilitate access to a wide range of scholarly resources and academic literature available in libraries worldwide.

University Libraries, which have digitized their collections and made them accessible through online repositories, integrate with the Google Books API to provide access to scholarly literature and academic publications. AI-driven tools can assist researchers and students in discovering relevant literature, conducting literature reviews, and accessing scholarly resources for their academic endeavors.

1.1 Problem Statement

In today's digital age, access to vast repositories of information, including books, has become increasingly accessible. However, navigating this wealth of knowledge efficiently remains a challenge. Users often encounter difficulties in obtaining relevant information about books, whether it be searching for specific titles, understanding their content, or discovering new reads. Traditional search engines and library catalogs may provide basic information, but they often fall short in addressing nuanced queries or offering personalized recommendations.

Identification of the need for an AI system that can effectively respond to questions about books arises from these challenges. Such a system would not only streamline the process of accessing book-related information but also enhance the user experience by providing tailored responses and recommendations. However, developing such a system presents several challenges:

Natural Language Understanding: Understanding user queries in natural language poses a significant challenge due to the complexity and ambiguity inherent in human language. A robust AI system must be able to parse and interpret various forms of queries accurately.

Content Understanding: Extracting relevant information from books requires the AI system to comprehend the content effectively. This involves tasks such as named entity recognition, summarization, and sentiment analysis to extract key information and provide meaningful responses.

Data Availability and Quality: Access to comprehensive and high-quality data is crucial for training and evaluating AI models. However, book-related datasets may be limited in size and diversity, posing challenges in model development and evaluation.

1.2 Objectives

The primary goal of this thesis is to address the identified need for an AI system that can effectively respond to questions about books. To achieve this goal, the following objectives are outlined:

Development of an AI-powered Book Question-Answering System: The thesis aims to develop an AI system capable of understanding natural language queries about books and providing accurate and informative responses. This involves leveraging techniques from natural language processing and machine learning to build a robust question-answering pipeline.

Creation of a Book Recommendation System: In addition to answering specific queries, the thesis also aims to build a book recommendation system that can propose relevant books based on their similarity with other titles. This involves examining book metadata and leveraging modern NLP and machine learning techniques.

2. Literature Review

2.1. Natural Language Processing

Natural Language Processing (NLP) has seen significant advancements in recent years, particularly with the development of transformer models and the integration of various machine learning techniques. This review examines key literature on NLP and its associated methodologies, drawing from influential texts and recent articles to provide a comprehensive overview of the current state of the field.

NLP's evolution has been marked by significant milestones, particularly the introduction of transformer models such as BERT and GPT-3. These models have revolutionized the field by enabling machines to process and generate human-like text with remarkable accuracy. Transformer architectures, characterized by their self-attention mechanisms, have surpassed traditional models in tasks like translation, summarization, and question-answering (Kumar & Singh, 2024).

Natural Language Understanding and Natural Language Generation are two primary components of NLP. NLU focuses on interpreting and deriving meaning from human language, enabling machines to understand context, sentiment, and intent. NLG, on the other hand, is the process of producing phrases, sentences, and paragraphs that are meaningful from an internal representation (Khurana et al., 2023).

The significance of NLP permeates modern life, shaping interactions in sectors as varied as retail and medicine. Retail giants employ customer service chatbots, while medical professionals rely on NLP to interpret and summarize electronic health records. Smart assistants like Amazon's Alexa and Apple's Siri harness NLP to interpret user queries, while advanced models such as GPT-3 produce coherent prose across diverse subjects. In

the realm of information retrieval, Google leverages NLP to refine search results, while social media platforms employ it to identify and mitigate hate speech.

Despite remarkable progress, challenges persist. Bias and incoherence plague current systems, occasionally yielding unpredictable behavior. Yet, these hurdles offer opportunities for machine learning engineers to refine and advance NLP applications, increasingly integral to societal functioning.

In their book 'Natural Language Processing: Python and NLTK, Hardeniya et al. offer a comprehensive overview of different techniques used in NLP models. Let us go through some of the techniques that are fundamental in building our AI.

2.1.1. Tokenization

Hardeniya et al. define tokenization as the process of breaking down raw text into smaller, meaningful units known as tokens. These tokens, which encompass words, phrases, or even entire sentences, play a crucial role in aiding data scientists to grasp the context during the development of NLP models. By breaking down text into tokens, tokenization facilitates the conversion of unstructured data into a numerical format, which is essential for machine learning applications. The authors emphasize the fundamental role tokenization plays in enabling machines to comprehend and process raw text. Figure 1-1 illustrates the tokenization process.

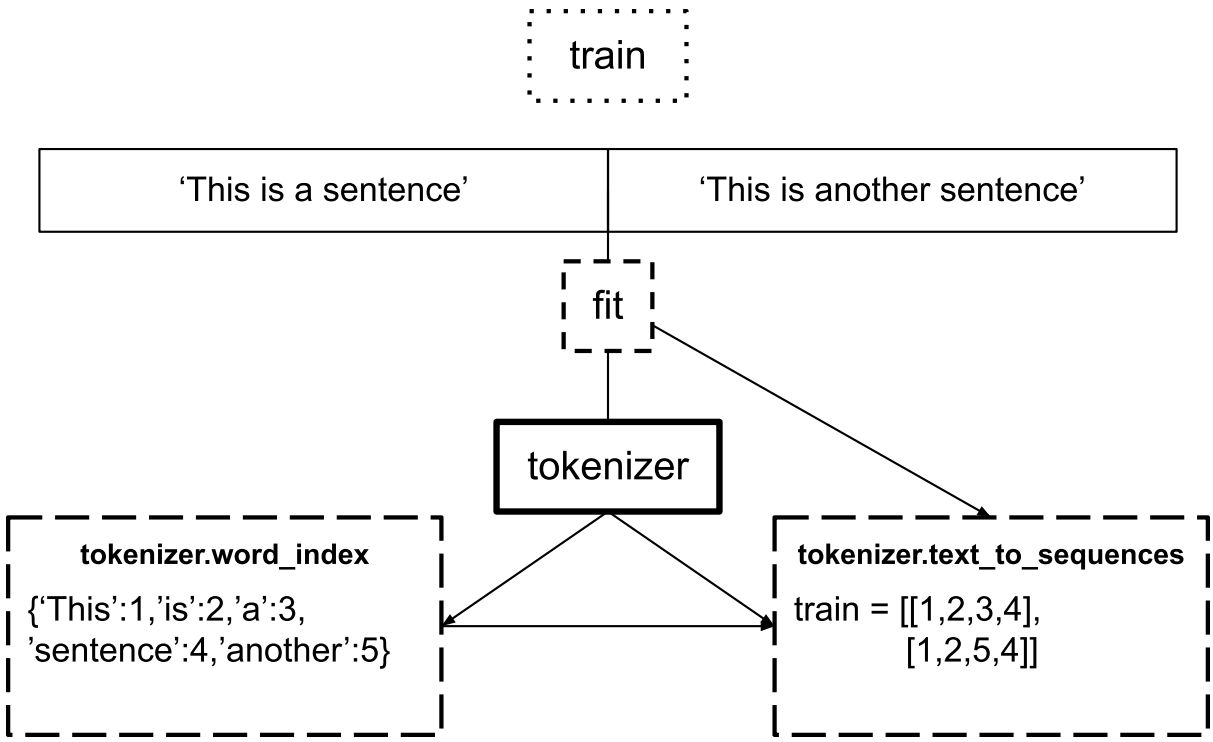

Figure 1-1: Upon receiving a corpus of documents, a tokenizer assigns a unique index to each word, thereby enabling the translation of any document into a sequential arrangement of numbers.

2.1.2. Named Entity Recognition

Named Entity Recognition is a core element of this thesis. It's a fundamental task in natural language processing that plays a pivotal role in identifying specific categories of entities within textual data. Tunstall et al. describe NER in their book *'Natural Language Processing with Transformers: Building Language Applications with Hugging Face'* as the process of extracting real-world objects like products, places, and people from text. Essentially, it involves analyzing text at various levels (sentences, paragraphs, or entire documents) to pinpoint and classify entities based on their respective categories.

Organizations employing NER for extracting insights from unstructured data utilize a variety of methodologies. In their overview of NER, IBM categorizes these methodologies into three main approaches: rule-based, machine learning, and hybrid methods.

Rule-based approaches entail the formulation of grammatical rules specific to the language being analyzed. These rules guide the identification of entities within the text based on their structural and grammatical characteristics. While effective, these approaches can be labor-intensive and may struggle to generalize well to unseen data.

Machine learning approaches involve training AI-driven models on labeled datasets using sophisticated algorithms such as conditional random fields and maximum entropy. These techniques span from traditional machine learning methods like decision trees and support vector machines to more advanced deep learning techniques such as recurrent neural networks (RNNs) and transformers. While these methods tend to perform better on unseen data, they necessitate a substantial volume of labeled training data and can be computationally intensive.

Hybrid approaches combine the strengths of both rule-based and machine learning methodologies. They employ rule-based systems to swiftly identify easily recognizable entities and machine learning systems to detect more intricate entities. This hybridization optimally balances efficiency and accuracy in entity recognition tasks.

Over the course of NER's development, notable methodological advancements have emerged, particularly in the realm of deep learning techniques. Some of the latest advancements encompass:

Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM): RNNs are specialized neural networks adept at sequence prediction tasks. LSTMs, a variant of RNNs, excel in capturing temporal patterns and retaining information across extended sequences. This capability proves invaluable for contextual understanding and entity identification (Salehinejad et al, 2017).

Transformers and BERT: Transformer architectures, notably exemplified by BERT (Bidirectional Encoder Representations from Transformers), have revolutionized NER methodologies. Through a self-attention mechanism, BERT effectively weighs the

relevance of individual words, accounting comprehensively for contextual cues by examining preceding and subsequent words (Devlin et al., 2018).

These advancements underscore the evolution of NER techniques, showcasing the efficacy of deep learning-based approaches in enhancing the extraction of book titles from user queries.

2.1.2.1. The NER process

IBM's overview of NER provides a detailed explanation of the NER process which comprises a series of steps, each pivotal in extracting meaningful insights from textual data.

Data Acquisition: The foundation of NER lies in assembling an annotated dataset, wherein text passages are labeled to denote the presence and classification of named entities. This dataset acquisition can entail manual annotation or automated techniques.

Data Preprocessing: This phase encompasses text cleansing and formatting procedures, encompassing tasks such as character removal, text normalization, and segmentation into cohesive units like sentences or tokens.

Feature Engineering: At the heart of NER lies the extraction of features from the preprocessed text. These features span several elements, including part-of-speech tagging, word embeddings, and contextual insights. The selection of features is tailored to the specific NER model under consideration.

Model Training: The NER model undergoes training, leveraging the annotated dataset and extracted features. Through this process, the model discerns patterns and interrelations within the text, thereby learning to accurately identify and categorize named entities.

Model Evaluation: The efficacy of the trained NER model is scrutinized through comprehensive evaluation measures. Metrics such as precision, recall, and F1 score serve to assess the model's aptitude to label and categorize named entities.

Model Refinement: Building upon evaluation insights, iterative refinement endeavors ensue to enhance the model's performance. This iterative process may involve hyperparameter tuning, dataset modifications, and integration of advanced techniques such as ensembling or domain adaptation.

Inference: We now have a great model! It can be used to analyze new text we haven't seen before. The model will clean up the text, find important features, and predict the named entities.

2.1.2.2. Challenges in NER Adoption

Despite significant advancements and widespread adoption, NER faces several notable challenges worth considering:

1. Language Specificity: Dr. W.J.B. Mattingly, in his book "Introduction to Named Entity Recognition," highlights a key challenge in NLP: handling documents written in multiple languages. While NER performs well for languages like English with abundant labeled data, its accuracy suffers for others. This is where advancements like BERT and transformer-based models come in potentially being key to overcoming this language barrier.

2. Domain-specific Challenges: Beyond language barriers, another challenge noted in IBM's NER overview is domain specificity. General NER models might misidentify entities specific to certain fields, like technical terms or, in our case, book titles.

Despite these challenges, ongoing advancements in NER technology are steadily enhancing its accuracy and applicability, promising to narrow existing technology gaps and broaden its impact across various domains.

Let us take a look at an example of an NER annotation process:

"Harry Potter, a student at *Hogwarts School of Witchcraft and Wizardry*, lives in the magical world of *Hogwarts*, where he learns spells and encounters mythical creatures like *dragons*."

NER Annotations:

Person: "Harry Potter"

Organization: "Hogwarts School of Witchcraft and Wizardry", "Hogwarts"

Location: "Hogwarts"

Miscellaneous: "dragons"

These annotations highlight the named entities present in the sentence, including names of individuals (e.g., "Harry Potter"), organizations (e.g., "Hogwarts School of Witchcraft and Wizardry"), and locations (e.g., "Hogwarts"). Additionally, the term "dragons" is annotated as a miscellaneous entity, indicating its significance within the context.

2.1.3. Information Retrieval

Information retrieval stands as one of the most prevalent and widely utilized applications in modern computing. A quintessential example of IR in action is Google Search, where, upon receiving a user's query, the retrieval algorithm endeavors to fetch information pertinent to that query.

Hardeniya et al. describe information retrieval as the process of finding the most relevant information needed by the user. There are different ways to express the user's information needs to the system, but the main goal is always the same: to retrieve the most relevant information.

They explain that a typical IR system generates an indexing mechanism called an inverted index. This is similar to a book's index, listing words found throughout. An IR system's inverted index posting list typically looks like this:

<Term, DocFreq, [DocId$_1$, DocId$_2$]>

{"Reading", 2 ---> [1, 2]}

{"is", 2 ---> [1,2]}

{"cool", 1 ---> [2]}

Here, if a word appears in both document 1 and document 2, the posting list will contain a list of documents referencing terms. With this data structure in place, various retrieval models can be introduced, each tailored to different types of data.

Vector space model:

Hardeniya et al. (2016, Natural Language Processing with Python and NLTK) effectively explain the concept of Vector Space Model. VSM represents documents and vocabulary terms as vectors in a high-dimensional space. Each document is essentially a unique vector in this space. While various methods exist for representing these document vectors, TF-IDF (Term Frequency-Inverse Document Frequency) is a popular and efficient approach. TF-IDF considers both the frequency of a term within a document and its importance across the entire document collection. This helps us distinguish between common words and those specific to a document's content.

Given a term and a corpus, we can calculate the term frequency (TF) and inverse document frequency (IDF) using the following formula:

$$tf(t,d) = 0.5 + \frac{0.5 \times f(t,d)}{max\{f(w,d) : w \in d\}}$$

TF, or Term Frequency, simply denotes the frequency of a term within a document. Conversely, IDF, or Inverse Document Frequency, represents the reciprocal of document frequency, indicating the number of documents in the corpus where the term appears.

$$idf(t,D) = log\frac{N}{|\{d \in D : t \in d\}|}$$

Numerous normalization variants exist, yet by integrating both Term Frequency (TF) and Inverse Document Frequency (IDF), we can devise a more robust scoring mechanism for assessing the significance of each term within a document. To compute a TF-IDF score, we simply multiply these two scores together.

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

In TF-IDF, we evaluate a term based on its presence within the current document and its distribution across the corpus. This approach identifies terms that are uncommon across corpora yet have a high frequency wherever they occur, making them discriminative for document retrieval. This scoring can represent documents as vectors. Once all documents are vectorized, the Vector Space Model can be constructed.

2.1.4. Question Answering

As data scientists and researchers, we often find ourselves wading through oceans of documents to find the information we seek. Search engines like Google continually enhance the efficiency of this process by highlighting specific answers to our questions whenever possible. For example, when querying 'When did Richard Feynman win his first Nobel Prize?' on Google, the correct answer—'October 21, 1965'—is immediately provided, as illustrated in the Figure below.

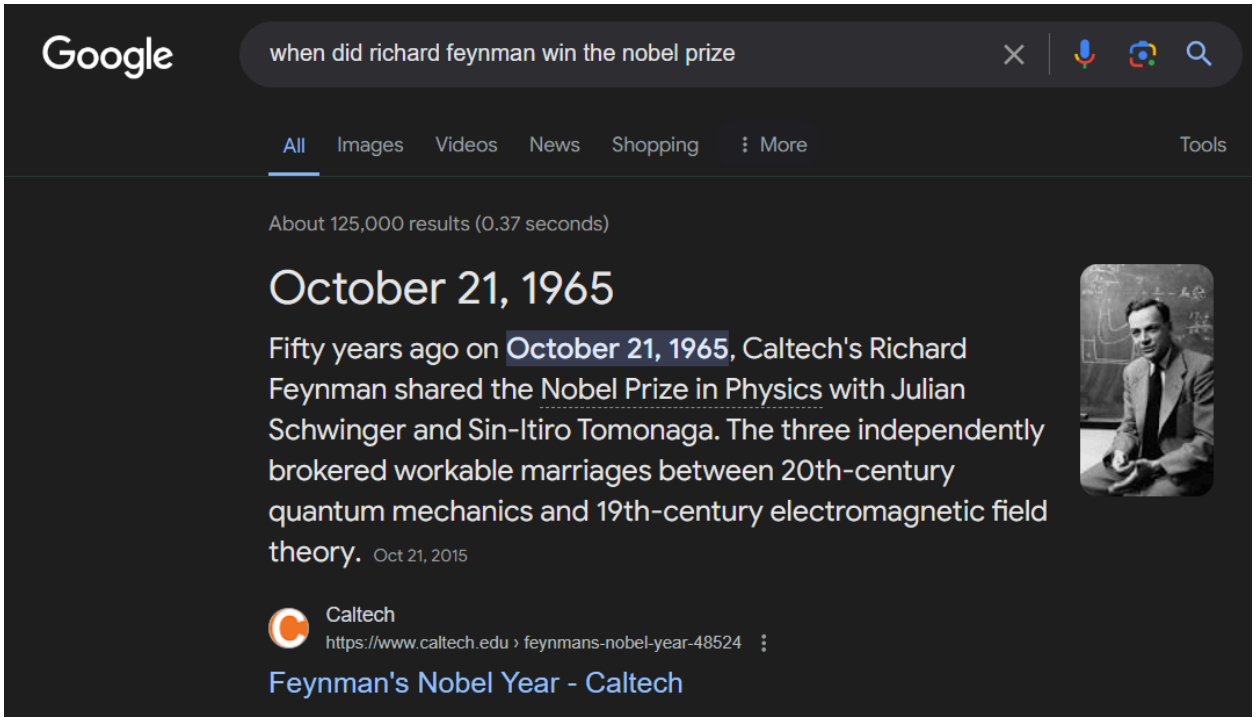

Figure 1-3: A Google search query and corresponding answer snippet.

In this example, Google first retrieved around 125,000 documents that were relevant to the query, and then performed an additional processing step to extract the answer snippet with the corresponding passage and web page. It's not hard to see why these answer snippets are useful. For example, if we search for a trickier question like "Which guitar tuning is the best?" Google doesn't provide an answer, and instead we have to click on one of the web pages returned by the search engine to find it ourselves.

The general approach behind this technology is called question answering (QA). Tunstall et al. in their book Natural Language Processing with Transformers provide a deep dive into extractive QA which is the most common type of Question Answering. They explain that extractive QA involves questions whose answers can be identified as a span of text in a document, where the document might be a web page, legal contract, or news article. The two-stage process of first retrieving relevant documents and then extracting answers from them is also the basis for many modern QA systems, including semantic search engines, intelligent assistants, and automated information extractors.

Existing QA systems employ various methodologies, including:

2.1.4.1. Information Retrieval-based Systems

Information Retrieval-based Systems have a primary objective: to efficiently locate and retrieve relevant information from a large collection of text data in response to a user's query. This query can take various forms, including keywords, phrases, or even natural language questions. Operating on a corpus of text, IR functions within a vast collection of documents or passages containing sought-after information. This corpus encompasses diverse text data types, such as articles, reports, web pages, or academic papers (Hambarde & Proença, 2023).

The process of information retrieval involves finding the documents most relevant to a query, a challenge faced by every search and recommendation system. Document retrieval systems primarily execute two processes: indexing and matching.

These systems may also utilize similarity measures to assess the resemblance between the user's query and the content of the documents in the corpus. These measures consider factors such as semantic relevance, context, and document structure to identify relevant documents.

Indexing techniques are essential for efficient retrieval in IR. By creating an index or catalog of terms, keywords, or features present in the documents along with their corresponding locations, indexing enables rapid lookup and retrieval of relevant documents during query processing.

Once potential documents or passages are identified, IR may employ ranking algorithms to prioritize the results based on their relevance to the user's query. Documents closely matching the query terms or exhibiting higher similarity scores are typically ranked higher in the list of results.

2.1.4.2. Neural Network-based Systems

These systems employ deep learning architectures, such as recurrent neural networks (RNNs) or transformer models, to learn the mapping between questions and answers from large datasets. Example: OpenAI's GPT (Generative Pre-trained Transformer) models, such as GPT-3.

2.1.4.3. Domain-specific QA Systems

Designed to answer questions within specific domains, such as medical, legal, or technical. Example: IBM Watson for Oncology, which assists oncologists in treatment decisions.

2.1.4.4. Hybrid QA Systems

Combining multiple approaches for improved performance and robustness. Example: IBM Watson, which incorporates various techniques including NLP, machine learning, and information retrieval.

2.2. Transformers

In 2017, Google researchers introduced a new way to understand and process language called the Transformer. This approach was a big improvement over older methods like recurrent neural networks (RNNs), especially for translating languages. At the same time, another method called ULMFiT showed that by using a lot of different texts, computers could get really good at understanding and classifying language, even with only a little bit of labeled data.

The success of Transformer and ULMFiT led to the creation of today's most well-known transformers: GPT and BERT. By blending the Transformer architecture with unsupervised learning, these models eliminated the requirement to build specific architectures for each task from the ground up. This breakthrough allowed them to surpass nearly every standard in NLP by a considerable margin. Since the debut of GPT and BERT, a multitude of transformer models have surfaced, showcasing a diverse range of capabilities. A timeline featuring the most notable entries is depicted in Figure 2-1.

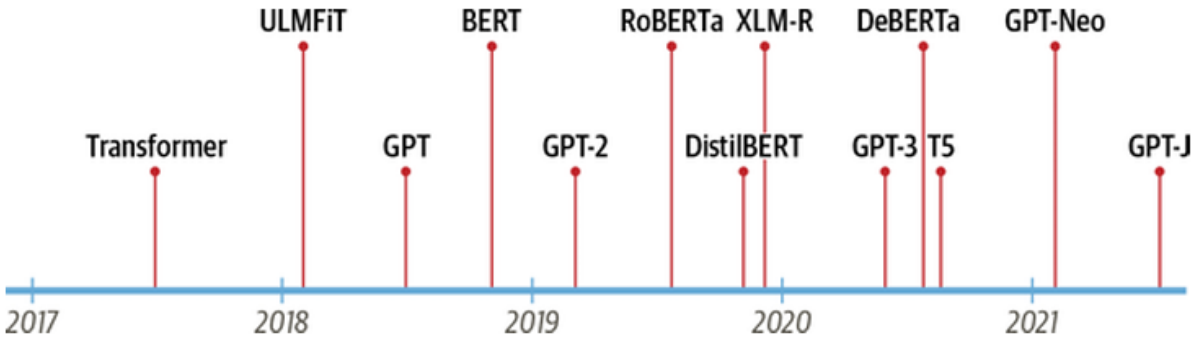

Figure 2-1: The Transformers timeline (From Natural Language Processing with Transformers by Leandro von Werra, Lewis Tunstall, and Thomas Wolf).

To understand what transformers bring to the table, Tunstall et al. provide an introduction to the core concepts underlying transformers, starting with the encoder-decoder framework.

2.2.1. The Encoder-Decoder Framework

Before transformers, recurrent architectures such as LSTMs were the state of the art in NLP. Tunstall et al. (2022) state that these architectures contain a feedback loop in the network connections that allows information to propagate from one step to another, making them ideal for modeling sequential data like text.

As illustrated on the left side of Figure 2-2, an RNN receives some input (a word or character), feeds it through the network, and outputs a vector called the hidden state. The model sends information back to itself via the feedback loop, which it can then utilize in the subsequent step.

This becomes clearer when we "unroll" the loop, as demonstrated on the right side of Figure 2-2: the RNN transfers information about its state at each step to the following operation in the sequence. This enables an RNN to maintain a record of information from previous steps and employ it for its output predictions.

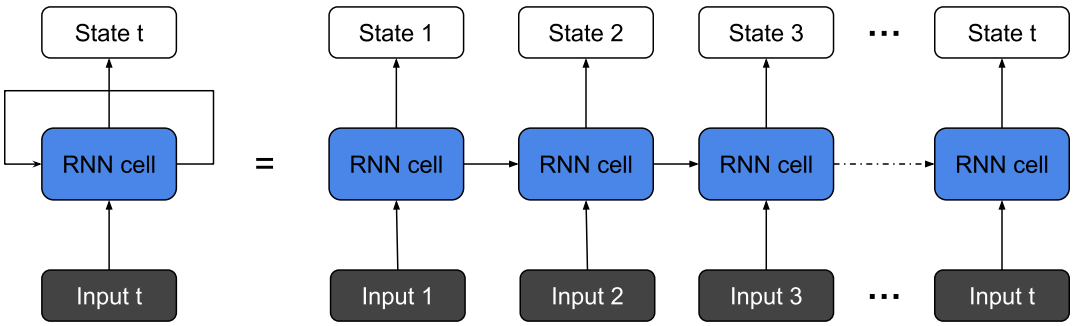

Figure 2-2: Unrolling an RNN in time.

These architectures are still extensively utilized for NLP tasks, speech processing, and time series analysis.

Typically, the encoder and decoder components can encompass any neural network architecture capable of sequence modeling. This concept is exemplified using a pair of RNNs in Figure 2-3, where the English sentence "Transformers are great!" is encoded into a hidden state vector, which is then decoded to yield the German translation

"Transformer sind grossartig!" Input words are sequentially processed through the encoder, while output words are generated sequentially, top to bottom, by the decoder.

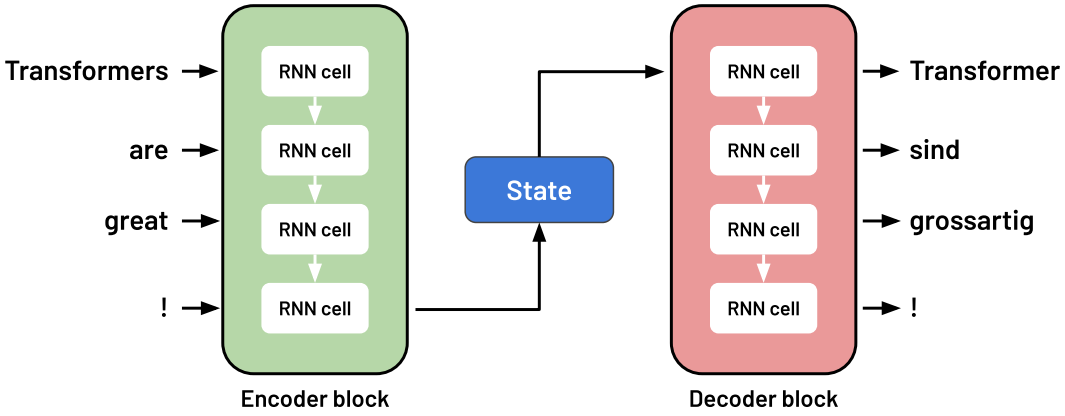

Figure 2-3: Simplified encoder-decoder architecture with a pair of RNNs (Adapted from Natural Language Processing with Transformers by Leandro von Werra, Lewis Tunstall, and Thomas Wolf).

A drawback of this architecture is that it creates an information bottleneck at the encoder's final hidden state. This single state must encapsulate the meaning of the entire input sequence, as it is the only information the decoder can access during output generation. This issue is especially problematic with lengthy sequences, as information from the beginning of the sequence may be lost when compressing everything into a single, fixed representation.

Tunstall et al. propose an attention mechanism that enables the decoder to access all of the encoder's hidden states. They specify that this mechanism is a cornerstone of many modern neural network architectures. Understanding the development of attention within RNNs lays the groundwork for comprehending one of the key elements of the Transformer architecture.

2.2.2. Attention Mechanisms

The main idea behind attention mechanisms is to give the decoder access to multiple hidden states generated by the encoder at each step of the input sequence.

However, utilizing all states simultaneously could overwhelm the decoder, so a mechanism is needed to prioritize them. Attention allows the decoder to assign different levels of importance, or "attention," to each encoder state during each decoding step. This process is illustrated in Figure 2-4, which shows how attention helps in predicting the third token in the output sequence.

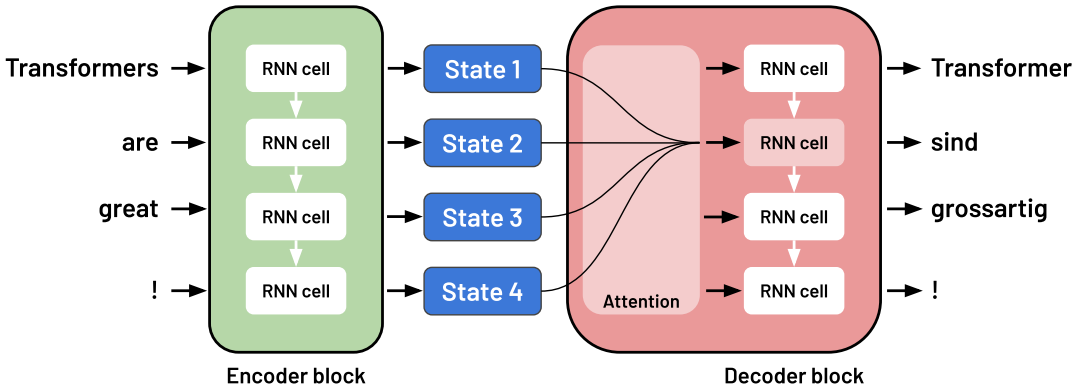

Figure 2-4: Attention Mechanism in Predicting the Third Token (Adapted from Natural Language Processing with Transformers by Leandro von Werra, Lewis Tunstall, and Thomas Wolf).

Attention-based models focus on determining the most relevant input tokens at each timestep, facilitating the learning of complex alignments between words in generated translations and those in the source sentence. For instance, Figure 2-5 illustrates attention weights in an English-to-French translation model, with each pixel representing a weight. This visualization demonstrates the decoder's ability to accurately align words like "zone" and "Area," despite their differing order in the two languages.

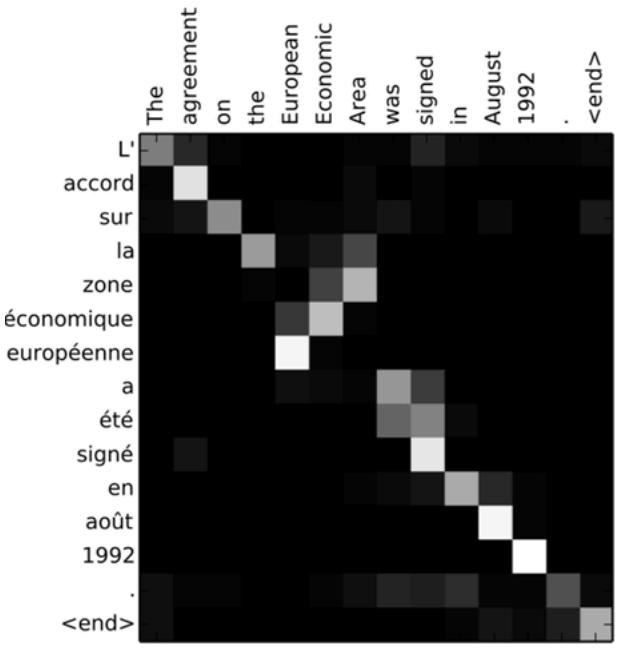

Figure 2-5: RNN encoder-decoder alignment of words in English and the generated translation in French (credits. Dzmitry Bahdanau).

While attention mechanisms significantly improved translation quality, a major drawback persisted with recurrent models for the encoder and decoder: their computations are inherently sequential and cannot be parallelized across the input sequence. Tunstall et al. noted that introducing the transformer model brought a new paradigm by completely abandoning recurrence and relying solely on a specialized form of attention known as self-attention.

2.2.3. Transfer Learning in NLP

In the years 2017 and 2018, a breakthrough occurred when research groups introduced new approaches that made transfer learning work for NLP. This advancement began with a pivotal insight from OpenAI researchers, who achieved notable performance in sentiment analysis through the utilization of features derived from unsupervised pretraining. Subsequently, the ULMFiT (Universal Language Model

Fine-tuning) method emerged, presenting a versatile framework for adapting pre-trained Long Short-Term Memory (LSTM) models to diverse tasks.

ULMFiT comprises three principal stages, as depicted in Figure 2-7.

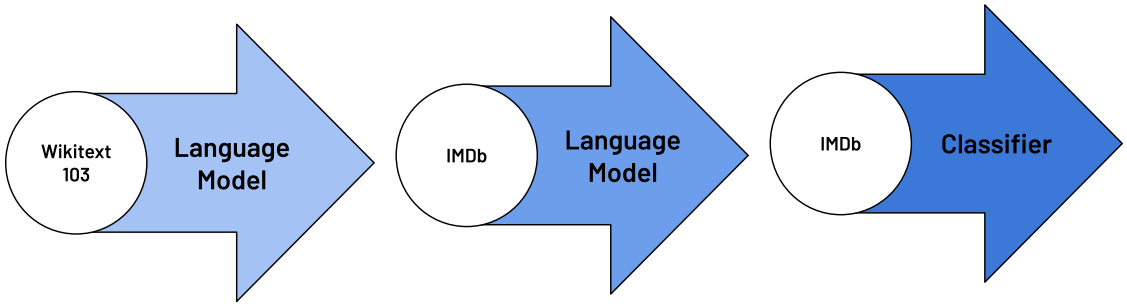

Figure 2-7: ULMFiT stages.

Tunstall et al. describe the stages as follows:

Pretraining: Initially, the model undergoes pretraining wherein it learns to predict the subsequent word based on preceding words, a task known as language modeling. Remarkably, this process requires no labeled data and capitalizes on the vast amount of available text, such as that found in Wikipedia.

Domain Adaptation: Here, the focus shifts to adapting the model to a specific domain corpus (e.g., from Wikipedia to IMDb movie reviews). This phase still involves language modeling but with the objective of predicting the next word within the target corpus.

Fine-tuning: The final step involves fine-tuning the language model with a classification layer tailored to the target task (e.g., sentiment classification of movie reviews). This fine-tuning process further refines the model's performance for the specific task at hand.

ULMFiT played a pivotal role in advancing pretraining and transfer learning within NLP, filling a critical gap and facilitating the widespread adoption of transformer models. In 2018, the emergence of two influential transformer architectures marked a significant milestone in NLP:

1. GPT: This transformer model exclusively employs the decoder component of the architecture and adopts a language modeling approach similar to ULMFiT. GPT's

pretraining was conducted on the BookCorpus, a dataset comprising 7,000 unpublished books spanning various genres.

2. BERT: BERT utilizes the encoder segment of the transformer architecture and employs a specialized form of language modeling known as masked language modeling. This approach involves predicting masked words within a text, enhancing contextual understanding. BERT's pretraining utilized both the BookCorpus and English Wikipedia (Devlin et al., 2018).

The release of Transformers, as noted by Tunstall et al., ushered in a unified API encompassing over 50 architectures. This library acted as a catalyst for the surge in transformer-focused research and quickly spread through the NLP community, streamlining the integration of these models into various real-world applications.

2.2.4. The Hugging Face Ecosystem

As highlighted by Tunstall et al., applying new machine learning architectures to NLP tasks can be cumbersome. Traditionally, researchers publish code alongside their work, but adapting this code can be time-consuming. HuggingFace Transformers offer a standardized interface for various transformer models, along with tools to adapt them to new tasks. The library supports multiple frameworks (PyTorch, TensorFlow, etc) and simplifies fine-tuning for tasks like text classification and question answering. This significantly reduces the time and effort needed for training and testing different models. The Hugging Face ecosystem comprises primarily two components: a suite of libraries and the Hub, illustrated in Figure 2-8. The libraries offer the necessary code, whereas the Hub supplies pre-trained model weights, datasets, evaluation metric scripts, and additional resources. In this section, we will provide a concise overview of these different components.

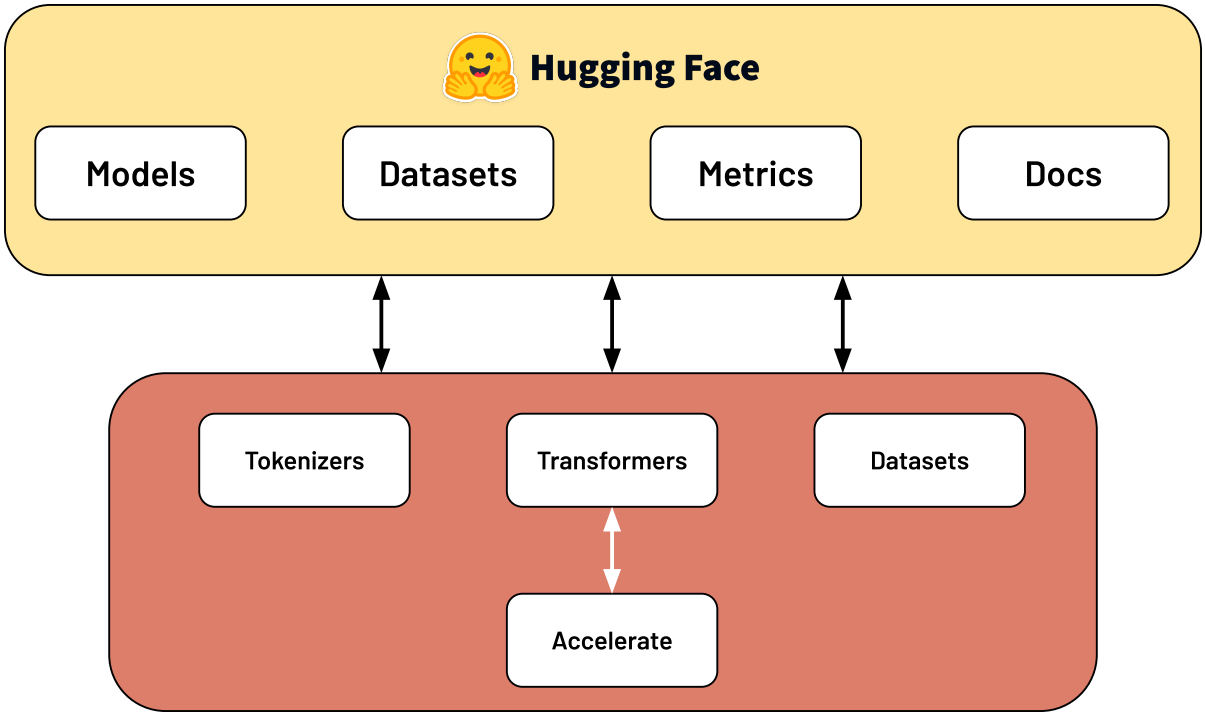

Figure 2-8: The Hugging Face Ecosystem (From Natural Language Processing with Transformers by Leandro von Werra, Lewis Tunstall, and Thomas Wolf).

The Hugging Face hub hosts over 600,000 pre-trained models. The integration of pipelines makes loading a wide range of models and experimenting with them very simple, allowing us to focus on the domain-specific parts of the project.

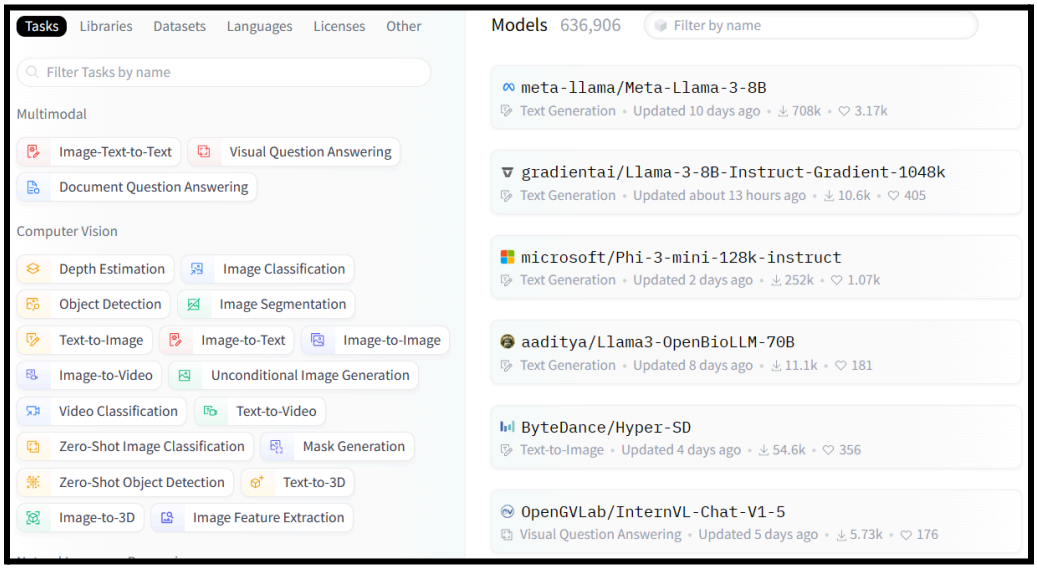

Figure 2-9: The Models section of the Hugging Face hub.

2.2.4.1. Hugging Face Tokenizers

Hugging Face Tokenizers library offers a plethora of tokenization strategies and boasts remarkable speed in text tokenization, attributed to its Rust backend. Additionally, it handles all preprocessing and postprocessing tasks, including input normalization and output transformation to the desired format. Similar to loading pre-trained model weights with Transformers, Tokenizers allow us to effortlessly load a tokenizer.

2.2.4.2. Hugging Face Datasets

Hugging Face datasets simplifies the process of loading, processing, and storing a dataset by providing a standard interface for thousands of datasets that can be found on the Hub. It includes smart caching to avoid redundant preprocessing, while also overcoming RAM limitations through memory mapping. The library seamlessly integrates with popular frameworks like Pandas and NumPy, ensuring compatibility with familiar data manipulation tools.

2.2.4.3. Hugging Face Accelerate

Hugging Face Accelerate adds a layer of abstraction to training loops that takes care of all the custom logic necessary for the training infrastructure. This accelerates the workflow by simplifying the change of infrastructure when necessary.

2.2.5. Limitations of Transformers

Tunstall et al. explain several challenges associated with using Transformers. One challenge is language: most NLP research focuses on English, making it difficult to find pre-trained models for less common languages. Another issue is data availability; despite the use of transfer learning to reduce data requirements, models still need significantly more labeled data than humans to perform the same tasks. Additionally, while self-attention is effective for short paragraphs, it becomes costly when working with longer documents. The interpretability of Transformers is also a concern, as these

models, like other deep learning models, lack transparency, making it difficult to understand their predictions. Lastly, bias is a significant challenge, as transformer models pre-trained on internet text inherit biases, complicating efforts to ensure fairness and inclusivity in their outputs.

2.3. Book Recommendation Systems

Book recommendation systems aim to provide tailored suggestions to users, enhancing their experience and facilitating the discovery of new content.
In their paper, *"Recommendation System Development Based on Intelligent Search, NLP, and Machine Learning Methods,"* Balush et al. categorize recommender systems into three main types:

2.3.1. Collaborative Filtering

This approach recommends items based on the preferences of similar users. Collaborative filtering methods include:

- User-based: Recommends books based on the preferences of users with similar tastes. Example: Amazon's "Customers who bought this item also bought" feature.
- Item-based: Recommends books similar to those the user has previously interacted with. Example: Netflix's recommendation system for movies and TV shows.

2.3.2. Content-based Filtering

This approach recommends items by analyzing the similarity between item features and user preferences. Content-based methods examine attributes such as genre, author, and plot summary. Example: Goodreads' recommendation system.

2.3.3. Hybrid Methods

These methods combine collaborative filtering and content-based filtering techniques to provide more accurate and diverse recommendations by leveraging both user preferences and item attributes. Example: Netflix's hybrid recommendation system, which integrates collaborative filtering with content analysis of movies and user behaviors.

3. Methodology

In the Literature review section, we covered several techniques used in NLP as well as different approaches that can be implemented to build question-answering and recommendation systems.

The methodology section of the thesis navigates through the development of an AI-powered assistant tailored for readers, encompassing an array of techniques in Natural Language Processing and methodologies for constructing question-answering and recommendation systems. This segment explores the entire project lifecycle, from data collection to the intricate implementation of named entity recognition, information retrieval, and question-answering systems, and culminating in the integration of a recommendation mechanism. The analysis traverses through the strategic maneuvers employed, the technical methodologies deployed, the encountered challenges, and potential pathways for enhancements.

3.1. Named Entity Recognition

To develop a robust AI system capable of effectively addressing user queries, it's paramount to train it to accurately recognize book titles. This task requires attention to two critical factors: precision and dynamism.

As discussed previously, Named entity recognition is an NLP technology that empowers systems to extract names of individuals, locations, organizations, quantities, monetary values, and percentages, among others. Unfortunately, “book_title” is not one of those entities that NER is trained to detect. This calls for the need to train a custom NER model that is specialized in book title detection.

To initiate the model-building process, we begin by compiling a diverse array of book titles that vary in structure and format. These titles are chosen to represent the broad spectrum of formats found in book titles. Following this, a dataset containing frequently asked questions about books is created and stored. Each question in this dataset is

designed to abstractly reference a specific book title, such as "Who is the author of {title}?", ensuring the questions are adaptable to various titles.

Next, NLP techniques are applied to each collected book title. This step ensures that the model can identify the titles from contextual cues rather than relying solely on factors like capitalization or special formatting. The cleaned title then replaces the placeholder {title} in each question. After applying all questions to the collected book titles, an annotated dataset is generated. This dataset maps annotations to each question, indicating the start and end indices of the book title referenced in every question. This annotation process enhances the model's ability to accurately identify and respond to inquiries regarding various book titles, which will later prove crucial for the effectiveness of our AI assistant.

Let us break down the steps involved in training the custom entity recognition model.

We begin by importing the necessary libraries, including spaCy and Hugging Face's Transformers, and load a pre-trained NER model based on BERT from Hugging Face's Transformers library.

The main library used for this task is spaCy, which stands out as a leading open-source toolkit for NLP. With support for over 75 languages and 84 trained pipelines across 25 languages, spaCy offers a comprehensive suite of functionalities for diverse NLP tasks. It incorporates multi-task learning capabilities with pre-trained transformers like BERT, enabling sophisticated processing of text data. Additionally, spaCy provides pre-trained word vectors and boasts state-of-the-art speed, making it an efficient choice for NLP applications.

Furthermore, spaCy's production-ready training system facilitates the development of custom models, leveraging linguistically-motivated tokenization and components for various tasks such as named entity recognition, part-of-speech tagging, dependency parsing, and more. Its architecture is easily extensible with custom components and attributes, supporting integration with frameworks like PyTorch and TensorFlow. spaCy

also offers built-in visualizers for syntax and NER, aiding in model interpretation and debugging.

We use our annotated data to fine-tune the pre-trained spaCy model, training it to detect the BOOK_TITLE label in text. To further enhance the training process, we leverage a transformer imported from Hugging Face known as DistilBERT. The pre-trained BERT model serves as a feature extractor, enhancing the model's ability to capture contextual information relevant to book title recognition. DistilBERT, a distilled version of the BERT model, retains much of its performance while being computationally more efficient. It achieves this by reducing the number of parameters and employing various optimization techniques. DistilBERT is well-suited for scenarios where computational resources are limited, making it an appropriate choice for training our custom NER model on book titles.

Below is a demonstration of the implementation of the custom NER model:

User input:

"who wrote the lonely polygamist?"

Extracted entities from the model:

[('the lonely polygamist', 'BOOK_TITLE')]

In this example, the user is asking for the author of the book "The Lonely Polygamist". The NER model successfully identifies the book title from the query and feeds the information to the API call. Subsequently, data about the book in question is fetched from the Google Books API to be leveraged for our Question Answering models.

3.2. Data Collection

The data collection stage involved two different stages. The first one being the curation of a bestseller dataset to collect book titles that would later be used to build the

named entity recognition models in order to accurately identify and classify book titles within queries. The second and main data collection stage aims to enable the development and real-time implementation of the question-answering and information retrieval system by leveraging data from the Google Books API both for training and testing the Information Retrieval models.

The new Google Books API offers programmatic access to a myriad of operations available on the Google Books website, enabling developers to create robust applications with deeper integration. Key features include search and browse functionality to explore books matching specific queries, access to detailed book information such as metadata, availability, price, and preview links, and the management of personalized bookshelves.

To collect information about a specific book, we send a request to the Google Books API including the book title provided by the user:

https://www.googleapis.com/books/v1/volumes?q=intitle:{title}&langRestrict=en&orderBy=relevance&printType=books

We use the *"intitle"* parameter to make sure the API only returns results where the title provided by the user is found in the book title returned by the API.

We also restrict the API to only return results in the English language by assigning the value "en" to the *"langRestrict"* parameter.

The *"orderBy"* parameter is used to order the results by relevance so we can guarantee receiving the most relevant books to the user's query.

The *"printType"* parameter ensures the results contain only books as opposed to other types of prints like magazines.

We may also specify the maximum number of results to return by using the *"maxResults"* parameter.

Retrieving volume information from the Google Books API does not require authentication, so we do not have to provide an API key when sending a request.

Since the data collection process is dynamic, meaning that it depends on users' queries, we will not be storing data locally, hence the non-explicit specification of the scale of

data collection. This allows us to be scalable but most importantly it does not limit the amount of data available to the AI.

The Google Books API offers access to an extensive repository of literary works, serving as a rich source of textual data. The data retrieved from the API is formatted as a JSON containing a list of dictionaries with key-value pairs that vary for each book. Typically, the returned data includes, but is not limited to, the book's title, author, description, publisher, publication date, edition, page count, and text snippets.

3.3. Information Retrieval-based Question-Answering

We previously went over the various approaches for building question-answering systems. The data fetched from the API provides a solid ground for the implementation of an information retrieval system where answers to user queries are extracted from the fetched data corpus.

The first thing we'll need for our QA system is to find a way to identify a potential answer as a span of text in a JSON text. For example, if we have a question like "Who wrote To Kill a Mockingbird?" and the returned textual data contains "Voted America's Best-Loved Novel in PBS's The Great American Read Harper Lee's Pulitzer Prize-winning masterwork..." or simply "authors": ["Harper Lee"], then the model should output "Harper Lee". To do this we'll need to understand how to:

- Frame the supervised learning problem.
- Tokenize and encode text for QA tasks.
- Deal with long passages that exceed a model's maximum context size.

Let's start by taking a look at how to frame the problem.

The most common way to extract answers from text is by framing the problem as a span classification task, where the start and end tokens of an answer span act as the labels that a model needs to predict. This process is illustrated in the figure 3-1.

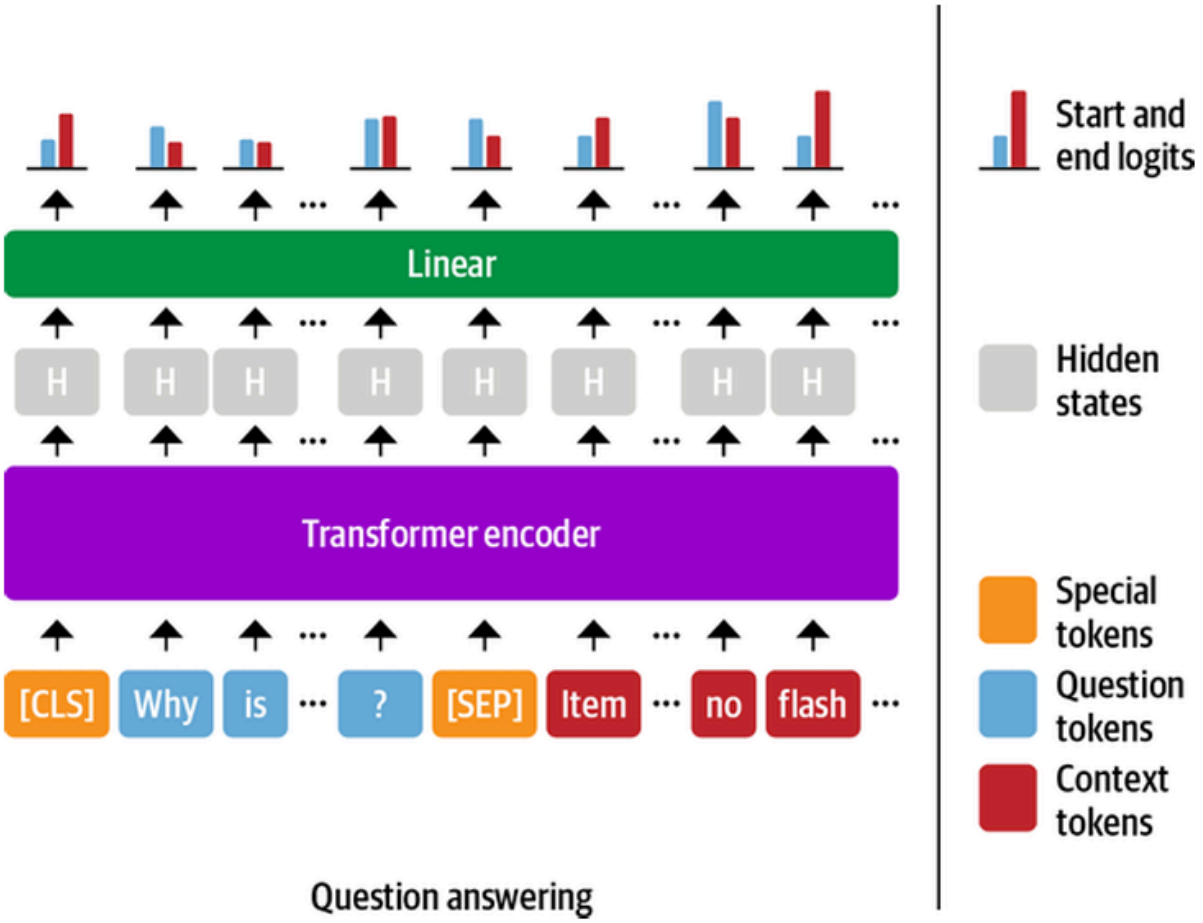

Figure 3-1: Span classification head for QA tasks (From Natural Language Processing with Transformers by Leandro von Werra, Lewis Tunstall, and Thomas Wolf).

Given the scale of this project, a good strategy is to start with a language model that has already been fine-tuned on a large-scale QA dataset like WikiAnswers, SQuAD, and GOOAQ. In general, these models have strong reading comprehension capabilities and serve as a good baseline upon which to build a more accurate system.

The choice of the QA model depends on various factors like whether the corpus is mono or multilingual and the constraints of running the model in a production environment.

Model Name	Performance Sentence Embeddings (14 Datasets)	Performance Semantic Search (6 Datasets)	Avg. Performance	Speed	Model Size
all-mpnet-base-v2	69.57	57.02	63.30	2800	420 MB
multi-qa-mpnet-base-dot-v1	66.76	57.60	62.18	2800	420 MB
all-distilroberta-v1	68.73	50.94	59.84	4000	290 MB
all-MiniLM-L12-v2	68.70	50.82	59.76	7500	120 MB
multi-qa-distilbert-cos-v1	65.98	52.83	59.41	4000	250 MB
all-MiniLM-L6-v2	68.06	49.54	58.80	14200	80 MB
multi-qa-MiniLM-L6-cos-v1	64.33	51.83	58.08	14200	80 MB
paraphrase-multilingual-mpnet-base-v2	65.83	41.68	53.75	2500	970 MB
paraphrase-albert-small-v2	64.46	40.04	52.25	5000	43 MB
paraphrase-multilingual-MiniLM-L12-v2	64.25	39.19	51.72	7500	420 MB
paraphrase-MiniLM-L3-v2	62.29	39.19	50.74	19000	61 MB
distiluse-base-multilingual-cased-v1	61.30	29.87	45.59	4000	480 MB
distiluse-base-multilingual-cased-v2	60.18	27.35	43.77	4000	480 MB

Figure 3-2: Performance of SentenceTransformers models on Sentence Embedding and Semantic Search (Source: https://www.sbert.net/docs/pretrained_models.html).

For the thesis, we use a fine-tuned SentenceTransformers model designed for semantic search. The all-* models were trained on all available training data (more than 1 billion training pairs) and are designed as general-purpose models. While the "all-mpnet-base-v2" model boasts the highest performance, we prioritize speed for our project. Therefore, we've opted for the "all-MiniLM-L6-v2" model, which offers a compelling balance between speed (being 5 times faster) and performance.

3.3.1. Extractive QA with Haystack

Haystack is a complete framework for creating robust Pipelines using Large Language Models for various search purposes. Whether it's retrieval-augmented generation (RAG), question answering, or semantic document search, Haystack utilizes cutting-edge LLMs and NLP models to offer tailored search experiences, enabling users to query in natural language.

Its modularity allows us to combine powerful technologies from OpenAI, Chroma, and other open-source projects, like Hugging Face's Transformers.

We leverage Haystack 2.0, a significant upgrade that redesigns various components, including Document Stores and Pipelines. Unlike its predecessor, Haystack 1.x, where pipelines were constructed by sequentially adding nodes, in Haystack 2.0, this process undergoes a two-step evolution. Initially, components are added to the pipeline without any predetermined order using the add_component method. Then, to establish the final graph, these components must be explicitly connected through the connect method. The Pipeline concept is a fundamental requirement and an optimal fit for building applications with LLMs, which is why Pipelines and Components are still the foundation of Haystack 2.0.

In practice, users typically ask questions about books, so we need some way of selecting relevant passages from all the book info we gather. One approach is to fetch all the responses from the Google Books API for a query and feed them to the model as one long context. But this can slow things down a lot and introduce an unacceptable latency for our users' queries. As an example, let's suppose that on average, each book query returns 50 results in Google Books and each result takes 100 milliseconds to process. If we need to process all the reviews to get an answer, this would result in an average latency of 5 seconds per query—much too long for a chatbot interaction! Let us look at how to handle this by dissecting Haystack components.

3.3.1.1. Haystack Components

Haystack offers various components, each performing different kinds of tasks. These are often powered by the latest LLMs and transformer models.

3.3.1.2. Document Store

The Document Store is like a container for documents in Haystack. While it's not actually considered a component as it doesn't have the run() method used in all Haystack components, it's more like an interface to our database where we can store information

and search through it. So, it's not a part of the Pipeline; rather, it's a tool that components within the pipeline can use and interact with.

Choosing the right document store is a crucial step because it determines where we store the information we fetch from the API. The table below provides a quick summary of different Document Stores available in Haystack.

Type	Best for
Vector libraries	Managing hardware resources effectively.
Pure vector DBs	Managing lots of high-dimensional data.
Vector-capable SQL DBs	Lower maintenance costs with focus on structured data and less on vectors.
Vector-capable NoSQL DBs	Combining vectors with structured data without the limitations of the traditional relational model.
Full-text search DBs	Superior full-text search, reliable for production.
In-memory	Fast, minimal prototypes on small datasets.

Figure 3-3: Haystack 2.0 Document stores.

After exploring different options, the in-memory document store makes the most sense to use. Given that we'll be storing information about the book queried by the user in real time, we are looking for a fast and minimalistic option that doesn't use up a lot of resources. Haystack ships with an ephemeral document store that relies on pure Python data structures stored in memory, so it doesn't fall into any of the vector database categories above. This special Document Store is ideal for creating quick prototypes with small datasets. It doesn't require any special setup, and it can be used right away without installing additional dependencies.

3.3.1.3. Embedders

The embeddings generated by Haystack embedders consist of fixed-length vectors. They encapsulate contextual information and semantic connections within the text. The main aim of these embeddings is to convert text into a format that enables the

language model to comprehend and analyze it with greater nuance and contextual awareness.

We use “SentenceTransformersTextEmbedder” for query embedding and “SentenceTransformersDocumentEmbedder” to embed a list of documents with a Sentence Transformer model.

3.3.1.4. Retrievers

Retrievers are in charge of finding relevant documents for a given query. They come in two types: sparse and dense. Sparse retrievers use word frequencies to represent documents and queries as sparse vectors. Relevance is determined by calculating the inner product of these vectors. Dense retrievers, however, use encoders like transformers to represent the query and document as contextualized embeddings (which are dense vectors). This allows them to understand query content better and improve search accuracy (Zhao et al., 2022).

Retrievers go through all the Documents in a Document Store, select the ones that match the user query, and pass it on to the next component. Various Retrievers are customized for specific Document Stores. For our AI assistant, we used The InMemoryEmbeddingRetriever, an embedding-based Retriever compatible with the InMemoryDocumentStore. It compares the query and Document embeddings and fetches the Documents most relevant to the query from the InMemoryDocumentStore based on the outcome.

3.3.1.5. Readers

Readers extract answers from the documents retrieved by the retriever. They're typically reading comprehension models, though some models can generate free-form answers. They evaluate answers by assigning them a probability score ranging from 0 to 1, indicating how closely they match the query. A score closer to 1 signifies higher confidence in the answer's relevance. Answers are then sorted based on these probability

scores, with the highest probabilities listed first. Optionally, you can specify the maximum number of answers returned by the Reader using the top_k parameter.

We can use these probability scores to establish quality standards for our system. By adjusting the confidence_score parameter of the Reader, we can set a probability threshold for answers. For instance, setting *confidence_threshold* to 0.6 ensures that only answers with probabilities greater than 0.6 are considered.

Figure 3-4 demonstrates how additional components can perform post-processing on the documents retrieved by the retriever or the answers extracted by the reader. For instance, retrieved documents might require reranking to remove noisy or irrelevant ones, which could otherwise confuse the reader. Likewise, post-processing of the reader's answers is often necessary when the correct answer spans multiple passages in a lengthy document.

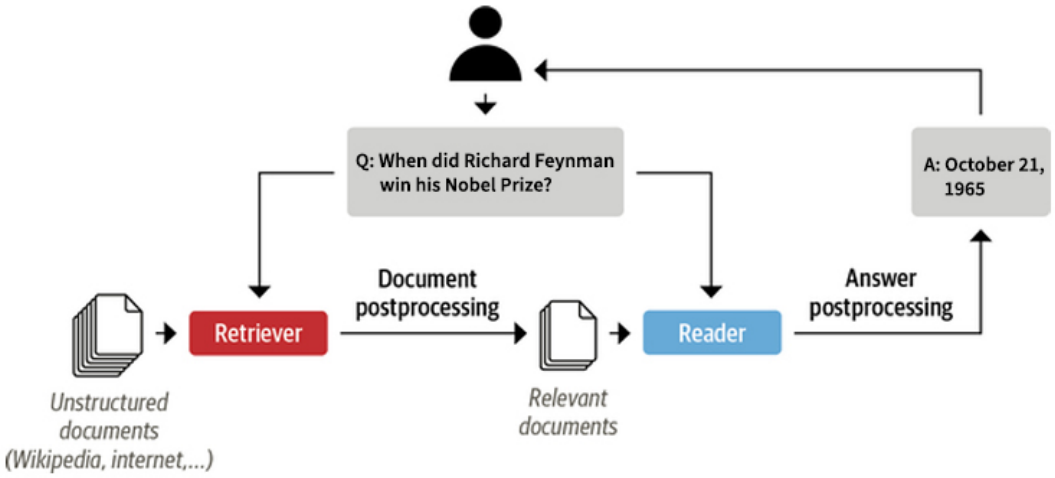

Figure 3-4: The retriever-reader architecture for modern QA systems (Adaprted from Natural Language Processing with Transformers by Leandro von Werra, Lewis Tunstall, and Thomas Wolf).

3.3.1.6. Generators

Generators produce text responses based on prompts provided to them. They are tailored for each LLM technology, such as OpenAI, Cohere, local models, and others. Generators come in two types: chat and non-chat.

Chat generators are geared towards conversational interactions, allowing for chat completion. They work with a list of messages to engage with the user.

Non-chat generators use LLMs for simpler text generation tasks, such as translation or summarization.

3.3.1.7. Pipelines

Pipelines allow us to integrate various components, Document Stores, and integrations, creating robust and customized systems. They offer great flexibility, enabling simultaneous flows, standalone components, loops, and diverse connections.
The diagram illustrated in Figure 3-5 provides a comprehensive summary of the information retrieval process.

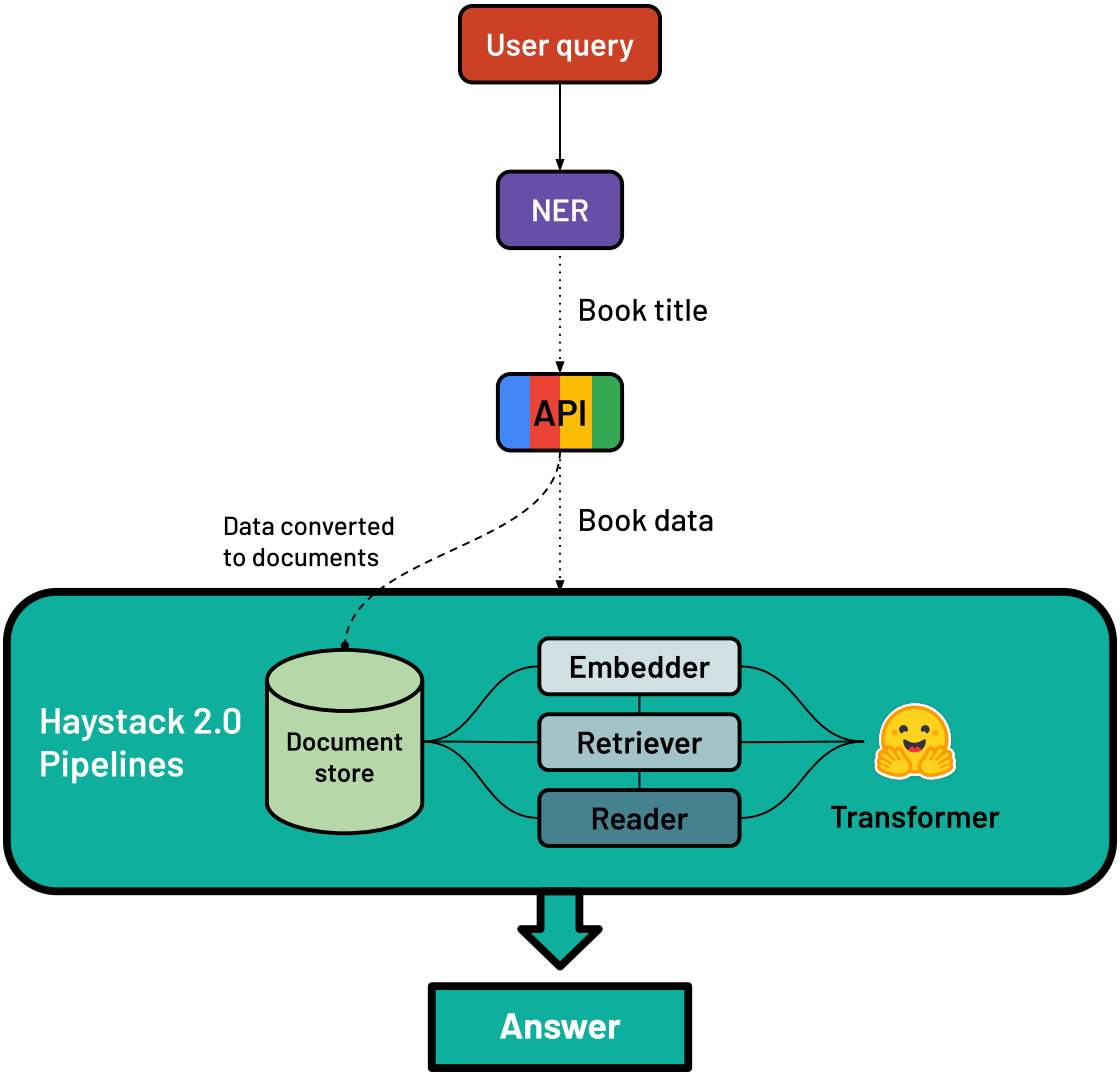

Figure 3-5: Overview of the Haystack information retrieval process.

3.4. Book Recommendation System

Integrating a recommender system into the chatbot project stems from the initial motive of creating an AI assistant that mimics a librarian. The goal is to suggest books similar to what users like, whether by recommending a list based on a single title provided by a user or by browsing their personal library. The recommender engine adopts the content-based filtering approach, utilizing the content or attributes of the item and a notion of similarity to generate similar items with respect to the given item. Two methods have been adopted for building the book recommender: one leveraging cosine similarity and the other based on a Hugging Face transformer.

3.4.1. Cosine Similarity Approach

Cosine similarity quantifies the similarity between two non-zero vectors in an inner product space by measuring the cosine of the angle between them. It equals 1 when the angle is 0° and decreases for any other angle, never exceeding 1.

$$cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

In this context, A_i represents a component of vector A, and B_i represents a component of vector B.

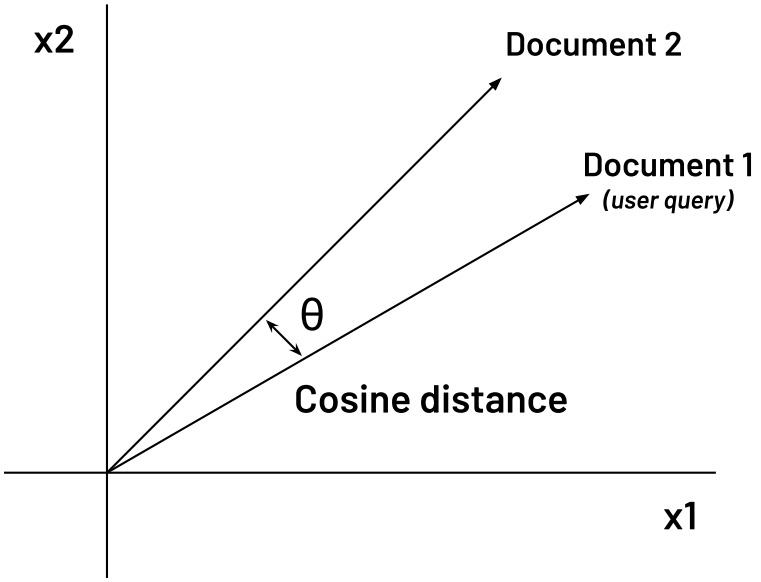

Figure 3-5: Cosine Similarity

Example: Let us assume A = [2, 1, 0, 1, 0, 1, 1, 2], B = [2, 1, 1, 1, 1, 0, 1, 2] are the two vectors and we would like to calculate the cosine similarity:

$$\sum_{i=1}^{n} A_i B_i = ((2 \times 2) + (1 \times 1) + (0 \times 1) + (1 \times 1) + (0 \times 1) + (1 \times 0) + (1 \times 1) + (2 \times 2)) = 11$$

$$\sqrt{\sum_{i=1}^{n} A_i^2} = \sqrt{(2^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2 + 1^2 + 2^2)} = 3.46$$

$$\sqrt{\sum_{i=1}^{n} B_i^2} = \sqrt{(2^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 1^2 + 2^2)} = 3.60$$

$$Cosine\ similarity\colon\ cos(\theta) = \frac{11}{3.46 * 3.60} = 0.88$$

A value of 0.88 signifies significant similarity between the two vectors, nearing the highest possible score of 1. When computing similarities between books, cosine similarity is applied to their TF-IDF vectors. These vectors are then arranged in descending order based on their cosine similarity scores, effectively organizing all other items by their proximity to the vector under comparison.

Breaking down the process:

The TF-IDF matrix, essential for analyzing textual content, is computed for all book data collected from the API. TF-IDF, a statistical measure covered in the Literature Review, evaluates the significance of words within a document concerning a broader collection (corpus) of documents. Leveraging the TfidfVectorizer tool from scikit-learn, this matrix translates textual attributes into a numerical format.

To ensure consistency and accuracy in analysis, text preprocessing is employed before generating the TF-IDF matrix. This preprocessing step encompasses various techniques such as tokenization, lowercasing, punctuation removal, stopword elimination, and lemmatization. By standardizing the textual data, the recommender system optimizes its ability to detect similarities effectively.

Once the TF-IDF matrices for both the fetched book and the collected data are prepared, cosine similarity is computed between their respective TF-IDF vectors. Cosine similarity measures the cosine of the angle between two vectors, providing a quantifiable metric for their textual resemblance.

Based on the computed cosine similarity scores, the system identifies the indices of books most akin to the fetched book. These indices serve as a reference to extract similar books from the DataFrame, allowing the system to compile a curated list of recommendations closely aligned with the user's preferences.

3.4.2. Transformer-based Approach

The second approach to calculating similarity between books involves using a Hugging Face transformer model, which maps returned contexts from the API into a high-dimensional vector space. This approach is the principal method adopted for our recommendation engine due to its superior ability to capture semantic relationships between collected book data.

We leverage the pre-trained transformer model "all-MiniLM-L6-v2," which we also employed for the Question Answering model. This model is part of the MiniLM family, designed to provide efficient and effective embeddings while maintaining a small footprint suitable for practical applications.

The book descriptions are fed into the all-MiniLM-L6-v2 model, which converts each description into a dense vector representation (embedding). These embeddings capture rich semantic information by considering the context in which words appear.

The transformer model employs a self-attention mechanism, which enables it to weigh the importance of different words in a sentence relative to one another, capturing relationships and dependencies that traditional vectorization methods (like TF-IDF) might miss.

After obtaining the embeddings for all book descriptions, we calculate the cosine similarity between the embedding of a reference book and the embeddings of other books in the dataset.

Unlike traditional vectorization methods, transformer embeddings can understand the context in which words are used, making them adept at capturing the nuances of language. For example, the model can differentiate between polysemous words (words with multiple meanings) based on the surrounding text (Ethayarajh, 2019).

The embeddings produced by the transformer model are dense and continuous, effectively encoding semantic relationships. This results in more accurate similarity measurements, as books with similar themes, genres, or topics will have closer embeddings in the vector space.

The transformer model is pre-trained on a large and diverse corpus, allowing it to generalize well across different types of text. This enables it to leverage a vast amount of linguistic knowledge when generating embeddings.

Using the transformer-based approach, we expect a notable improvement in the quality of recommendations compared to the traditional cosine similarity method, given the model's ability to capture deeper semantic connections between book descriptions.

4. Implementation

In this section, a demonstration of the beta version of the AI is given covering a practical representation of the features and technologies described in the methodology.

4.1. Demonstration of the AI assistant

We start with a quick introduction to Gradio - a tool that's been rising in popularity for its ease of use in building interactive interfaces for machine learning models. Gradio enables developers to create intuitive user interfaces that allow users to interact with AI models through web browsers, enabling seamless experimentation and showcasing of AI capabilities. ChatInterface is Gradio's high-level abstraction for creating chatbot UIs, and allows us to create a web-based demo around a chatbot model in a few lines of code.

As mentioned before, the data fetched from the API is stored in a context that's forwarded to a document store. To achieve a better user experience and faster runtimes, the AI is built to persist the context whilst the kernel is running. If the user's queries switch to a different book, the AI is able to recognize that and act accordingly. Similarly, if the user keeps asking questions about the same book the AI retrieves the answers from the same context.

The AI is also capable of persisting context to carry out a conversation about the same book without the need to repeatedly mention the book of interest.

Figure 4-1 illustrates the chat interface and demonstrates the AI's ability to dynamically adapt to context changes.

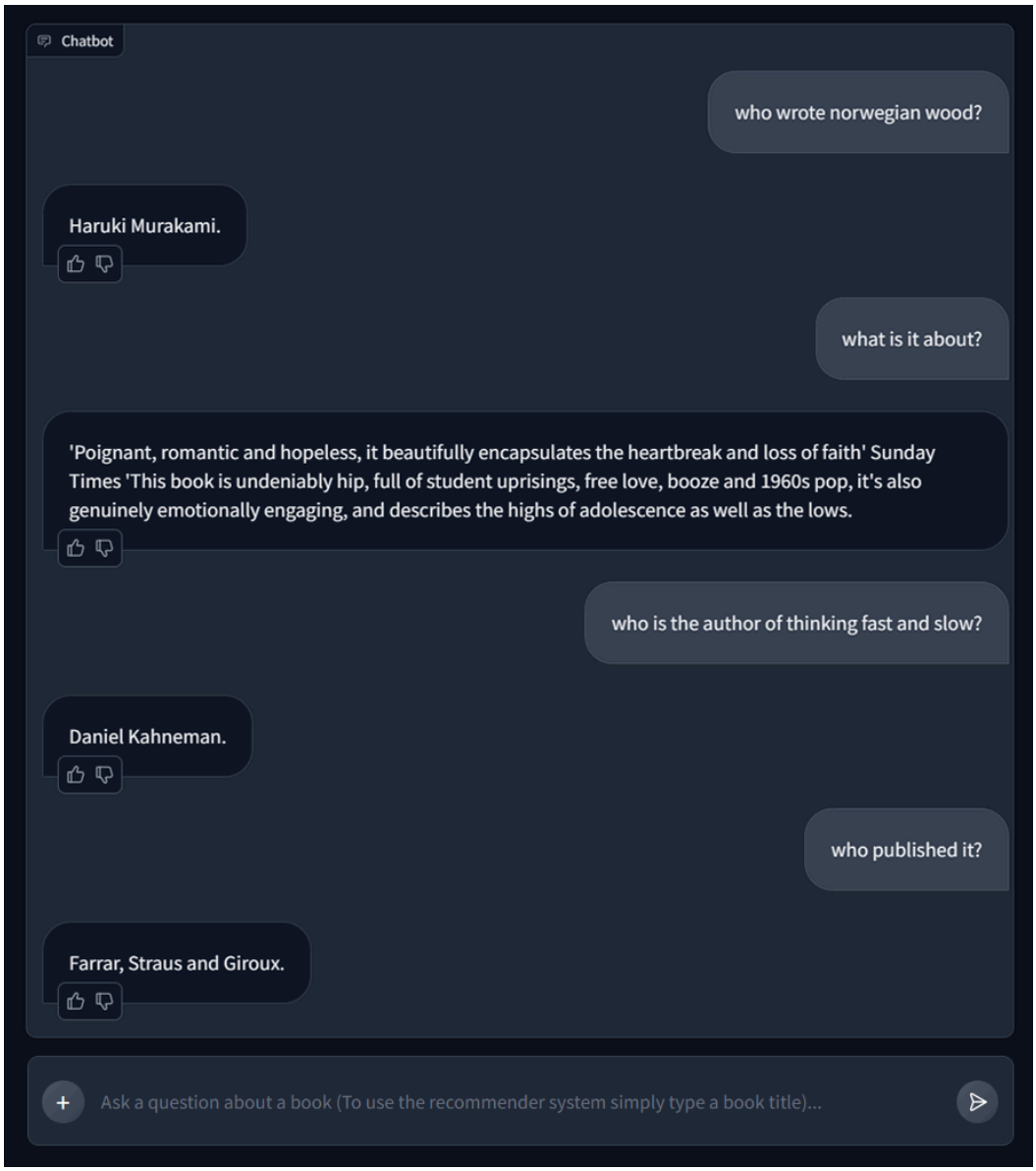

Figure 4-1: Demonstration of the chatbot.

4.1.1. Integrated Recommendation System

In order to make use of the built-in recommendation system, users simply input a book title. The recommender tool outputs a curated list of 10 similar books ordered by their cosine similarity scores. Here is an overview of how the recommendation tool works:

Input: When a user enters a book title, the tool queries the Google Books API to retrieve pertinent details. These include the book's title, author, and description, all of which are preprocessed for enhanced efficiency and model performance.

Embeddings Generation: Pre-trained sentence transformer models are employed to generate semantic embeddings for the queried book. This process encapsulates the book's essence in a mathematical representation.

Similarity Computation: Utilizing cosine similarity, the tool calculates the resemblance between the embeddings of the queried book and the existing dataset. This step determines the closeness in meaning between the queried book and others in the collection.

Ranking and Output: The books in the dataset are then ranked based on their cosine similarity to the queried book, and the top 10 most similar books are presented to the user.

The integrated recommendation engine is illustrated in Figure 4-2.

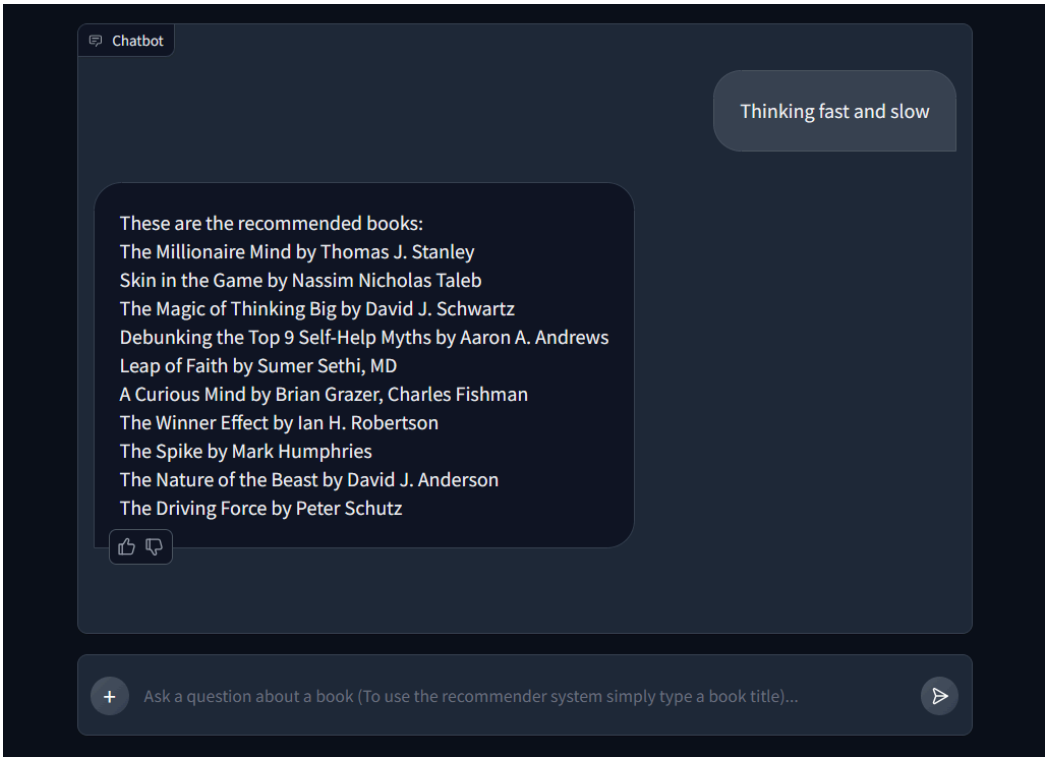

Figure 4-2. The integrated recommendation engine.

4.1.2. Model Evaluation

To evaluate both the NER and question-answering models, Haystack offers tools to assess entire pipelines or individual components like Retrievers, Readers, or Generators. In this case, we are interested in evaluating the NER model's capacity in detecting book titles from queries as well as the question-answering model's accuracy in fetching the right information.

One of Haystack's Evaluation models, the "SASEvaluator" (Semantic Answer Evaluator), is used to evaluate answers predicted by Haystack pipelines. This evaluation involves checking the semantic similarity between a predicted answer and the ground truth answer using a fine-tuned language model.

Let's go over the evaluation process of the Named Entity Recognition model. We begin by generating queries for 100 random book titles and we compare the extracted titles obtained using the custom NER to the actual book titles. Additionally, to better understand the performance of our NER model specifically built for detecting books, we run the same evaluation experiment using a baseline NER model pre-trained for extracting general entities instead of book titles. This comparison will provide a clearer perspective on how well our custom model performs relative to a general-purpose entity recognition model.

The table below presents the evaluation results for both models.

		Custom NER model trained for book detection	Baseline NER Model
Features	Pre-trained	✓	✓
	Entity recognition	✓	✓
	Fine-tuned on book labels	✓	✗
Performance	Accuracy	0.98	0.05
	F1 Score	0.96	0.03
	Semantic evaluation score	0.96	0.533

The performance metrics reveal a strong contrast between the two models. Our custom NER model achieves an impressive accuracy of 0.98 and an F1 Score of 0.96, reflecting its high precision and recall in identifying book titles. In contrast, the baseline model scores a mere 0.05 in accuracy and 0.03 in F1 Score, indicating poor performance. Additionally, the custom model's semantic evaluation score of 0.96 demonstrates a strong understanding of the context and meaning of book titles, whereas the baseline model scores only 0.533. This significant disparity underscores the effectiveness of domain-specific fine-tuning, as the custom NER model consistently outperforms the baseline model across all metrics.

The question-answering model, on the other hand, is assessed by generating questions about random books and comparing the chatbot's answers to the ground-truth values that are structurally extracted from the key-value pairs found in the API response. The QA model demonstrated a semantic evaluation score of 81.56% during testing.

Evidently, deployment of the AI will provide room for more extensive testing which eventually contributes to further performance improvements.

4.1.3. Evaluation of the Recommendation Engines.

To evaluate the recommendation engine, we compare the Transformer-based approach with the traditional Cosine Similarity approach.

The system calculates the similarity between each reference book's description and all descriptions available in our dataset. For each reference book, the 10 books with the highest similarity scores are added to a dataframe. This process is repeated for all 10 reference books, resulting in a final dataframe containing 100 recommendations.

Once the recommendations are collected, we evaluate and compare both approaches based on two main metrics: the similarity score and a generated relevance score that measures how relevant the recommended books are to the reference books.

To obtain the relevance score, we implement a feature extraction transformer model called *"distilbert-base-uncased"* to extract the 10 most important keywords for all the books in our evaluation dataframe. We then examine the author and genre for each recommended book and compare them against their corresponding reference book.

A recommendation is deemed "relevant" if one of the following conditions are met:

- The similarity score of the recommendation is higher than or equal to 0.5 (50%).
- There is a presence of common keywords between the reference book and the recommendation.
- The author of the recommended book is also the author of the reference book.
- The recommendation and the reference book share the same genre.

The relevance score is then obtained by computing the ratio of relevant recommendations in the dataframe.

A snapshot is provided in the following table depicting the results of running both recommendation engines on an example reference book ("1984").

Reference Book	Recommendation		Similarity Score		Relevance	
	Traditional Cosine Similarity	**Transformer**	**Traditional Cosine Similarity**	**Transformer**	**Traditional Cosine Similarity**	**Transformer**
"1984"	*On Nineteen Eighty-Four*	*On Nineteen Eighty-Four*	28%	70%	False	True
	Nothing to Envy	*The Heart of the Matter*	10%	43%	False	True
	Brave New World	*The Seventh Cross*	9%	43%	True	True
	You Can't Go Home Again	*SS-GB*	9%	41%	True	True
	The Talisman	*The Marriage Plot*	8%	41%	True	True
	Betrayal in the City	*Nothing to Envy*	8%	40%	False	False
	World at Risk	*The Spy Story*	7%	40%	False	False
	The Stars Look Down	*You Can't Go Home Again*	7%	40%	True	True
	The Testaments	*Notes of a Native Son*	7%	40%	True	True
	Tenth of December	*A Column of Fire*	7%	39%	True	True
Average			**10%**	**44%**	**60%**	**80%**

In this example snapshot, the Transformer model consistently produces higher similarity scores, with an average of 44%, compared to the Cosine Similarity approach's average of 10%. In terms of relevance, the Transformer model achieves an 80% relevance rate, indicating that 8 out of 10 recommendations meet the relevance criteria. This pattern of higher similarity and relevance scores is consistent across most recommendations, illustrating the superior performance of the Transformer-based model.

The following table provides a comprehensive comparative evaluation for the two recommendation approaches tested on 10 different reference books.

	Traditional Cosine Similarity Approach	Transformer Based Approach
Average Similarity Score	11%	44%
Recommendation Relevance Score	49%	68%
Overlapping Recommendations	15%	

We can clearly see that by using the transformer-based approach, we had a notable improvement in the quality of recommendations. The similarity scores obtained are significantly higher compared to the traditional cosine similarity method, reflecting the model's ability to capture deeper semantic connections between book descriptions. This results in a recommendation engine that is more effective at identifying books with similar content, themes, and styles, thereby providing users with more relevant and satisfying suggestions.

The integration of the "*all-MiniLM-L6-v2*" transformer model into our recommendation system represents a substantial advancement over traditional methods. By leveraging the model's advanced text representation capabilities, we achieve a more accurate and contextually aware similarity assessment, enhancing the overall performance of the AI with better book recommendations.

5. Conclusions

This thesis demonstrated the effectiveness of consolidating new technologies in building state-of-the-art AI systems. By leveraging advanced NLP techniques, such as Haystack 2.0, Hugging Face transformers, and Named Entity Recognition, we successfully developed an information retrieval-based question-answering system integrated with a book recommendation engine. This approach not only showcased the capabilities of modern AI frameworks but also highlighted the synergy achieved through their integration.

The primary contributions of this work include:

1. Development of an Intelligent QA System: By integrating information retrieval techniques with modern NLP frameworks, we built a chatbot capable of understanding and responding to user queries about books. The system utilizes the Google Books API to collect relevant information, ensuring the responses are up-to-date and comprehensive.

2. Implementation of NER for Query Understanding: The adoption of NER played a crucial role in extracting book titles from user queries. This step was vital for the accurate retrieval of information and recommendations. Despite the challenges in implementing NER, such as handling ambiguous and complex queries, the system demonstrated robust performance in identifying relevant entities.

3. Incorporation of a Recommendation Engine: Alongside the QA capabilities, a recommendation system was integrated to recommend books based on their similarity to other titles. By employing pre-trained transformers from HuggingFace, the system could offer personalized and contextually relevant book recommendations that perform significantly better than traditional methods.

5.1. Key Findings

Comprehensive Coverage of AI Technologies: Through detailed exploration, this thesis provided a comprehensive understanding of various technologies and frameworks pervasive for building AI systems. Concepts such as Transformers, which revolutionize natural language processing tasks, were unveiled, along with insights into the Hugging Face ecosystem, showcasing its pivotal role in facilitating NLP model development and deployment. Similarly, by examining the modular components and methodologies within Haystack 2.0, we gained valuable insights into its architecture and its applicability in constructing robust and scalable AI systems.

Accuracy and Performance: The system showed a high degree of accuracy in extracting book titles and retrieving relevant information. The use of transformers and advanced embedding techniques significantly enhanced the precision of responses. Similarly, leveraging transformers to build the recommendation system demonstrated huge improvements compared to traditional methods.

Scalability and Flexibility: The modular design of the system, particularly the use of Haystack 2.0 components, ensures scalability and flexibility. This allows for easy updates and enhancements, making the system adaptable to evolving needs and technological advancements.

5.2. Technical challenges and future improvements

Limitation of computational power and resources:

Developing AI systems faces significant constraints, particularly in accessing high computing resources necessary for working with large-scale data. Limited access to high-performance computing infrastructure hampers the ability to process vast datasets

efficiently, hindering the AI's capability to learn from diverse and extensive information sources.

Incorporating LLMs such as OpenAI's GPT can significantly enhance chatbot responses, fostering more natural conversations with users. This integration is facilitated by implementing a haystack GenerativeQAPipeline component, combining a Retriever and a Generator to effectively address user queries. By leveraging these advanced language models, chatbots can better understand and respond to user inputs, enriching the overall conversational experience.

Dependency on External APIs:

Reliance on the Google Books API algorithm may occasionally lead to inaccuracies, particularly when fetching information related to the wrong book, such as works analyzing the main book in question.

Runtime Efficiency:

While efficient runtimes were achieved for the question answering and Named Entity Recognition models, the integrated recommendation system's runtime could be improved. Therefore, further optimization is needed before deployment.

Data Limitations:

The dataset used to test the recommendation system is currently limited in scope. To ensure robustness and accuracy in deployment, a larger-scale database is essential.

Model Recency and Documentation:

Some of the models used in building the AI are very recent. Haystack 2.0 for instance was first released in December 2023 and the pipelines integrated in the AI are continuously being updated. While this is a positive aspect, it may pose challenges in finding documentation or support for building machine learning systems with the

Haystack 2.0 framework. Additionally, compatibility issues with older packages persist, necessitating migration to the latest versions of the framework.

5.3. Conclusion

In summary, the thesis demonstrated the potential of integrating advanced NLP techniques with information retrieval systems to create a sophisticated QA system with an embedded recommendation engine. The developed AI assistant provides accurate and relevant responses to book-related queries along with offering an integrated state-of-the-art book recommendation system, thereby enhancing the overall user experience. The insights gained and the challenges encountered during this research lay a solid foundation for delving into the intricacies of building intelligent QA systems. As the field continues to evolve, the groundwork laid here serves as a fundamental starting point for advancing the capabilities and effectiveness of intelligent QA systems.

Bibliography

1. Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv*. /abs/1810.04805

2. Ethayarajh, Kawin. (2019). How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings. 55-65. 10.18653/v1/D19-1006.

3. Gradio. (2024). *Gradio Documentation*. Retrieved from https://www.gradio.app/docs.

4. Hambarde, Kailash & Proença, Hugo. (2023). Information Retrieval: Recent Advances and Beyond. IEEE Access. PP. 1-1. 10.1109/ACCESS.2023.3295776.

5. Haystack. (2024). *Haystack Documentation*. Retrieved from https://docs.haystack.deepset.ai/docs.

6. Hugging Face. (2024). *Hugging Face Documentation*. Retrieved from https://huggingface.co/docs.

7. IBM. (2023). *Named Entity Recognition*. Retrieved from https://www.ibm.com/topics/named-entity-recognition.

8. Joseph Weizenbaum. 1983. ELIZA — a computer program for the study of natural language communication between man and machine. Commun. ACM 26, 1 (Jan. 1983), 23–28. https://doi.org/10.1145/357980.357991

9. Khurana, D., Koli, A., Khatter, K. et al. Natural language processing: state of the art, current trends and challenges. Multimed Tools Appl 82, 3713–3744 (2023). https://doi.org/10.1007/s11042-022-13428-4

10. Kumar, Deepak & Singh, Shoumya. (2024). ADVANCEMENTS IN TRANSFORMER ARCHITECTURES FOR LARGE LANGUAGE MODEL: FROM BERT TO GPT-3 AND BEYOND. International Research Journal of Modernization in Engineering Technology and Science. 06. 2582-5208. 10.56726/IRJMETS55985.

11. Lewis Tunstall, Leandro von Werra, & Thomas Wolf. (2022). *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*.

12. Liu, Z., Jiang, F., Hu, Y., Shi, C., & Fung, P. (2021). NER-BERT: A Pre-trained Model for Low-Resource Entity Tagging. ArXiv. /abs/2112.00405

13. Mattingly, William. Introduction to Named Entity Recognition, 2021 (2nd ed.). ner.pythonhumanities.com.

14. Nitin Hardeniya, Jacob Perkins, Deepti Chopra, Nisheeth Joshi, & Iti Mathur. (2016). *Natural Language Processing with Python and NLTK*.

15. Pratap Danget. (2017). *Statistics for Machine Learning: Build Supervised, Unsupervised, and Reinforcement Learning Models using both Python and R*.

16. Salehinejad, H., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2017). Recent Advances in Recurrent Neural Networks. *ArXiv*. /abs/1801.01078

17. Shelar, H., Kaur, G., Heda, N., & Agrawal, P. (2020). Named Entity Recognition Approaches and Their Comparison for Custom NER Model. Science & Technology Libraries, 39(3), 324–337. https://doi.org/10.1080/0194262X.2020.1759479

18. Zhao, W. X., Liu, J., Ren, R., & Wen, J. (2022). Dense Text Retrieval based on Pretrained Language Models: A Survey. ArXiv. /abs/2211.14876