Final Report: Natural Language Processing for Automated Fact Verification

CM3203 – One Semester Individual Project

Final Report

Computer Science BSc

Cardiff University

Author: Theodor Baur

Supervisor: Nedjma Ousidhoum

Moderator: Nathan Jones

Spring 2024

Abstract

Automated fact-checking is a crucial step to combatting the rising issue of misinformation, particularly online. This report implements an automated evidence retrieval system, ESOTERIC (Elasticsearch Semantic Optimised Text Extraction Retrieval from Information Corpus), and deploys this system in a web application. ESOTERIC is designed to identify evidence documents and passages from the FEVER dataset which are relevant to a factual claim, whilst operating on consumer-grade hardware. The system employs a novel approach by aiming to ask fact-finding questions to verify information from the claim using named entity recognition, answer-aware question generation, and dense passage retrieval to identify relevant documents, training a custom classification model to identify relevant evidence passages. Key findings show that ESOTERIC achieves recall rates of 77.37% for documents and 60.84% for passages, achieving a notably high F1 score of 60.56% for the retrieval of passages. Ultimately, this project contributes to the field by demonstrating the feasibility of using automated evidence retrieval on consumer-grade hardware within reasonable timeframes.

Acknowledgements

I would like to thank my supervisor Nedjma Ousidhoum for offering guidance, suggestions, and foundational resources for this project.

Content

1 Introduction

> *"Natural Language Processing is part of multiple automated fact-checking systems and related tasks such as rumour detection. It can be helpful at different steps in the verification process such as when identifying claims, retrieving evidence, etc.*
>
> *In this work, the student will choose the fact verification sub-task they would like to focus on and study the related literature. They will then implement an NLP, or an Information Retrieval (IR) based method for this subtask, analyse their results and share their insights."*

1.1 Motivation

Misinformation is a growing issue in society, and whilst it is not new, the adoption of the internet into people's daily lives seems to have magnified its power. It can be described as information which is fake or misleading and is spread unintentionally. (Mathew & T, 2022). Public internet platforms allow for the rapid dissemination of misinformative content which can sometimes elicit harmful social responses (Del Vicario, et al., 2016). One instance of this phenomenon was recorded by the World Health Organization, which found that exposure to misinformation about vaccinations led to approximately a 6-percentage-point reduction in the intention to get vaccinated among those who initially stated they would "definitely accept a vaccine." (Linden, 2022). One proposed method to try and help tackle the issue of misinformation is a corrective approach, wherein people are provided with a wider range of better-quality fact-checking facilities (Bernhard & Dohle, 2015).

Fact-checking is the task of assessing whether claims communicated are true. This process can be time-consuming if done manually, taking professional fact-checkers a considerable amount of time to parse through hundreds of sources to determine if a claim is true. Automated fact-checking aims to help speed up this process. One proposed framework for automated fact-checking can be modelled

as a series of components that can be represented using natural language processing tasks, which is shown in Figure 1. This pipeline includes three stages: claim detection, the process of deciding which claims to fact-check; evidence retrieval, the process of finding sources supporting or refuting the claim; and claim verification, the assessment of the veracity of the claim based on the retrieved evidence. (Guo, et al., 2022).

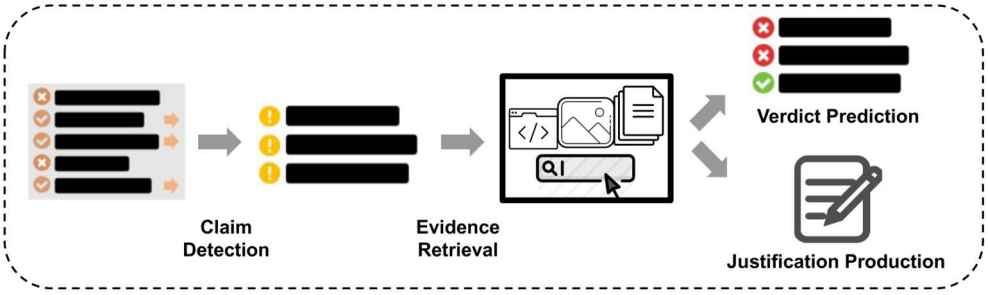


Figure 1: A natural language processing framework for automated fact checking. (Guo, et al., 2022)

The evidence retrieval stage is crucial in the fact-checking process as it sets the foundation for any downstream veracity judgements. The quality of the evidence retrieved directly affects the veracity reliability and accuracy of the fact-checking outcome. (Lee, et al., 2021)

We propose ESOTERIC: Elasticsearch Semantic Optimised Text Extraction Retrieval from Information Corpus, an evidence retrieval system, capable of retrieving source documents and passages supporting or refuting a given claim, from within a large corpus of information. This corpus is the pre-existing Fact Extraction and VERification (FEVER) dataset, covering a broad range of non-specialised topics, extracted from 5.45 million Wikipedia articles, as to let the system receive a wide catchment of potential users due to its general applicability, promoting the general use of automated fact-checking systems.

We intend to make this system accessible and attractive to the general population by wrapping the retrieval system in a simple user-friendly web application, allowing internet users immediate access to fact-checking facilities which in turn aims to curb the growing pertinence of online misinformation.

We aim to design a system that can be run on non-specialised hardware enhancing its usability across various platforms and devices, including machines

often used by individuals not equipped with high-end computing resources. This accessibility will democratise the ability to perform accurate fact-checking, thereby empowering a broader audience to engage in critical analysis and contribute to the fight against misinformation effectively, addressing the current lack of evidence-retrieval systems designed to be run on consumer hardware.

The development of a unique evidence-retrieval system is also aimed at supporting researchers who are looking to create new fact-checking or evidence-retrieval systems, by adding to the body of academic research into different stages of the fact-checking framework.

1.2 Problem Definition

The evidence retrieval process aims to retrieve relevant evidence documents and passages from a corpus of evidence, from a claim given by a user. An example is provided in Figure 2.

Claim: The world is flat.

Evidence:

[Spherical_Earth]

The roughly spherical shape of Earth can be empirically evidenced by many different types of observation, ranging from ground level, flight, or orbit. The spherical shape causes a number of effects and phenomena that combined disprove flat Earth beliefs.

Corpus:

[Spherical_Earth]

The roughly spherical shape of Earth can be empirically evidenced by many different types of observation, ranging from ground level, flight, or orbit. The spherical shape causes a number of effects and phenomena that combined disprove flat Earth beliefs.

[Earthworm]

An earthworm is a soil-dwelling terrestrial invertebrate that belongs to the phylum Annelida. The term is the common name for the largest members of the class (or subclass, depending on the author) Oligochaeta.

Figure 2: Example claim, retrieved evidence for claim, and evidence corpus.

Whilst there are many different existing solutions for evidence retrieval systems on large document corpora, with some specifically designed for the dataset being used in this paper, there is not much focus on making these evidence retrieval systems perform on consumer hardware.

2 Background

This section explains key information and concepts relating to different natural language processing techniques used in the solution, types of language models used, different retrieval strategies, different data stores, utilised datasets, and existing works.

2.1 Core Natural Language Processing Techniques

Natural Language Processing (NLP) is a field focused on understanding human language. It is important to note that NLP tasks aim to understand the meaning of words individually, but more crucially within the contexts that they are given in. The NLP field contains a range of subtasks such as Question Answering, Named Entity Recognition, and language translation. (Hugging Face, 2024)

2.1.1 Question Answering (QA)

Question Answering (QA) is a subtask of NLP that involves the extraction of answers from a given context to answer a given question. There two main variants of QA: Extractive QA and Abstractive QA.

Extractive QA – The model extracts an answer passage from a context for a question (Wang, et al., 2022), both given as inputs as demonstrated in Figure 3:

Question:	What shape is the Earth?
Context:	The roughly spherical shape of Earth can be empirically evidenced by many different types of observation, ranging from ground level, flight, or orbit. The spherical shape causes a number of effects and phenomena that combined disprove flat Earth beliefs.

Answer:	Spherical

Figure 3: Example inputs and outputs using Extractive Question Answering

Abstractive QA – The model generates free text from a context for a given question (Zafar, et al., 2024), both given as inputs as demonstrated in Figure 4.

Question:	What shape is the Earth?
Context:	The roughly spherical shape of Earth can be empirically evidenced by many different types of observation, ranging from ground level, flight, or orbit. The spherical shape causes a number of effects and phenomena that combined disprove flat Earth beliefs.
Answer:	The Earth has a spherical shape

Figure 4: Example inputs and outputs using Open Generative Question Answering

The implemented solution primarily makes use of Extractive QA to help score documents based on how well they answer questions relating to the claim.

2.1.2 Question Generation

Question generation (specifically automated question generation for the scope of this paper) is a task where questions are generated based on a natural language paragraph (Mulla & Gharpure, 2023). Question generation systems can be answer-aware or answer-unaware. Answer-aware systems require a specific answer and context to be given in order to generate a question. Answer-unaware question generation systems only require a context to generate a question (Do, et al., 2023).

2.1.3 Named Entity Recognition (NER)

Named Entity Recognition (NER) is a subtask of NLP that involves the extraction of named entities in a piece of unstructured text. A named entity refers to a key subject in a piece of text such as names, companies, locations, times, or topics that have proper names or noun phrases that act as unique identifiers (Nadeau & Sekine, 2007). For example, consider the passage:

*"**Barack Obama** (a former president of the **United States**) drove a **Kia Soul**."*

The following named entities can be extracted using NER:

["Barack Obama", "United States", "Kia Soul"]

The implemented solution makes use of NER to pick spans of text out from the claim to match with document titles or document text.

2.2 Language Models

Different language models are used for different processes. This section aims to explain how language models work fundamentally, then list the key language models used in this paper.

2.2.1 Neural Networks

A neural network is a structure comprised of connected nodes called neurons, connected by edges. A neuron is a single node that takes one or more inputs, applies weights to these inputs, then performs a computation on these inputs to produce an output. A neural network consists of a multitude of connected neurons organised into layers, typically starting with the input layer, into the intermediary hidden layers, finally into the output layer. Each neuron and edge are typically assigned a weight value.

An input layer passes data directly into the first hidden layer using a set of neurons. Each hidden layer is a set of neurons that receives data from the previous layer, applying weights and transforming them using non-linear functions and passing the output onto the next layer. The output layer receives data from the final hidden layer and transforms this data to produce a final result. (Lek & Park, 2016)

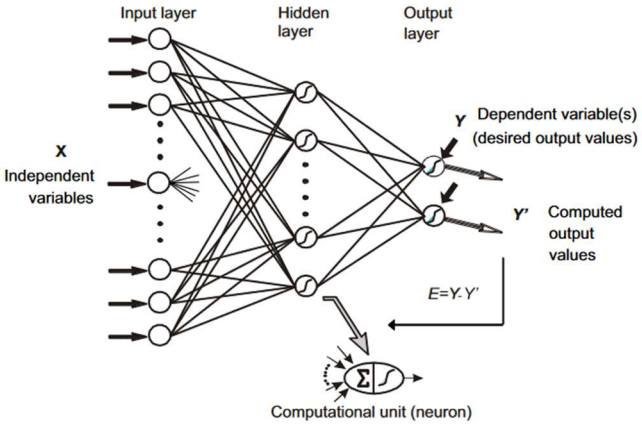


Figure 5: Three-layered feed-forward neural network with one input layer, one (or more) hidden layer(s), and one output layer: X, independent variables; Y, dependent variables; and Y0, values computed from the model. (Lek & Park, 2016)

Neural networks provide the basic framework for machine learning models.

2.2.2 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a type of neural network that contains a "memory" of previous inputs to influence the current input and output. A traditional neural network assumes that each set of inputs and outputs are independent of each other (Schmidt, 2019).

In an NLP context, past text tokens in a passage are often needed as context for a task. For example, consider the passage:

"My family is German, so I speak a bit of ___."

When trying to predict the next word in the passage, the context provided by the earlier part in the sentence is crucial, as the word "German" would strongly imply that the next word would also be "German". A non-RNN would struggle to generate the next word as it does not have a memory of the earlier context in the passage.

2.2.3 Attention Mechanism

Attention Mechanism is a method employed by neural networks to focus on specific parts of input data that may be more relevant, therefore require more

focus when producing an output. In the context of NLP this can mean that certain words or sequences can be given a higher weight, showing their higher relative importance within a larger word sequence (Vaswani, et al., 2017). Take the NLP task of translating the following phrase from German to English:

"Kannst du mir helfen diesen Satz zu uebersetzen?"

Translating this sentence word-by-word would result in the following English translation:

"Can you me help this sentence to translate?"

Using attention mechanisms, higher weights can be applied to words such as "Kannst" (can) and "helfen" (help), along with the main action "zu uebersetzen" (to translate), meaning that, when generating a response, the model can more accurately capture the intention of the sentence as understood in English:

"Can you help me to translate this sentence?"

2.2.4 Tokenisation

Tokenisation is the process of splitting individual pieces of text, which are normally words or pieces of punctuation, into "tokens", which are designed to be readable by a specific language model. This is normally done by first splitting text into tokens, then mapping these tokens to their specific token ID. These token IDs can then be read by a language model and converted into a useful vector representation. (Webster & Kit, 1992)

2.2.5 Transformer Architecture

Transformer Architecture is a deep learning architecture that consists of two pieces, the encoder and the decoder. The encoder accepts inputs, passing on data into the decoder. The decoder accepts multiple outputs from the encoder to generate a prediction (Vaswani, et al., 2017).

Encoder – The encoder transforms embedding IDs from the tokeniser into vector representations using bi-directional self-attention as well as positional encoding. This means that for a given text passage, the encoder considers the

entire sentence and weights important words in the sentence using self-attention, and records the positions of the words using positional encoding. Using these mechanisms, a numerical vector representation of the sentence can be generated and passed into the decoder.

Decoder – The decoder's purpose is to generate an output step-by-step, considering both the original output received from the encoder to maintain context, and any previous outputs generated by the decoder to decipher what to generate next. The decoder also uses self-attention to help weight important words, however, it masks itself from seeing future tokens like the encoder, making it uni-directional. It is also autoregressive, meaning that it uses past outputs to predict future outputs.

The encode-decoder structure can be seen below in Figure 6.

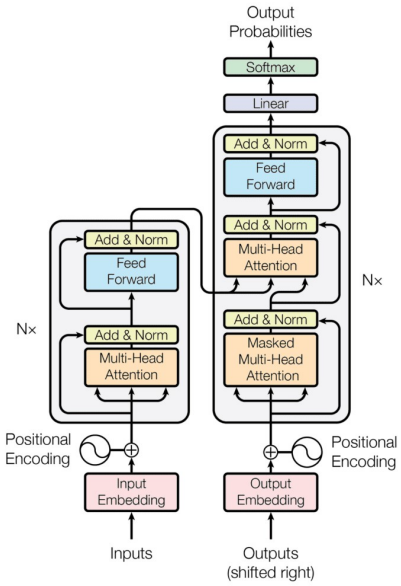


Figure 6: The encoder-decoder structure of the Transformer architecture. (Vaswani, et al., 2017)

Consider the NLP task of generating a textual response for the following question:

"How are you?"

1. The tokeniser first splits the question into the following tokens:

["How", "are", "you", "?"]

These tokens are then mapped to token IDs. Using the token IDs the encoder can then generate vector representations of these words.

2. Using self-attention, weights are applied to each token to indicate the significance of each token. This is done using bi-directional understanding, meaning that for each token, any past or future tokens relating to it can be seen.
3. Positional encoding is applied to give the model information about the word order of the question. This is important as the question can take on a different meaning if the positions of words are not considered. Consider the swapping of the following two words:

*"**How** are **you**?"*

*"**You** are **how**?*

4. Using self-attention, the decoder can generate the output sequentially (token-by-token), considering both the encoder's output and its previous outputs to produce the next token. This architecture is central to tasks like translation or text generation, where understanding the context and sequence is crucial.

The encoder process is used by all language models in this project, and the decoder process is also used by the T5 model (see 2.2.10).

2.2.6 Vector Embeddings

In NLP, words can be represented as vectors, with words that are closer together in the vector space being more contextually similar. For example, the word "dog" and "pet" may be relatively close to each other in the vector space. These vectors often have hundreds of dimensions to help better represent relationships between words such as synonyms, antonyms, and other linguistic patterns. (Liu, et al., 2017)

2.2.7 Language Models

Language models are machine learning models designed to understand and generate natural language, as well as perform different NLP tasks. Modern language models are normally pretrained on a large corpus of text to build an understanding of language and grammar patterns. Once pretrained, language models can be fine-tuned to perform a specific NLP task, such as NER, using a dataset containing samples relating to that specific NLP task. (Devlin, et al., 2019)

Foundationally, language models rely on a neural network architecture to process a set of word vector representations into a deeper understanding of text. A language model can produce a range of outputs such as: predicted next word tokens, answers to questions, attention weights that indicate how much focus was given to different input tokens, or contextual vector embeddings to represent each token within the input sequences context.

Language models make use of RNNs to "remember" previous inputs, helping to understand context in longer passages of text. Attention mechanism is used inside the neural network to weight more contextually important words, whilst using transformer architecture to take input sequences and produce outputs step-by-step while referring back to the encoder and previous decoder outputs.

Large Language Models (LLMs) are characterised by their size. They are often pretrained on a far larger text corpus than traditional language models, using a much larger neural network. This scale allows them to capture a much broader range of linguistic patterns and contextual nuances. LLMs sometimes exhibit emergent abilities not explicitly programmed or expected such as explaining text (Wei, et al., 2022) or in the case of this project, making judgements if a set of evidence passages supports, refutes, or does not provide enough evidence for a factual claim.

2.2.8 Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers (BERT), is a language model made by Google in 2018, based on the transformer architecture. BERT

was pre-trained using the Toronto BookCorpus (containing 800 million words), and English Wikipedia (containing 2.5 billion words) (Devlin, et al., 2019). BERT is an encoder-only model, meaning it does not contain a decoder. This means that it can convert word tokens into a vector representation, which can help solve NLP tasks, but BERT does not have a mechanism to decode these vector representations back into word tokens. It was regarded as significant due to its capability to understand both left and right contexts of a word in a sentence, unlike previous models that primarily focused on one-directional understanding. BERT set a new standard in the NLP field and as a result, many different language model variants are adaptations of BERT.

2.2.9 DistilBERT

The DistilBERT model is a smaller variant of the BERT model. The DistilBERT model is 40% smaller than BERT and is shown to retain 97% of its language understanding capabilities whilst being 60% faster than BERT. (Sanh, et al., 2020). The improved speed and smaller size make DistilBERT the most viable language model for certain use-cases in this project due to the performance limitations of my personal computer and laptop. DistilBERT is also an encoder-only model.

2.2.10 T5

The T5 model is an encoder-decoder model, unlike BERT and DistilBERT, meaning it can receiver word tokens, encode the tokens into vector representation, and decode the vector representation back into word tokens. T5 is already pre-trained on a multitude of NLP tasks such as translation and summarisation (Raffel, et al., 2023). The T5 model is naturally suited to tasks like text generation or answer-aware question generation. The system in this report makes use of the T5 model for extracting key information out of claims, and generating questions based on this key information.

2.2.11 Mistral

Mistral is a decoder-only large language model which operates at a 187 times reduced cost compared to GPT-4 models and is capable of outperforming Meta's much larger `LLaMA 2 70B` LLM (Jiang, et al., 2023). It is a generative text model with 7 billion parameters that demonstrates impressive performance as well as a relatively low size and cost compared to competitor LLMs. It is open source and publicly available using either the Hugging Face hub (Hugging Face, 2024) or Mistral's dedicated API service (Mistral AI, 2024).

2.2.12 Hugging Face

The Hugging Face platform contains a hub that hosts language models, datasets, and other NLP-related content. It hosts a wide variety of language models that have been fine-tuned for specific and niche purposes, which have been uploaded by users. This project makes use of some of these models. The models can be accessed using the Transformers library (which is maintained by Hugging Face).

2.2.13 Transformers Library

The Transformers library provides an API that enables users to perform a range of machine learning tasks such as NLP, computer vision, audio classification, and multimodal tasks (Hugging Face, 2024). It is used in this project to download language models from the Hugging Face hub, train and fine-tune language models, and perform tokenisation.

2.3 Retrieval

This section explains some of the different methods used to retrieve text from a corpus.

2.3.1 Lexical Similarity

Lexical Similarity is the measure to which degree two given word sets are similar, based on the intersection of words. It does not account for the semantic meaning

of the words. Two texts that receive a lexical similarity of 1 suggest that they have a complete overlap of words. Consider the following passages:

"The cat sat on the mat."
"The feline rested on the rug."
"The cat ate on the mat."

The first passage and the last passage would receive a high lexical similarity score as they have a high overlap of words, however the first passage and the second passage would receive a low lexical similarity score as they share less words, of which are common words. The final system makes use of lexical similarity when ordering document texts from a database, previous versions of the system featured lexical similarity more heavily.

2.3.2 Semantic Matching and Semantic Similarity

Semantic Matching is a technique used to detect if two textual passages have a similar meaning (Shvaiko, et al., 2007). Consider the previous passages in 2.3.1, the first passage and the second passage will likely return a semantic match due to their similarity in meaning, even though much of the vocabulary is not shared between the two passages. Conversely, despite sharing similar vocabulary, the first and third passage will not return a semantic match as they do not share a similar meaning. Our solution makes use of semantic matching to rank documents and select passages that have a similar meaning to the input claim.

2.3.3 BM25

BM25 is a widely used lexical similarity measure. It takes a query and a set of documents, ranking them based on the relevance the documents have to the search query. It considers the frequency of each term in the document, how frequently the term appears in other documents (higher scores are given if the term is rarer in other documents), the document length, and a number of other factors. BM25 score was the primary lexical similarity measurement used in the final system.

2.3.4 Dense Passage Retrieval (DPR)

Dense Passage Retrieval is a more advanced retrieval technique than BM25, classing as a semantic matching technique. It is designed to receive a corpus of document texts, represented as vector embeddings, and a query, also represented as vector embeddings. The retriever then uses some sort of similarity metric such as dot product similarity (a method of measuring the similarity between two vectors by comparing the angle of two vectors in the vector space), cosine similarity, or Euclidean distance to determine the closest document vectors to the query in the vector space (Karpukhin, et al., 2020).

Embeddings are normally encoded using the same model, however, sometimes a dual-encoder DPR system is used. A dual-encoder system encodes the query using a special model, specifically designed to convert the query into vector representation that captures semantically important information in the query, and encodes the document corpus using a context encoder, which is specifically designed to capture important semantic information and properties in the corpus. In a dual-encoder system, both encoders are specialised to allow each encoder to tailor its parameters to better handle the characteristics of its respective inputs, such as different lengths or complexities found in queries versus documents (Devendra Sachan, et al., 2023).

The final system makes use of a Dense Passage Retriever that uses a custom dual-encoder system specifically designed to encode questions as queries, and documents as contexts.

2.3.5 Haystack

Haystack is a Python framework created by deepset containing tools designed to help build applications with large language models and NLP techniques. It supports models hosted on the Hugging Face hub as well as a number of different hosting platforms. It also can easily access and format data for NLP tasks that are stored on a range of different data hosting platforms such as Elasticsearch, which was used as data storage for this project. It contains an integrated DPR tool (Deepset, 2024) which is used in this project.

2.4 Data Storage on Elasticsearch

Elasticsearch is a data storage system and search engine specifically designed to handle large volumes of data and perform full-text search efficiently on this data. It is built upon another high-performance full-text search engine library called Lucene. It makes use of an inverted index to help map words in text back to the address of their original documents as well as having inbuilt processes to sort documents by BM25 score. It also contains its own query language that provides custom text-match searching features. Crucially, Elasticsearch also provides native support for high-dimensional word embedding vectors, meaning that word embeddings can be processed before runtime and then accessed at runtime, cutting out the time-consuming step of encoding passages of text.

2.5 Key Datasets

This section introduces the dataset which we will use for our document corpus (the FEVER dataset), and two datasets which are used to train models found in our final solution.

2.5.1 FEVER Dataset

The FEVER dataset is comprised of 184,445 human-generated claims, each claim is labelled as SUPPORTED, REFUTED, or NOTENOUGHINFO. Each claim was made by mutating sentences from the introductory sections of Wikipedia articles, the label reflecting the veracity of the claim, given the set of evidence sentences from the Wikipedia articles. The portion of the dataset used in this report follows the structure in shown in Figure 7. The evidence document is a tuple containing the Annotation ID, Evidence ID, Document ID, and Sentence ID. The test set follows the same format as the training and development sets; however, it does not have a label, evidence, or verifiable field.


```
{
   "id": 89891,
   "verifiable": "VERIFIABLE",
   "label": "REFUTES",
   "claim": "Damon Albarn's debut album was released in
2011.",
```

```
   "evidence": [
      [
            [107201, 120581, "Damon_Albarn", 17]
      ]
   ]
}
```

Figure 7: FEVER training and development set example value.

The FEVER dataset also includes the source documents which the claims are referring to. These source documents are extracted introductory sections of Wikipedia articles and contain an ID field (referring to the Wikipedia URL of the source document), a text field containing the introductory section, and a lines field which contains the text field with preserved formatting. There are 5,416,537 separate articles split between 109 JSONL documents, each of which follows the same structure as the example shown in Figure 8.

```
{
   "id": "1928_in_association_football",
   "text": "The following are the football -LRB- soccer RRB-
   events of the year 1928 throughout the world . ",
   "lines": "0\tThe following are the football -LRB- soccer
   -RRB- events of the year 1928 throughout the world .\n1\t"
}
```

Figure 8: FEVER Wikipedia set example value.

2.5.2 Stanford Question Answering Dataset 1 and 2 (SQuAD)

The SQuAD dataset is a collection of question-answer pairings that were created by crowd-workers around a set of Wikipedia articles. The SQuAD 1 dataset contains 100,000 questions with answers within the Wikipedia article set (Rajpurkar, et al., 2016), and the SQuAD 2 dataset contains 50,000 questions that were unanswerable within the Wikipedia set (Rajpurkar, et al., 2018). Specifically, the SQuAD datasets contain answer, question, and context fields. This dataset is one of the most notable QA datasets and is often used to train QG models and QA models. The final system does make use of models that use the SQuAD dataset.

2.5.3 Natural Questions (NQ)

The NQ dataset is a large-scale QA dataset, like SQuAD, produced by Google. Its questions are formulated using real Google search queries regarding certain contexts. It follows a similar structure to the SQuAD dataset. This dataset was used to train models for a dual-encoder system used in the final solution. (Kwiatkowski, et al., 2019)

2.6 Existing Works

The Fact Extraction and VERification (FEVER) shared task was proposed in 2018 to raise interest in automated fact verification. It uses the FEVER dataset and required participants to develop systems to retrieve evidence and predict the truthfulness of human-generated claims against a set of textual evidence retrieved from the Wikipedia extracts in the FEVER dataset.

Due to the nature of being a shared task, it has several pre-existing solutions developed by researchers and developers. This paper discusses the Nie et al. (2018) implementation, which was the highest scoring solution to the shared task, as well as the FEVER baseline solution, which was the original solution proposed that achieved lower results, acting as a baseline solution to compare other solutions to (Thorne, et al., 2018).

3 Methodology

This section states the specification that the system needed to fill, and the approach used to develop the solution. The section then briefly describes the overall design of the system, before going into detail about how each part of the system was implemented, what decisions were made during implementation, and the justifications for those decisions.

3.1 Specification

The system was designed to take a factual claim from a user to be checked, retrieving documents that are relevant to either support or refute the claim from a document corpus. It then needed to retrieve relevant passages that either supported or refuted the claim, at which point these passages and documents needed to be presented to the user, sorted by their overall likelihood that they support or refute the claim.

The second objective of the system was to present this information in a website in a user-friendly manner, as such it needed to present both the documents and passages in a clear manner to the user. The following requirements were set before the undertaking of the project to help:

1. **Retrieval of Evidence** - The system must efficiently search and retrieve relevant evidence passages to the user's claim, retrieved from the FEVER dataset's Wikipedia extracts.
2. **Retrieval of Documents** - The system must efficiently search and retrieve contextually relevant documents to the user's query.
3. **Retrieval of Passages** - The system should identify and extract specific passages from documents that are contextually relevant to the user's query.
4. **Presentation of Evidence -** Retrieved evidence must be presented in a user-friendly format that clearly highlights the relevance to the query.
5. **Performance of System** - The system should be able to perform retrieval within a reasonable time frame for the user, meaning that queries should not take excessively long, as this could discourage users

from engaging with the system. The system should also run on the device specifications listed in Section 3.1.1.

3.1.1 Resource Limitations

The task solution was created on a personal computer, therefore any models used were required to run on this computer. In addition, the solution was designed to operate in the context of a Flask web-application, therefore it was required to finish execution within a reasonable time, without extensive resource use. In practice this resulted in decisions being made to prioritise performance or faster execution over accuracy. Google Colab was used for small parts of development. Exact specifications for the computer are listed below:

Type	**Component**
CPU	AMD Ryzen 5 4500U 2.38 GHz
GPU	None
RAM	8GB DDR4 RAM
Storage	1TB SSD

3.2 Approach

The project development followed an iterative methodology strategy. Due to the existence of the FEVER shared task, we had access to previous systems alongside reporting on the effectiveness of these systems. This meant that we could use some effective methods from these systems as inspiration for our own.

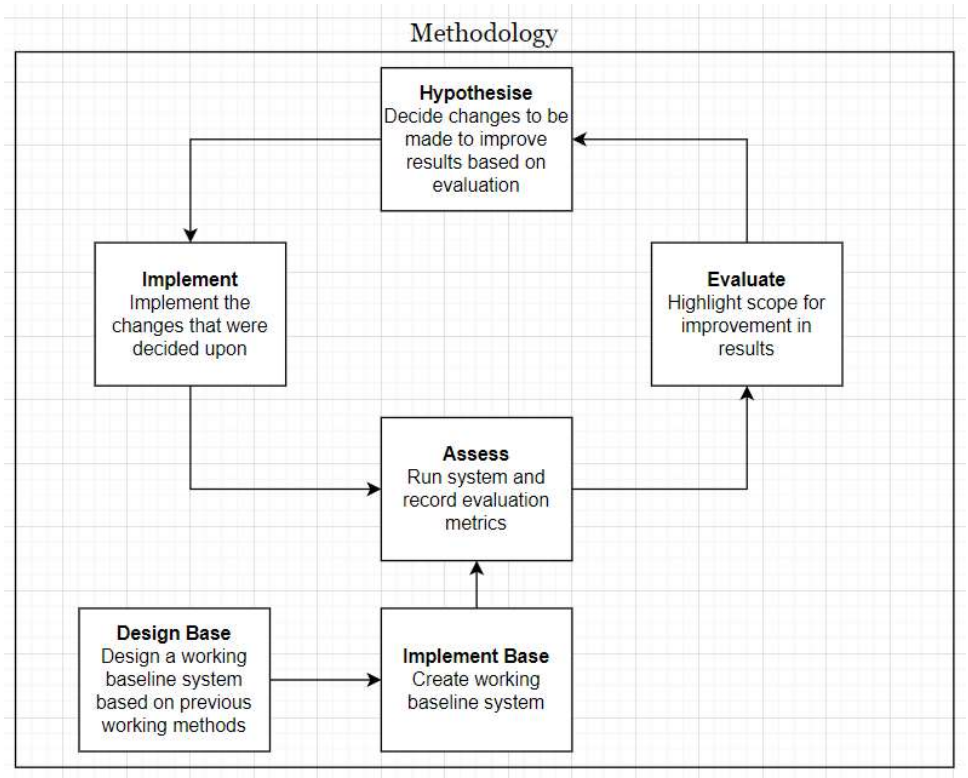


Figure 9: Iterative Methodology Diagram

The process began by reading the analysis reports for the other systems and choosing methods and techniques that were reported to be effective to design a baseline system. The design for the baseline system was then implemented. The purpose of the baseline system was to build a foundation that could be analysed and improved on iteratively.

The next step in the process was to assess the system based on certain evaluation metrics. These metrics can change based on the specific system being assessed but generally tend to be the precision, recall, OFEVER and F1 score of

both the passage and document retriever (explained in 4.1), alongside individual metrics for each step of the passage and document retriever.

Based on these metrics, we can evaluate which parts of the system are ineffective at what they are trying to achieve. For example, the recall of a document retriever could be brought down by an individual ineffective step in the function, which will be identified in the assessment (mentioned previously in Figure 9).

At this stage parts of the system deemed to need improvement have been identified, therefore potential changes to these parts can be hypothesised.
Once the changes to the system have been hypothesised, they can be implemented in a second iteration of the system, at this stage the second iteration of the system can be assessed, leading to an assess-evaluate-hypothesise-implement cycle as seen in Figure 9.
This methodology allows for several advantages:

1. The cycle ensured that the system was constantly being refined.
2. The cycle ensured that each part of the system was continuously being evaluated for effectiveness.
3. It ensured that any changes made to the system were being evaluated both independently of the system and in the context of the whole system to see if the changes are effective.
4. The hypothesis-driven approach allowed for evidence-driven improvements on parts of the system that were identified as being weak.
5. The creation of a baseline system ensured that there was a control system to compare different systems to.

3.3 Design and Implementation

The system has two separate modules: a document retriever, and a passage retriever. This structure is like that of the FEVER Baseline (Thorne, et al., 2018) and UNC-NLP systems (Nie, et al., 2018). The document retriever's function is to take an input claim from the user and use NLP techniques to retrieve documents from an Elasticsearch database which contains the document titles, texts, and vector embeddings of the entire FEVER evidence corpus (consisting of all 5.45 million Wikipedia pages) that can be used to assess the veracity of the claim. The passage retriever's function is to take the original claim and an array of documents (acquired using the document retriever) and select the passages in the documents that can be used to assess the veracity of the claim. With the combination of these two modules, a user can enter a claim into the system and receive passages that validate the veracity of the claim. The structure can be seen in Figure 10.

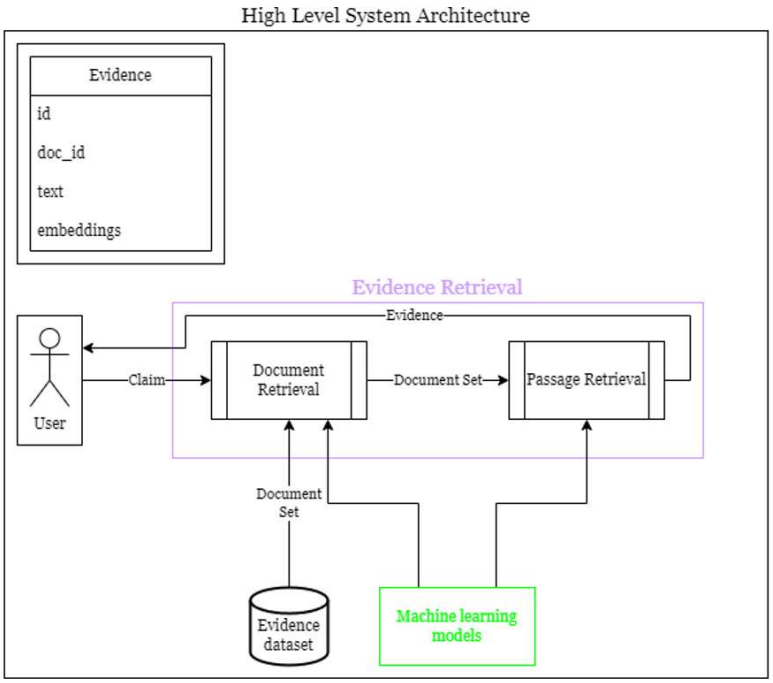


Figure 10: High-Level Data Flow Diagram

Once implemented, the evidence retrieval system was run inside a Flask application and the dataset was stored in an Elasticsearch database. This means that users could view a website containing the evidence retrieval system which

communicates with an external Elasticsearch database. In this project, both the Elasticsearch database and the Flask application were run locally, however both can be easily run on dedicated servers. Additionally, a final step was added which was only present on the web application. The application would take the retrieved evidence set and claim and use a Mistral large language model to predict the truthfulness of the claim, giving a justification based on the evidence available. This was also displayed to the user alongside the evidence passages. A diagram showing this architecture can be seen below in Figure 11:

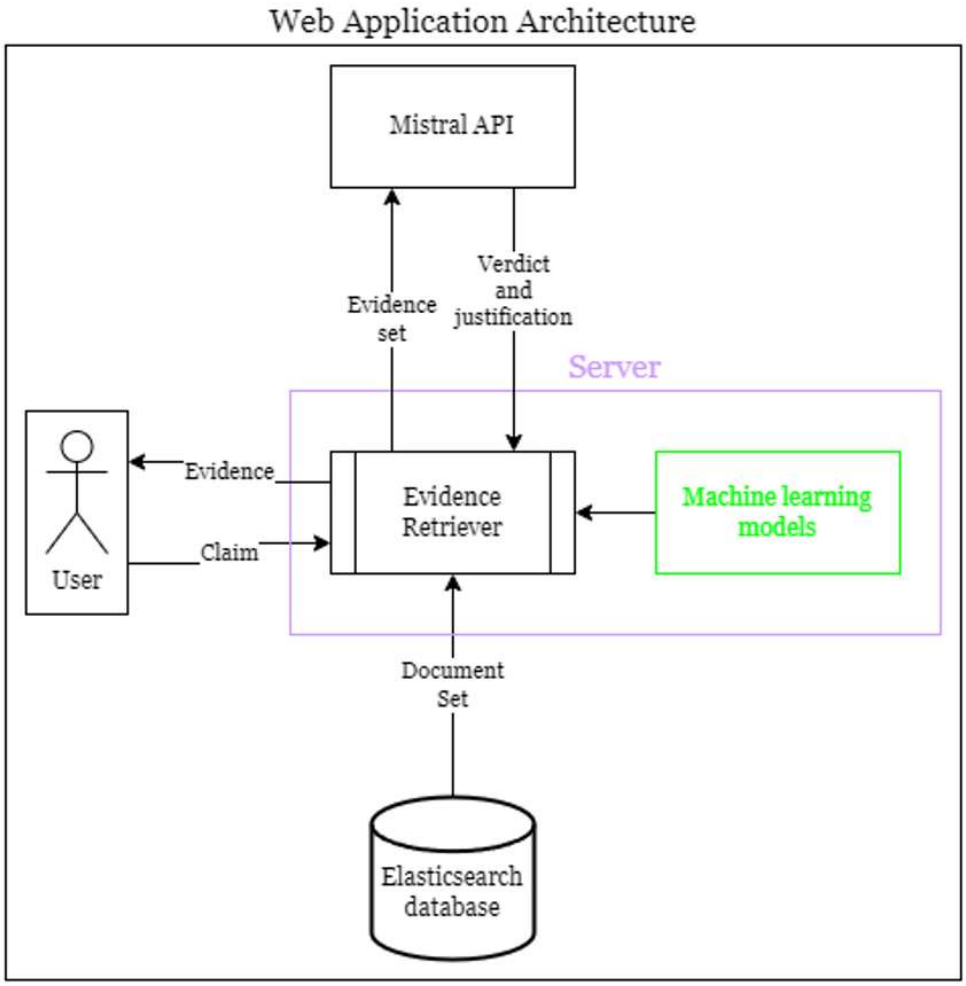


Figure 11: Web Application Diagram.

The final evidence retrieval system contains a range of different components in both the document retriever and the passage retriever. The architecture of both the document and passage retriever is shown in Appendix B.

3.3.1 Document Retriever

The structure of the document retriever can be seen below in Figure 12.

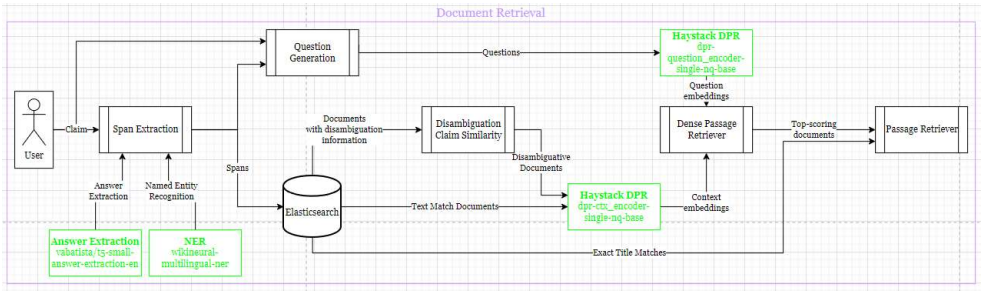


Figure 12: Document Retriever Diagram.

The final document retriever first takes the input claim given by the user. It then extracts all named entities from the claim using an NER model as well as all answer spans (slice of words from a text) using an answer-extraction model. For example, consider the following claim:

Named Entities**, **Answers

*Claim: "**Telemundo** is an **English-language television network.**"*

At this point the system runs two parallel pipelines, the Title Matching pipeline, and the Text Matching pipeline. The **Title Matching** pipeline takes the extracted spans from the claims and searches the database for documents that contain a document title that exactly matches any of the extracted spans (case-insensitive). It then searches the database for any documents containing titles that exactly match the extracted spans but have "disambiguation information" (as detailed in the Nie et al. implementation and below in Figure 13).

Rule: If the claim contains a span of text that is an exact match of a document title, however the document title contains disambiguation information in parentheses, the matching will be considered without considering the text in the parentheses.
Claim: Savages was exclusively a German film. **Retrieved Documents**: **`[Savages]`**

[Savages_(Band)]
[Savages_(2012_film)]

Corpus:

[Savages]
[Savages_(Band)]
[Savages_(2012_film)]
[Savages_Film]
[Noble_Savages]

Figure 13: Disambiguation information rule, as detailed in the Nie et al. implementation.

Following this, disambiguative documents are then ranked according to their semantic similarity with the claim and a cutoff score and document limit is applied to the disambiguative documents. For example, consider the following spans:

Spans: ["Telemundo", "English-language", "television network"]

*Retrieved Document Titles: ["Telemundo", "Telemundo **(TV show)**", "English Language", "Television Network"]*

Discarded Document Titles: ["Al Rojo Vivo (Telemundo)", "English Language GCSE"]

The title-matched documents are then split into a group containing documents that were retrieved using an exact match, and a group containing documents that have disambiguated information in the title. The disambiguative documents are then separated for further ranking using the question generation ranking whilst the documents retrieved with an exact title match are automatically assigned a document score of 1 and are passed into the passage retriever.

The **Text Matching pipeline** is run in parallel to the Title Matching pipeline. Using the extracted spans the database is then searched for all documents that contain a word inside the document text that matches any of the extracted spans. These are automatically ordered by Elasticsearch in order

of BM25 score. Of these documents the top 1000 are retrieved. These are concatenated with the disambiguative documents for question generation ranking.

A separate question generation process also simultaneously takes place at the same time as the Text Matching and Title Matching pipelines. Using the extracted spans from the answer extraction model, each extracted span is treated as an answer to a potential question regarding the claim, whilst the claim itself is treated as the context of a potential question. Using an extractive question generation model several potential questions are generated from these answers and contexts. An example is provided below:

*Claim: "**Telemundo** is an **English-language television network**"*

*Answer Spans: ["**Telemundo**", "**English-language**", "**television network**"]*

Generated Questions: ["What is the name of the English-language television network?", "What language is Telemundo?", "Telemundo is an English-language what?"]

Additionally, each claim also has a manually added polar question to the list of questions. This is achieved by using a grammatical rule-based approach or, in difficult cases, the same question generation model is used, but with the answer set as "No".

Once the questions have been generated, the document texts of both the disambiguative documents, and documents obtained using the Text Matching pipeline are converted alongside their titles using a dense passage retrieval context encoder model. Each question is then encoded using a dense passage retrieval question encoder model. The documents are retrieved and assigned a document score based on the distance in the vector space between their document texts and each of the generated questions. A cutoff score is applied, and these documents are then passed into the passage retriever alongside the documents retrieved by the Title Match pipeline with an exact title match.

3.3.2 Passage Retriever

The structure of the passage retriever can be seen below in Figure 14.

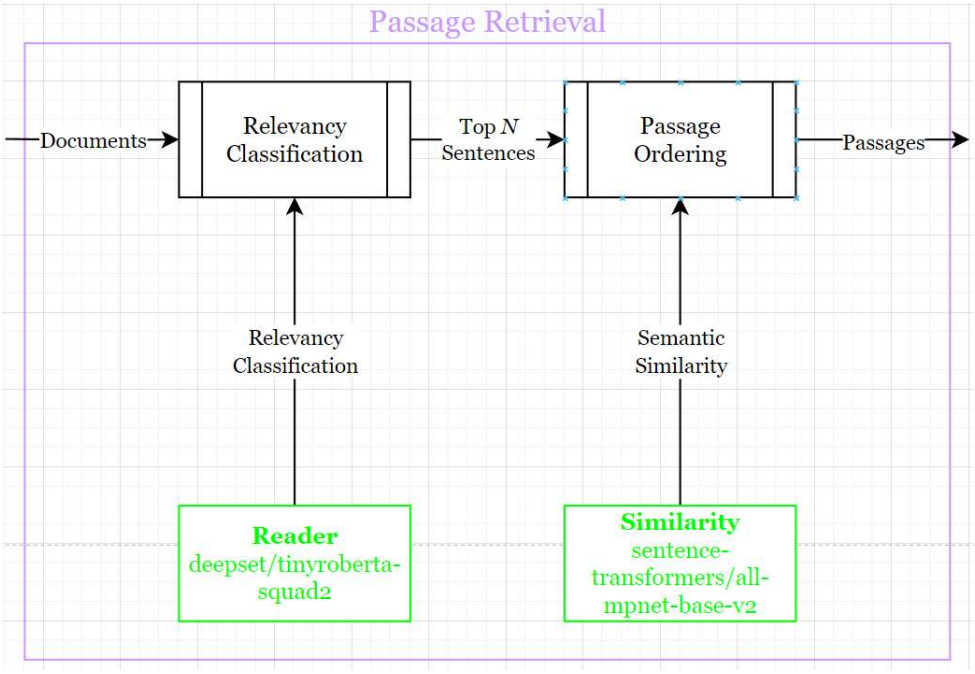


Figure 14: Passage Retrieval Diagram

The passage retriever then splits the document texts into individual sentences. After this these sentences are put through a fine-tuned language classification model which was trained to classify whether a sentence was relevant or not to justify a claim. This model was specifically trained on the FEVER dataset. The final passage scoring metric is retrieved by calculating the semantic similarity score between each passage and the claim. Each passage is then sorted by passage score.

3.3.3 Classes

Three basic classes were created to house evidence (see Figure 15):

- **Evidence** – Designed to represent entire evidence documents, these can contain evidence sentences.
- **Sentence** – Designed to represent individual sentences or passages.
- **Evidence Wrapper** – Designed to act as an output structure for a single claim and its retrieved sets of evidence documents and passages.

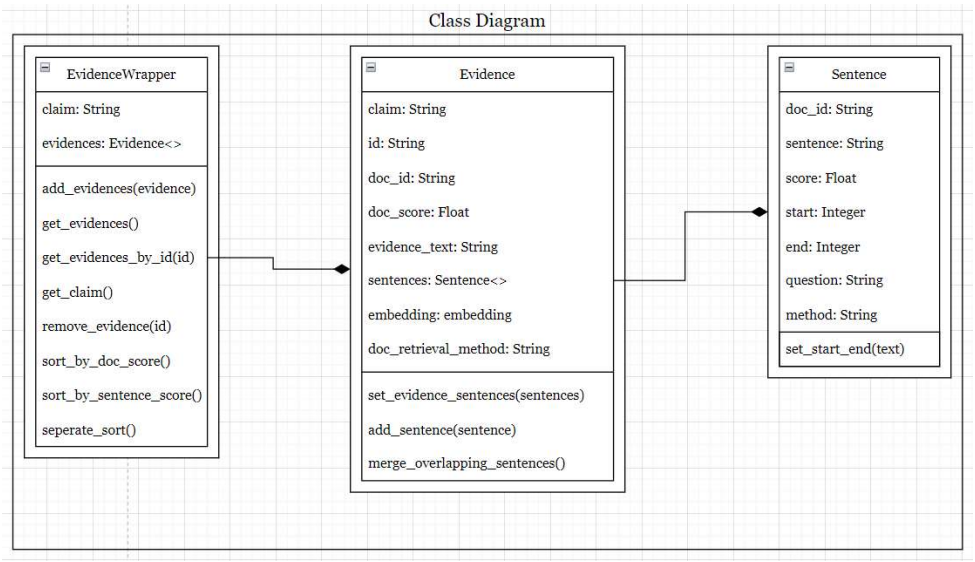


Figure 15: Class Diagram.

An `EvidenceWrapper` object is initialised when the retrieval starts, it must be preloaded with a claim. In the process of the evidence retrieval, evidence documents in the form of `Evidence` objects are added. The `EvidenceWrapper` contains a range of sorting functions and methods to add, remove, and export different `Evidence` objects.

The `Evidence` class represents a single evidence document. It must be initialised with evidence text. After initialisation, `Sentence` objects can be added which represent individual evidence passages. Each `Evidence` object contains a document score, representing how relevant the document is to the claim. It also contains methods to set and add sentences, as well as merge duplicate or overlapping sentences which are retrieved using separate methods.

The `Sentence` class can be initialised with a single evidence passage. It also has an associated passage score, representing how relevant the passage is to the original claim, as well as the start and end indices of the passage within the parent document text. It contains a method to set these indices for a given document text.

3.3.4 Elasticsearch Document Store

Basic Design

The final system makes use of an Elasticsearch document store to store all 109 JSONL files (containing 5.45 million Wikipedia entries seen in 1.2). From the JSONL files, only the `id` and `text` fields were used and loaded into the database. The remaining `lines` field only contained the text from the `text` field, with added passage breaks and tags, therefore was discarded. The `id` and `text` field were copied directly into a new field `doc_id` in the Elasticsearch index as these fields needed to be accessed in both the Title Matching and Text Matching stage of retrieval. Finally, the `text` field was also encoded into vector embedding representations using a DPR context encoder, which would later be used when using a DPR to re-rank disambiguated and text-matched documents to how well they match generated questions about the claim. This process can be seen below in Figure 16.

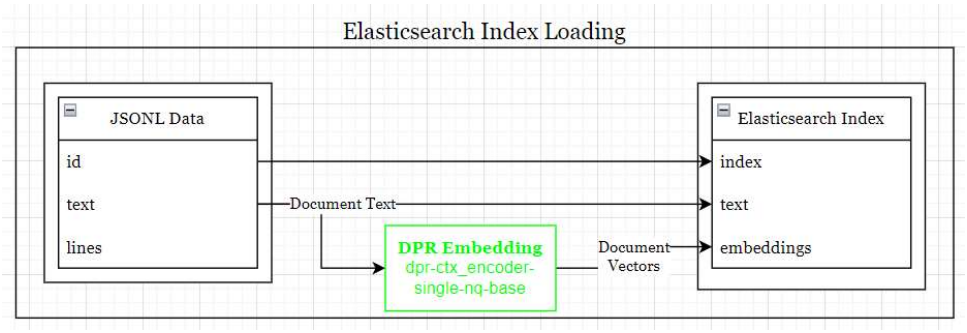


Figure 16: Elasticsearch database loading from JSONL files.

Justification

In the Title Matching and Text Matching pipelines both the title field (`doc_id`) and text field (`content`) needed to be parsed for strings using a text search. Elasticsearch makes use of an inverted index to help quickly search for terms found in the database (Kathare, et al., 2021), therefore was considered a good choice for this use-case. An inverted index is a data structure that is used in databases to point to specific rows that contain certain words. If a database has a column that contains text, this text can be split into word tokens (tokenised), then for each word, the IDs of the rows in the database that contain that particular term are stored alongside that particular term. This means that a

row's ID can immediately be accessed when looking up a certain word. (King, 1974)

It also provided full support for storing high-dimensional vector embeddings, which are required inside this system to help perform dense passage retrieval without needing to encode the embeddings each time the system is run.

Two different variations of document stores were used, ultimately Elasticsearch proved to be the most efficient. An older version of the system used SQLite with an FTS5 virtual table to perform full-text search on the document title and text fields rather than Elasticsearch. Table 1 shows the performance of system Version 1.3, containing a locally hosted SQLite3 database with FTS5 compared with system Version 1.4, containing the exact same retrieval process (only including document retrieval), however with a locally hosted Elasticsearch database being used instead of an SQLite database. We can see that, when all else is kept equal, the average execution time for the SQLite3 version is more than triple that of the Elasticsearch version while bearing similar retrieval metrics (for more information on how this test data was created, visit 4.2.1.1)

Metric	**Version 1.3 (SQLite3 + FTS5)**	**Version 1.4 (Elasticsearch)**
Document Recall	74.90%	73.66%
Document Precision	10.00%	12.14%
Document F1	17%	21%
OFEVER Document Score	77.40%	74.01%
Average Execution Time	64.70s	18.27s

Table 1: Performance of system Version 1.3 (SQLite + FTS5) compared to Version 1.4 (Elasticsearch).

3.3.5 Split Pipelines for Document Retrieval

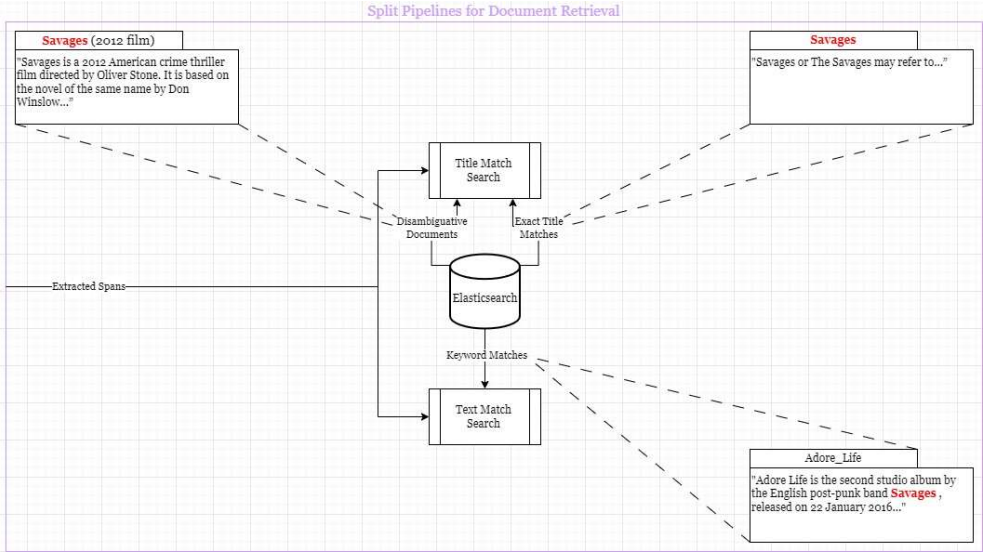


Figure 17: Split Pipeline for Document Retrieval Diagram.

The reason as to why the Title Matching pipeline and Text Matching pipeline were both included in the document retriever module was to help make use of computationally inexpensive methods of retrieval.

The implementation made by Nie et al. was able to achieve an 88.86% recall for their keyword matching system, which the Title Search section of this system aims to mimic. The stated reason that their solution uses a title matching system is to reduce the search space to make the task of performing semantic similarity computationally tractable (Nie, et al., 2018), which is a goal also shared by our system. One issue with their system is that the keyword matching system for titles is semantics-agnostic, therefore if an exact match of a keyword is not found in a document title, then the relevant document cannot be retrieved. We attempt to somewhat mitigate this issue by creating an extra pipeline that is designed to slightly widen the search space to include not just documents where the keywords found in the claim are in the title, but also documents where matching keywords are found inside the document.

It was discovered in a past implementation of this document retrieval system that trying to process the vector representations of all 5.45 million document texts is unfeasible. This past implementation encoded 5 of the 109 JSONL files

using the model `paraphrase-MiniLM-L6-v2`, totalling around 150,000 documents, and stored them using a FAISS index. `paraphrase-MiniLM-L6-v2` is regarded as a relatively lightweight model with an encoding speed of 19,000 sentences per second on a V100 GPU, whilst achieving a sentence embedding score of 62.29 and a semantic search score of 39.19 (Sentence-Transformers, 2024). However, with a single claim running on the system specifications listed in 3.1.1, this system was not able to retrieve a single claim without running out of memory and crashing the application. After running the system on a Google Colab instance with a V100 GPU and 50GB RAM it was discovered that a single claim would take an estimated 106 hours to retrieve (see Figure 18).

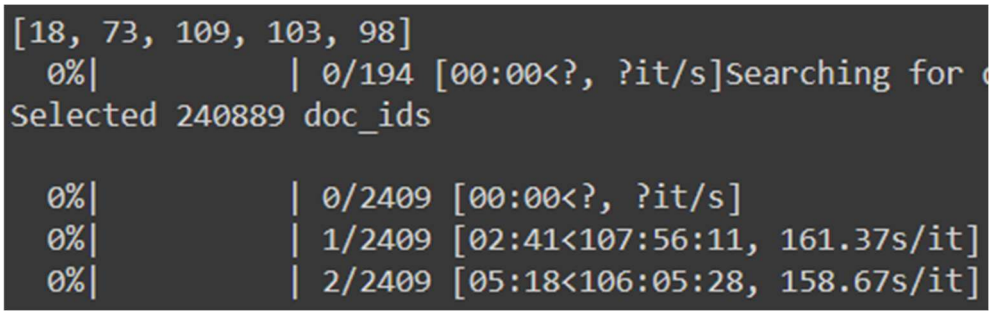


Figure 18: Screenshot of singular claim running on Google Colab on previous system.

3.3.6 Named Entity Recognition + Answer Extraction

Basic Design

The NER system in this version used Hugging Face's pipeline function to automatically load the default version of the `WikiNEuRal-multilingual-NER` model from the Hugging Face hub. Additionally, this system also uses an answer extraction model (`t5-small-answer-extraction-en`) to extract extra relevant spans to search for as well as correct for named entities not being correctly extracted. After this the input claim for the user is passed in and spans from the claim are returned.

Justification

The implementation made by Nie et al. used a keyword matching system where they would search for document titles that match spans of text from the claim. These spans of text were identified by a set of grammatical rules. Instead of opting for this approach, where it can be complicated and time-consuming to

create this set of rules as well as specifically tailored towards the FEVER dataset, a NER system was used to help identify these spans in a more straightforward approach.

NER was chosen as a suitable method for identifying these spans as the FEVER dataset is comprised of Wikipedia articles, which are almost all information regarding a named entity by design. In addition, almost all factual claims provided in the FEVER dataset contain named entities.

The WikiNEuRal model was initially chosen as it performed NER, and was specifically fine-tuned on the WikiNEuRal dataset, which was generated from Wikipedia articles (Tedeschi, et al., 2021). The assumption was made that since the FEVER dataset contains entirely Wikipedia articles, an NER model trained on a dataset with a similar structure and content from the same source would be more suited to the task.

The answer extraction model was also added to the function as it helped catch additional named entities that were not found using the NER model. It is a T5 model, trained to extract answers from marked sentences within a given context on the SQuAD 2 dataset. Whilst it is not specifically trained for NER, it generally tends to extract more accurate names for named movies and television show names, as often the WikiNEuRal NER struggles to correctly identify a television or show name beginning with an article. An example is given below in Figure 19 and Figure 20 where the correct entity is "The Kerner Entertainment Company":

Figure 19: Example output of `WikiNEuRal-multilingual-NER` using claim from FEVER dataset.

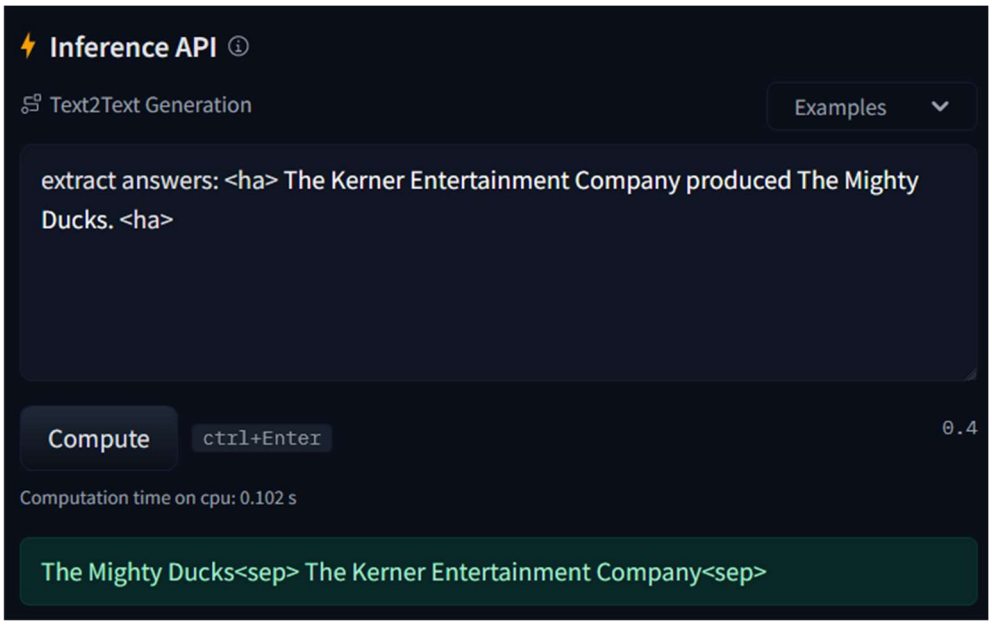


Figure 20: Example output of `t5-small-answer-extraction-en` using claim from FEVER dataset.

Several other different models and systems were tested for the extraction of spans. Namely using spaCy's `en_core_web_sm` model alongside its inbuilt entity extraction module, `bert_base_ner`, as well as just running `WikiNEuRal-multilingual-NER` and `t5-small-answer-extraction-en` on their own, however these all achieved comparatively worse results.

Implementation

We first initialise the models within an `EvidenceRetriever` class as attribute using the Transformers pipeline, automatically downloading the model from the Hugging Face hub. The output from both the NER and answer extraction model is then generated and reformatted, then concatenated. After this process duplicate values are removed, as well as useless values such as "#" if present in the list. The code can be seen in Figure 21.

```
self.NER_model = pipeline("token-classification",
model="Babelscape/wikineural-multilingual-ner",
grouped_entities=True)
self.answer_extraction_pipe = pipeline("text2text-
generation", model="vabatista/t5-small-answer-extraction-en")
```

```
def extract_entities(answer_pipe, NER_pipe, text):
    # Extract entities from text through answer pipeline
```

```
    input = "extract entities: <ha> " + text + " <ha>"
    output = answer_pipe(input)

    entities = []
    answers = output[0]['generated_text'].split("<sep>")
    entities.append(answers)

    # Extract entities from text through NER
    NER_results = NER_pipe(text)
    entities.append(NER_results)

    # Remove duplicates
    entities = list(set(entities))

    return entities
```

Figure 21: NER model loading and NER function code implementation.

3.3.7 Text Matching Pipeline

Implementation

The text matching search function receives all entity spans detected alongside a variable document limit (default set to 100, however final versions use a limit of 1000) and an Elasticsearch instance. A new query is created containing a `should` clause. A `should` clause in this scenario returns true if one or more conditions inside the clause are met, essentially acting as a logical OR. Inside the `should` clause is a `match_phrase` condition which simply searches for exact text matches of a specified phrase, in this case each entity span. The results are automatically ordered using BM25 ranking, having the effect ordering the documents where the entity span is most relevant to the document text first. The code for this query is shown in Figure 22.

```
def text_match_search(entities, es, limit=100):
    # Retrieve documents from db containing query
    query_body = {
        "query": {
            "bool": {
                "should": [
                    {"match_phrase": {"content": entity}}
                    for entity in entities
                ],
```

```
                "minimum_should_match": 1
            }
        },
        "size": limit
    }

    response = es.search(index="documents", body=query_body)
```

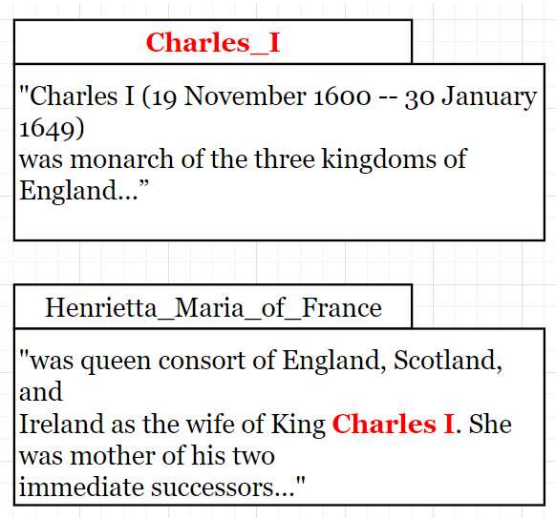
Figure 22: Code showing Elasticsearch query for Text Matching pipeline.

After this the N documents are passed into the generated question ranking system alongside the disambiguative documents.

Justification

As mentioned earlier, the reason the text-matching pipeline was included was to help retrieve documents that contained evidence for claims that do not have matching entity spans in the document title. An example of this is one of the claims inside the test subset below:

*Claim: "**Charles I**'s wife gave birth to his two immediate successors."*



In this example, `Henrietta_Maria_of_France` can never be found by using a purely title-matching approach, however it can be found using our Text Matching pipeline as its document text contains an entity span that matches one in the claim.

It is also worth noting that, this step was created with the intent of narrowing the search space as much as possible without filtering out relevant documents. Ideally the question ranking stage would simply be applied to all documents, however in previous versions, similarity search for the whole corpus was found to be intractable (see Figure 18) therefore this filter had to be applied.

Though the percentage of correct documents retrieved using the Text Matching pipeline tends to vary among the systems implemented, it always contributes a substantial amount towards the final set of correct documents retrieved, showing the pipeline is indeed worth including. This can be observed in Table 2.

Metric	**Title Match (as % of hits)**	**Disambiguation (as % of hits)**	**Text Match (as % of hits)**
Version 1.0	88.64%	5.11%	6.25%
Version 1.1	61.66%	5.13%	33.20%
Version 1.2	58.00%	4.46%	37.55%
Version 1.3	88.46%	4.95%	6.59%
Version 1.4	89.94%	5.03%	6.59%
Version 1.5	85.64%	9.57%	4.79%
Version 1.6	85.71%	9.82%	4.46%
Version 1.7	80.84%	8.14%	11.02%
Final Version	85.64%	9.57%	4.79%

Table 2: Comparison of percentage of documents retrieved by different retrieval methods.

3.3.8 Question-Based Ranking System

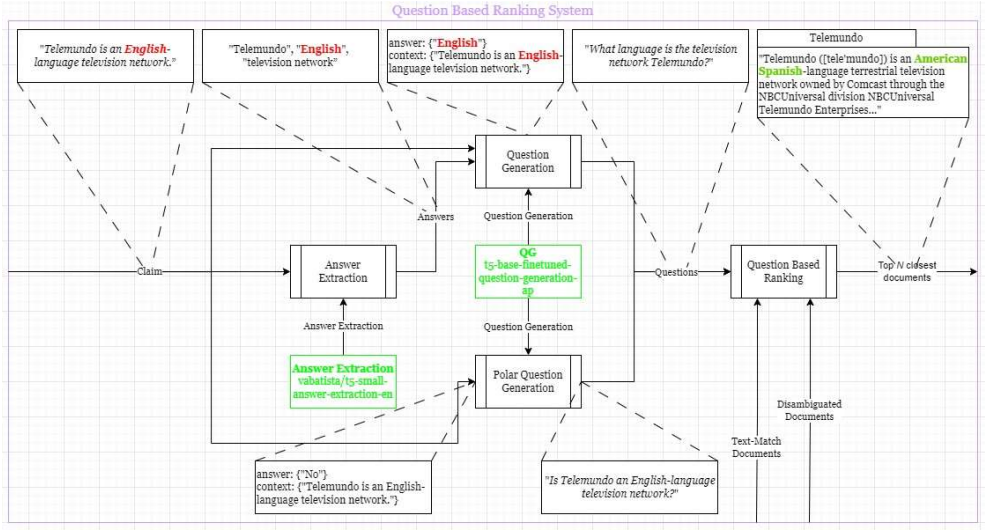


Figure 23: Diagram of Question Based Ranking System.

Implementation

The question generation system relies on answer-aware question generation to help generate a set of questions that ask for all key information needed to validate the claim. It takes extracted spans generated by passing the input claim through the `t5-small-answer-extraction-en` model to try and extract key information that could be used as "answers" for a probing question about the claim. This can include named entities, dates, and locations. This model was generated using the SQuAD 2 dataset which contained an answer, question, and context field, therefore any classes of spans classified as answers in the SQuAD 2 dataset can be classed as potential "answers". The implementation of this can be seen below, first the model is initialised from the Hugging Face hub, and the claim is formatted and entered into the model (see Figure 24).

```
self.answer_extraction_pipe = pipeline("text2text-
generation", model="vabatista/t5-small-answer-extraction-en")
```

```
# Generate questions for each answer in the query
claim_answers = extract_answers(self.answer_extraction_pipe,
claim)
```

```
def extract_answers(pipe, context):
    input = "extract answers: <ha> " + context + " <ha>"
    output = pipe(input)

    focals = []
    answers = output[0]['generated_text'].split("<sep>")
    for answer in answers:
        if answer != "":
            focals.append({'focal': answer, 'type':
"ANSWER"})
    return focals
```

Figure 24: Code showing answer extraction process.

A question is generated around each answer span using the `t5-base-finetuned-question-generation-ap` model. This model is an answer-aware question generation model, meaning that it requires both an answer and a context to generate a question as opposed to just a context. In this case the claim is treated as the context for the question and the extracted answer span is treated as the answer.

$$input = \text{“}answer: \{\boldsymbol{answer\_span}\}\ context: \{\boldsymbol{claim}\}\text{”}$$

The implementation can be seen in the code below in Figure 25:

```
self.question_generation_pipe = pipeline("text2text-
generation", model="mrm8488/t5-base-finetuned-question-
generation-ap", max_length=256)
```

```
for answer in claim_answers:
   question =
extract_questions(self.question_generation_pipe,
answer['focal'], claim)
   self.questions.append(question)
   print("Question for answer '" + answer['focal'] + "':",
question)

# Manually generate polar questions (yes/no questions)
polar_questions = extract_polar_questions(self.nlp,
self.question_generation_pipe, claim)
for polar_question in polar_questions:
   self.questions.append(polar_question)
   print("Polar question:", polar_question)
```

```
def extract_questions(nlp, focal_point, claim):
```

```
    question_generation_string = "answer: " + focal_point + "
context: " + claim
    question_generation_output =
nlp(question_generation_string)
    question =
question_generation_output[0]['generated_text'].replace("ques
tion: ", "")
    return question
```

Figure 25: Code implementation of question generation using extracted answer spans.

An extra "polar" question is then generated around the claim. A polar question is a yes/no question created by rephrasing the claim. An example can be seen below:

Claim: "Telemundo is an English-language television network"

Polar question: "Is Telemundo an English language television network?"

This is achieved by using dependency parsing. The sentence is pre-processed using the spaCy model `en_core_web_sm` so that it contains various grammatical annotations such as dependency tags. The claim is then split into sentences (normally not a required step as all claims in the FEVER dataset are only one sentence). In each sentence, if the sentence contains a root verb and the root verb is an auxiliary verb, then the auxiliary verb is removed and added to the start of the sentence. For example, consider the following claim:

*Claim: "Telemundo **is** an English-language television network."*

*Root Verb: "**is**"*

*Polar Question: "**Is** Telemundo an English-language television network?"*

If the sentence contains a root verb and the root verb contains an auxiliary verb as a child, then the auxiliary verb is removed and added to the start of the sentence. For example, consider the following claim:

*Claim: "Telemundo **has** **been** established as an English-language television network."*

*Root Verb: "**been**"*

*Auxiliary Child: "**has**"*

*Polar Question: "**Has** Telemundo **been** established as an English-language television network?"*

If the sentence does not contain a root verb, then the claim is treated as context for the question generation model and the answer is set to the word "No".

*input = "answer: {**'No'**} context: {**claim**}"*

The implementation for this can be seen in Figure 26 below:

```
def extract_polar_questions(nlp, pipe, claim):
  doc = nlp(claim)
  questions = []

  for sentence in doc.sents:
    altered = False
    for token in sentence:
      if token.dep_ == "ROOT":
        if token.pos_ == "AUX":
          # remove the auxiliary verb and add to the
beginning of the sentence
          question = sentence.text.replace(token.text, "")
          question = token.text + " " + question
          altered = True
        elif token.pos_ == "VERB":
          # if there is an auxiliary verb, remove it and add
to the beginning of the sentence
          aux = [child for child in token.children if
child.dep_ == "aux"]
          if aux:
            question = sentence.text.replace(aux[0].text, "")
            question = aux[0].text + " " + question
            altered = True
   if not altered:
     input_string = "answer: " + "No" + " context: " + claim
     output = pipe(input_string)
     question =
output[0]['generated_text'].replace("question: ", "")
```

Figure 26: Code implementation of polar question generation.

The DPR context vector embeddings for the disambiguative documents and documents retrieved using the Text Matching pipeline are retrieved from the Elasticsearch database. A Haystack `DensePassageRetriever` is initialised using the retrieved embeddings, at which point each question is then encoded using the DPR model `dpr-question_encoder-single-nq-base`, representing the question in a high-dimensional vector format. The closest ten context vector representations to the question vector representation (according to dot product similarity) within the vector space are retrieved and their dot product scores are used as overall document scores. These documents are then passed into the passage retriever.

The same generated questions are also used in the passage retriever to help rank passages found inside the document text, getting the top 30 passages, then filtering out passages that are below a dot product score of 0.7.

Justification for using Question Ranking

The idea behind using a set of questions to try and rank the documents is that each factual claim can be checked by fully answering a set of questions centred around key pieces of information in the claim. For example, consider the following claim:

Claim: "Telemundo is an English-language television network"

Theoretically, the truthfulness of this claim can be assessed by answering the following questions:

Questions: ["What language is Telemundo?", "Is Telemundo a television network?"]

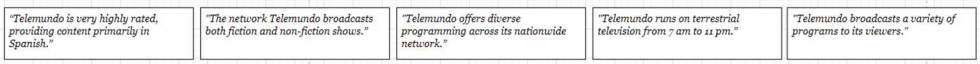
If an evidentiary document contains information that confirms that Telemundo is English-language, **and** a document contains information that confirms that Telemundo is a television network, then the claim is fully supported. Therefore, documents that are seen to contain the answers to these questions should be sought after in the evidence retrieval process.

This system was thought to provide an advantage over simply just searching for documents with the highest semantic similarity to the claim. This is because there is an assumption that the relevant evidence documents or passages are semantically similar to the claim itself, which is not always the case. Retrieving passages or documents that answer specific questions might be more effective for a range of reasons.

When searching for documents that are semantically similar to a singular claim passage, the ranking of those documents in terms of similarity score might not be an accurate way of ranking them in terms of usefulness to answering a claim. This is because a claim could consist of multiple pieces of information needing to be retrieved, and certain information may have a significantly larger amount of semantically similar information available, meaning that there is too much information being displayed for certain elements of the claim, but less for other, equally important areas of the claim. For example, take the following claim and passages:

Claim: "Telemundo is an English-language television network"

Passages:



To fully check the claim, it needs to be ascertained what language Telemundo is spoken in, and if Telemundo is a television network. Using the cosine similarity of the vector representations (using the `sentence-transformers/all-mpnet-base-v2` model) of each passage compared to the claim, we get the ranking shown in Figure 27.

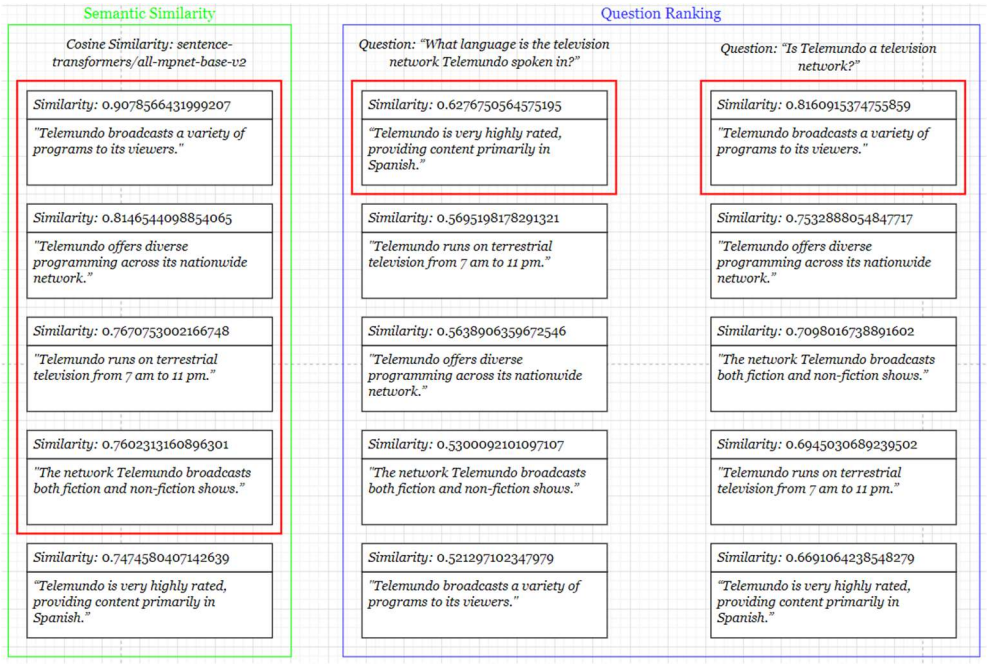


Figure 27: Comparison between ranking by semantic similarity and question ranking.

If we assume the cutoff point is four passages, then we essentially get four passages conveying the information that Telemundo is a television network, and no passages conveying what language Telemundo is spoken in. If we generate questions around key pieces of information inside the claim, using the similarity between the claim and the questions we observe better results.

If we take the passage cutoff point at one document per question, then we get one document that states the language of Telemundo, and one document conveying the information that Telemundo is a television network. Therefore, by retrieving information that answers specific questions about the claim, the final structure of the evidence might be more effective for a fact-checking scenario, retrieving relevant information that may be overlooked in a situation where questions are not generated.

We did observe positive effects when using question generation to help rank documents. In Version 1.0 of our implementation, questions were not generated around each claim, so instead the semantic similarity score between the claim and document texts were used to rank documents and the top ten most similar documents were taken. In Version 1.1, we first implemented our question generation system as a ranking system after the top ten most similar documents

were retrieved. A comparison between Version 1.0 and 1.1 in Table 3 between systems shows that the average position of the relevant document fell in Version 1.1, meaning that the relevant document was higher up in the ranking after the question generation ranking system was added.

Metric	**Version 1.0**	**Version 1.1**
OFEVER Document Score	74.01%	75.14%
Average Execution Time	28.30s	39.04s
Average Position of Correct Document	10.00	7.02

Table 3: Comparison between Version 1.0 and 1.1 showing the effectiveness of ranking documents by how well they answer probing questions.

Justification for Models Used

The `t5-base-finetuned-question-generation-ap` model was used for question generation and the `t5-small-answer-extraction-en` model was used for extracting answers.

The `t5-base-finetuned-question-generation-ap` model was observed to produce higher quality questions than the `t5-base-question-generator` model, which was the other model that we experimented with. The `t5-base-question-generator` model made by fine-tuning the T5 base model with QA datasets such as SQuAD (see 2.5.2), CoQA and MSMARCO to generate questions, however experiences some issues when confronted with short contexts (iarfmoose, 2024), sometimes generating incoherent text. This proved to be problematic as input claims were used for our contexts, which were only ever one short sentence long. The `t5-base-finetuned-question-generation-ap` model was fine-tuned using only the SQuAD dataset and does not experience this same effect when using short sentences. This effect can be seen when using a claim from the FEVER dataset to generate a question using the `t5-base-question-generator` model (see Figure 28) and when using the `t5-base-finetuned-question-generation-ap` model (see Figure 29).

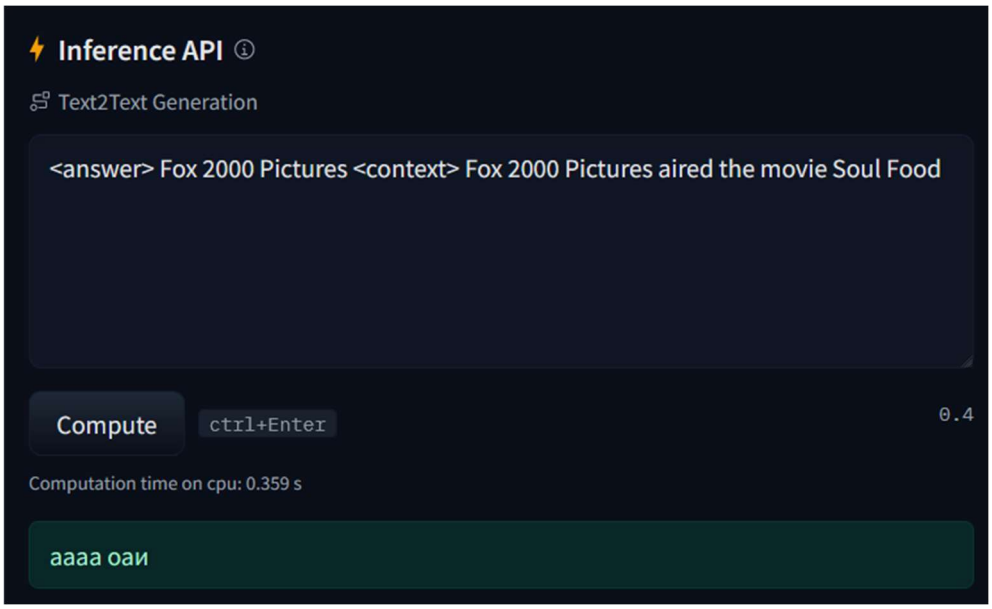


Figure 28: Example output of `t5-base-question-generator` using claim from FEVER dataset.

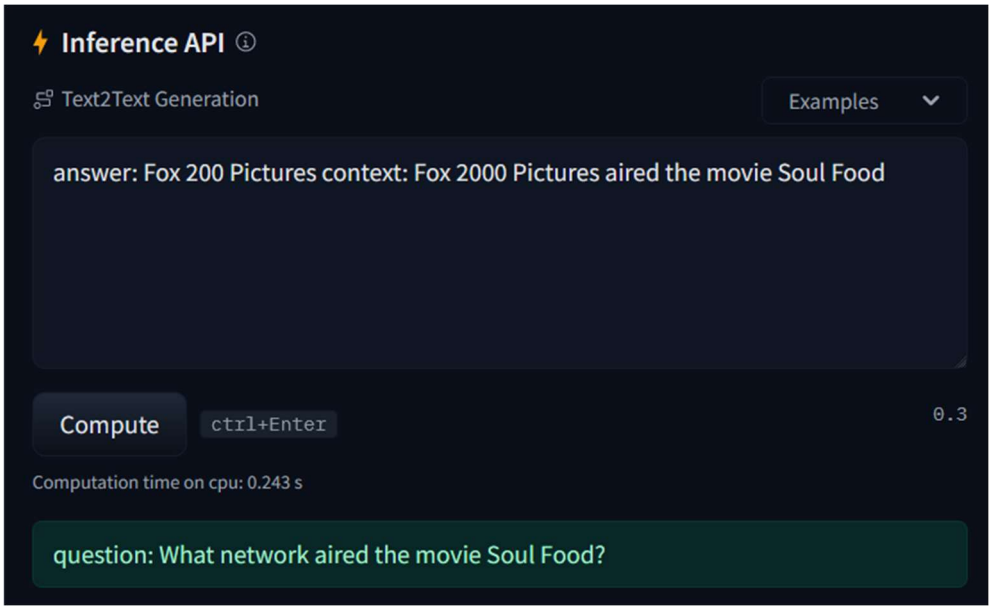


Figure 29: Example output of `t5-base-finetuned-question-generation-ap` using claim from FEVER dataset.

Justification for Polar Questions

Polar questions were appended to the question set for ranking to produce questions for claims that only had one piece of key information extracted. For example, consider the claim from the FEVER dataset:

Claim: "There is a capital called Mogadishu."

The only key text that can be extracted from the claim would be the word "Mogadishu", however using the word "Mogadishu" as the answer to the question returns the following:

Answer: "Mogadishu"
Context: "There is a capital called Mogadishu."
Question: "What is the capital of the city?"

This question is not particularly useful when retrieving documents as there are many different documents, not related to Mogadishu, that provide adequate answers to the question. Instead, we generate a "polar question":

Answer: "No"
Context: "There is a capital called Mogadishu."
Question: "Is there a capital called Mogadishu?"

This question is more useful as it directly targets documents relating to Mogadishu. Polar questions are less successful at retrieving relevant documents; however, they are more effective than not including them at all in the retrieval process, therefore they are kept in the final implementation (see Table 4).

Metric	**Final Version (with polar question)**	**Final Version (without polar question)**
Document Recall	77.36%	76.54%
Document Precision	6.24%	8.23%
Document F1	11.55%	14.87%
OFEVER Document Score	79.66%	78.53%
Average Execution Time	31.75s	31.54s

Table 4: Comparison of Final Version with polar question and without polar question.

3.3.9 Dense Passage Retrieval

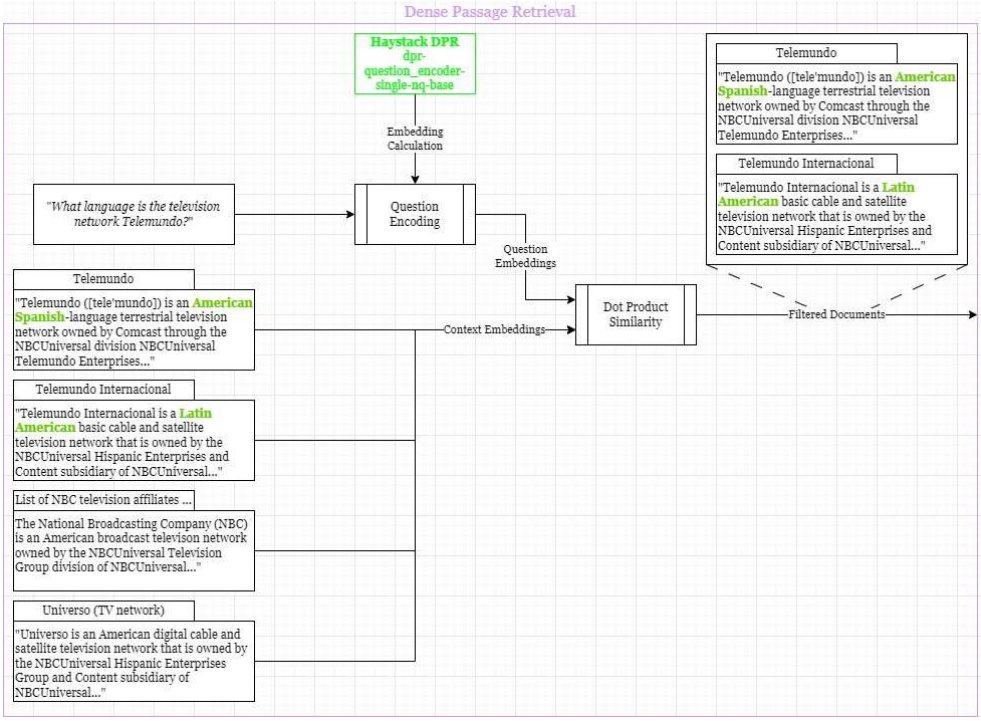


Figure 30: Dense Passage Retrieval Diagram.

Implementation

A Dense Passage Retriever is used inside the document retrieval module to retrieve a set of documents that answer each question. The DPR takes in a question vector and a set of context vectors, it then performs dot product comparison between each set of context vectors with the question vector to find the closest contexts. In our scenario, two different specialised models are used to convert questions and contexts into high-dimensional vector format.

To prepare the document text to work with the DPR system, the Elasticsearch database needed to be loaded with the document context vectors before using the DPR. To achieve this, the Elasticsearch database needed to be initialised with an empty `embeddings` field. The Haystack framework was used in this scenario to get each document inside the Elasticsearch database, converting it to an `ElasticsearchDocumentStore` object so that it could be used by Haystack's `DensePassageRetriever` class. A `DensePassageRetriever` object

was then initialised with the `dpr-question_encoder-single-nq-base` to encode questions into vectors and the `dpr-ctx_encoder-single-nq-base` to encode document texts. After this, all 5.45 million document texts in the Elasticsearch database were converted into vector representation using `dpr-ctx_encoder-single-nq-base`. As this particular stage could not be run on the system specification listed in 3.1.1, a Google Colab instance needed to be used with a V100 GPU. The process of loading the vectors into the Elasticsearch took a total of 39 hours and the embeddings take up a total of 80GB.

```
# Connect to Elasticsearch
document_store = ElasticsearchDocumentStore(
    host=os.environ.get("ES_HOST_URL"),
    port=os.environ.get("ES_PORT"),
    scheme=os.environ.get("ES_SCHEME"),
    username=os.environ.get("ES_USER"),
    password=os.environ.get("ES_PASS"),
    index="documents",
    embedding_field="embedding",
    embedding_dim=768,
)
```

```
# Load Dense Passage Retriever
retriever = DensePassageRetriever(
    document_store=document_store,
    query_embedding_model="facebook/dpr-question_encoder-
single-nq-base",
    passage_embedding_model="facebook/dpr-ctx_encoder-single-
nq-base",
    use_gpu=False,
    embed_title=True,
    batch_size=2,
)
```

```
# Load the document store
document_store.update_embeddings(
    retriever=retriever,
    index="documents",
    update_existing_embeddings=False
)
```

Figure 31: Code implementation of loading context embeddings for DPR into Elasticsearch database.

After these embeddings were loaded, the DPR document retrieval system could then be used. After this the disambiguative documents and the documents retrieved using the Text-Matching pipeline then had their context embeddings retrieved from the Elasticsearch database. At this stage there is a limit of around 1000-3000 documents, although normally only around 20 documents get passed into the DPR. Each question is then iterated through and encoded into a high-dimensional vector using the `dpr-question_encoder-single-nq-base` using Haystack's `DensePassageRetriever` class, which then calculates the dot product score between the documents and each question. For each document the highest dot product score amongst each question is kept as its final document score, and a final filter is applied, removing any documents that contain below a 0.65 dot product score. This process can be seen in Figure 32.

```
# For doc in both disambiguated and textually matched docs,
add to doc store
doc_store = listdict_to_docstore(disambiguated_docs +
textually_matched_docs)

# Initialise retriever
print("Initialising DPR")
retriever = DensePassageRetriever(
  document_store=doc_store,
  query_embedding_model="facebook/dpr-question_encoder-
single-nq-base",
  passage_embedding_model="facebook/dpr-ctx_encoder-single-
nq-base",
  use_gpu=False,
  embed_title=True,
  batch_size=2,
)
```

```
# Retrieve docs for each question keeping the highest scoring
docs
print("Retrieving documents for each question")
for question in tqdm(self.questions):
  results = retriever.retrieve(query=question)
  for result in results:
    id = result.id
    score = result.score
```

Figure 32: Code implementation of performing Dense Passage Retrieval in document retrieval module.

Justification

The decision was made to use Dense Passage Retrieval as it was particularly well suited towards assigning scores for the answerability of text based on a set of questions, it also maintained a good speed to performance balance. As the system was based around assigning answerability scores (essentially performing extractive QA on our document corpus) there were multiple different options, however Meta's Dense Passage Retriever consistently outperforms other retrieval methods such as BM25, GraphRetriever, PathRetriever and ORQA on QA datasets such as NQ, TriviaQA, WQ, and TREC, therefore it was considered the best option for this task. Once built, the DPR also boasts a higher runtime efficiency than other retrieval methods such as Lucene (Karpukhin, et al., 2020), which is also crucial as our evidence retriever must work on the limited specifications listed in 3.1.1.

The reason Haystack was used as the DPR framework was due to its integration with Elasticsearch. Hugging Face's DPR framework was also considered. Before implementing the DPR, the decision to use Elasticsearch as a data store was already established. Haystack's DPR works by using one of many `DocumentStore` objects, and the Haystack framework includes an easy conversion from an Elasticsearch database with embeddings into an `ElasticsearchDocumentStore`, which acts as a `DocumentStore` object for Haystack's DPR, making it relatively simple to implement.

The reason the dual encoder pair of models `dpr-question_encoder-single-nq-base` and `dpr-ctx_encoder-single-nq-base` were used in our implementation was because they were specifically designed with QA tasks in mind. The `dpr-question_encoder-single-nq-base` and `dpr-ctx_encoder-single-nq-base` have parameters that are fine-tuned in such a way that it represents question vectors in a similar vector space to context vectors, making the dot product higher for relevant question answer pairs.

The context embeddings were pre-encoded so that they did not have to be encoded at runtime. As there tends to be around 20 document texts, encoding these documents at runtime takes a significant amount of time compared to simply retrieving them.

3.3.10 Relevancy Classification Model

Implementation

The passage retrieval module contained two completely different approaches, with the final system opting to use a fine-tuned DistilBERT model that classifies whether a sentence is relevant to a claim. This model was trained using data specifically from the FEVER dataset.

First each document text retrieved using the passage retriever was split into sentences using spaCy's sentence splitting function. To do this document text was encoded using `en_core_web_sm`, then split using the `.sents` method. Each sentence was then formatted to work with the DistilBERT model by adding a separation token and concatenating it with the claim. The formatted text is then tokenised with the DistilBERT tokeniser and then passed into the relevancy classification model as shown in Figure 33.

```
doc = self.nlp(evidence_text)

evidence_sentences = []
for sentence in doc.sents:
  sentence = sentence.text
  input_pair = f"{claim} [SEP] {sentence}"
  result =
self.relevance_classification_tokenizer_pipe(input_pair)
```

Figure 33: Code showing the formatting of document texts into passages to be classified for relevancy.

The model outputs either a label of "`LABEL_1`" if the sentence is relevant to a claim or a label of "`LABEL_0`" if a sentence is not relevant to the claim, alongside its confidence score. The sentence is only kept if it gets labelled with a "`LABEL_1`".

Training

To train the relevancy classification model a random selection of 10000 claims from the FEVER dataset alongside their evidence documents were retrieved. The claims contained one or more evidence sentences from those evidence documents. To help the model understand different types of irrelevant documents, some sentences from completely random documents were retrieved as well as some irrelevant sentences from the same document.

Let:

$$r = Relevant\ sentence$$
$$i_s = Irrelevant\ sentence\ from\ same\ document$$
$$i_d = Irrelevant\ setentence\ from\ different\ document$$

The optimal ratio found was:

$$1r + 2i_s + 2i_d$$

A single sample of this dataset can be seen in Figure 34:


```
{
    "claim": "Chris Hemsworth appeared in A Perfect Getaway.",
    "sentences": [
        {
            "doc_id": "Chris_Hemsworth",
            "sent_id": 2,
            "sentence": "Hemsworth has also appeared in the science fiction action film Star Trek -LRB- 2009 -RRB- , the thriller
            "label": 1
        },
        {
            "doc_id": "Chris_Hemsworth",
            "sent_id": 5,
            "sentence": "In 2015 , he starred in the action thriller film Blackhat , had a comedic role in the fifth installment
            "label": 0
        },
        {
            "doc_id": "Chris_Hemsworth",
            "sent_id": 7,
            "sentence": "Hemsworth will reprise his role as George Kirk in the upcoming Star Trek sequel .\tStar Trek\tStar Trek
            "label": 0
        },
        {
            "doc_id": "Guanzhuang",
            "sent_id": 0,
            "sentence": "0\tGuanzhuang is the name of several places in the People 's Republic of China :",
            "label": 0
        },
        {
            "doc_id": "Racah_seniority_number",
            "sent_id": 1,
            "sentence": "The `` seniority number '' , in a loosing statement , is quantum number additional to the total angular
            "label": 0
        }
    ]
},
```


Figure 34: Sample row of generated dataset for relevancy classification model.

After the 10000 training samples were generated, the data was split into a training, validation, and test set in the ratio 8:1:1. Therefore, the training set contained 8000 samples, the validation set contained 1000 samples and the test set contained 1000 samples. The model was fine-tuned using Hugging Face's `Trainer()` class, initialised with the optimal hyperparameters shown in Table 5.

Hyperparameters	Value
Epochs	3
Batch Size	24
Warmup Steps	500
Weight Decay	0.01
Learning Rate	1e-5
Train Test Split	8:1:1
Dataset Size	10000 samples
Base Model	`distilbert/distilbert-base-uncased`

Table 5: Fine-Tuning Hyperparameters.

As the training could not be performed on a laptop, a Google Colab instance was initialised with a Tesla V100 GPU and 50GB of RAM. Overall, the training process took around 1 hour, however, began to overfit in the third epoch. The loss metrics can be seen in Figure 35.

Epoch	Training Loss	Validation Loss
1	0.005200	0.210088
2	0.000600	0.226910
3	0.476300	0.231556

Figure 35: Loss metrics for training of relevancy classification model.

Justification

Several decisions had to be made regarding the size and content of the training data. The reason why we chose to include irrelevant sentences from the same document as well as irrelevant sentences from random documents in the training dataset was because the model needed to be able to receive sentences that come from the same document as well as documents that may talk about a completely unrelated topic. Sentences from the same document enable the model to make fine distinctions as to which sentences are relevant to the claim, and the inclusion of irrelevant sentences from random documents helps to emulate the format that

the model would receive data in. A 1:2:2 split was used as a ratio for this data as it ended up returning a model with the highest accuracy, although 1:1:1, 1:3:3 and 1:1:2 splits were also used.

In the end, a dataset size of 10,000 was the most effective at training the model as datasets of a smaller size (sets of 1000 and 5000 claims were tested) did not provide the model with enough data to learn if a sentence was relevant to a claim. Datasets that were larger (a set of 20,000 claims was tested) did not provide a significant increase in performance over the 10,000-claim set but did double the training time.

A DistilBERT model was chosen for this task primarily due to its high speed compared to a traditional BERT model or a RoBERTa model, which was crucial for this project to run in a useful timeframe with the hardware limitations, and its ability to perform at a similar effectiveness to a traditional BERT model.

Different combinations of hyperparameters were used to train this model, as initially the model was overfitting. Eventually it was found that reducing the learning rate from $1e-4$ to $1e-5$ reduced the overfitting significantly over the 3 epochs.

We also chose to use data directly from the FEVER dataset as opposed to choosing data from another dataset as this meant that the model was accustomed to the specific claim and sentence format in our dataset. Our dataset contains short factual claims that need to be matched with varying length sentences, therefore it seemed important for our model to be fine-tuned with this structure in mind. It was also assumed that the model might be able to pick out unseen nuanced properties within the dataset that may not be immediately obvious.

It is also worth mentioning that other methods were used in different versions for passage retrieval that were found to be less effective. For example, in Version 1.6 and 1.7 we used a dual-pipeline approach, containing one pipeline that was focused on lexical similarity, splitting document texts into sentences and ranking using BM25 scores, and another pipeline that used a QA reader model to retrieve passages that contained answers to the questions generated in the document retrieval step. However, this approach had a significantly longer runtime as well as worse recall and a more-or-less identical precision, therefore it was substituted entirely for the relevancy classification model (see Table 6).

Metric	Final Version (relevancy classification)	Version 1.7 (dual pipeline)
Passage Recall	60.81%	47.90%
Passage Precision	60.28%	60.86%
Passage F1	60.56%	53.61%
OFEVER Passage Score	59.86%	52.27%
Average Execution Time	31.75s	36.24s

Table 6: Comparison between relevancy classification and dual pipeline approaches to passage retrieval.

3.3.11 Final Ranking Using Semantic Similarity

The final step of the passage retrieval was to rank the remaining passages. Results from the relevancy classification model produced rankings that were often extremely close together, with most passages retrieved having scores that would only vary between the range $0.9991 - 0.9999$, and often the order did not necessarily put the passages most relevant to the claim first. Therefore, a simple semantic similarity calculation was performed between the input claim and each of the remaining passages. To perform this, Sentence Transformer's `all-mpnet-base-v2` was used to encode both the claim and the passages. Finally, all passages that contained a semantic similarity score of below 0.7 were removed, and the passages were sorted by this final score. The implementation of this can be seen in Figure 36.

```
# Setup similarity model
self.sim_model = SentenceTransformer('sentence-
transformers/all-mpnet-base-v2')
```

```
def get_semantic_sim(self, claim, sentence):
  embeddings = self.sim_model.encode([claim, sentence])
  return util.cos_sim(embeddings[0], embeddings[1]).item()
```

Figure 36: Code implementation of final semantic similarity ranking.

Justification

The `all-mpnet-base-v2` model was used to rank the similarity of passages to the claim as it achieves the highest average performance out of all the Sentence

Transformers models for sentence embeddings and semantic search. Although it does have a comparatively low speed compared to some of the other models, this was not considered an issue as at this stage of retrieval most of the passages will have been filtered out, leaving a small number of remaining passages. These results can be seen in Table 7.

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

Table 7: Sentence Transformers model comparison.

3.3.12 Web Application

Implementation

The Flask framework, alongside simple HTML, CSS, and JavaScript was used to create a homepage, a loading page, and a results page for the evidence retriever. The homepage was designed to house a singular text box where the user could enter their claim, and some informational sections explaining what the system does and how to use it. As the system can take around 30 seconds to load evidence for a single claim, a loading page was implemented to show the user what stage the retriever was at in the evidence retrieval process. Finally, the results page shows the user the claim, the evidence set, and a final verdict for whether the claim is supported, refuted, or does not contain enough evidence.

The homepage design can be seen in Appendix C. it contains a text form that, on submit, saves the text entered inside the text field to a session variable, then redirects the user to the `/demo` route (shown in Figure 37). This route then retrieves the claim stored in the session variable, it generates a random unique

ID and starts a thread with that ID and the claim. The thread then executes the evidence retrieval process.

```
@app.route("/demo")
def demo():
  claim = session.get('claim', 'Not specified')
  task_id = str(uuid.uuid4())
  session["task_id"] = task_id
  Thread(target=background_task, args=(task_id, claim)).start()
  return render_template("demo.html", claim=claim, task_id=task_id)
```

Figure 37: `/demo` route.

The website is able to print out progress updates from the evidence retriever by setting up a route with the specific ID that returns a JSON object containing progress updates.

```
@app.route("/progress/<task_id>")
def progress(task_id):
    return jsonify(progress_store.get(task_id, None))
```

The evidence retriever runs the `log_progress()` function shown in Figure 38 whenever a significant progress update happens, for instance, if the passage retrieval has started.

```
def log_progress(task_id, log):
  if task_id:
    if task_id not in progress_store:
      progress_store[task_id] = {"status": "in progress",
"log": []}

      progress_store[task_id]["status"] = "in progress"
      progress_store[task_id]["log"].append(log)
```

Figure 38: `log_progress()` function.

The loading page contains a JavaScript script that sends a request to the progress update route every second, the information retrieved from this route is then displayed with some formatting to appear as shown in Appendix D.

After the evidence retrieval process is finished the retrieved evidence documents and passages are inserted into a prompt, which is sent to the `mistral-small-latest` large language model via the Mistral API. The prompt appears as follows:

prompt = "Is the following claim supported, refuted
or not enough evidence based on the evidence listed below?
The evidence is to be taken as completely factual.
Claim: {claim}
Evidence: {evidence_passages}"

The retrieved documents and passages, as well as the text generated by `mistral-small-latest` are packaged into a JSON object which is then sent to progress route alongside a status update telling the website that the retrieval process has finished. These are then formatted and displayed in the results page as seen in Appendix E.

The results page splits the evidence into retrieved evidence documents that contain retrieved passages, and documents that do not contain any passages retrieved. The documents with evidence passages are displayed first in order of the passage score, the documents without evidence passages are displayed afterwards in order of their document score. Each document is split, displaying all the evidence passages it contains, the passage score, and the document score. The document text is hidden by default to conserve space (see Figure 39); however, the user can click a button to expand the document box, showing the full document text. In the document text, if the document contains an evidence passage, that passage is highlighted in the text as shown in Figure 40.

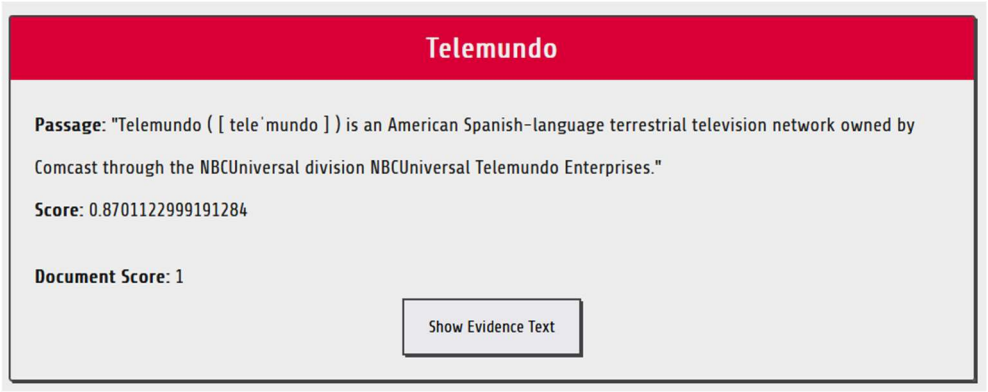


Figure 39: Results page document box with hidden document text.

Figure 40: Results page document box with expanded document text.

Justification

Flask was chosen as the web framework because the actual complexity of the web application was relatively low. Flask is a lightweight, micro web framework for Python. As such the web application was well suited to Flask as the evidence retrieval system was already implemented in Python, and the design of the web application was not complex, only requiring three pages therefore only basic web functionality, such as the setup of routes, was needed.

After constructing the website, it was theorised that adding the extra step of using the retrieved evidence to help make a prediction of the veracity of the claim would not take a long time to implement if a large language model was used. It was thought that it would significantly improve the user benefit of the system as the user could be told explicitly if there was enough evidence to back up their input claim, helping them make more informed decisions and not jumping to the conclusion that evidence either supported or refuted their claim unless verified by the large language model. Additionally, using an LLM to predict the veracity of the claim would make it clearer as to how our evidence retrieval system would work as a step in the automated fact-checking pipeline in the wider context of a full fact-checking system.

The `mistral-small-latest` was used for as it was easily accessible using the Mistral API, as running a large language model locally on the specifications listed in 3.1.1 was unfeasible. The small model was used for the task of classifying whether the claim was supported by the evidence as it was thought to be a relatively simple classification task, therefore not requiring a more complex model. As a result, the runtime speed was also quicker. It is worth mentioning that evidence retrieval, not veracity prediction, was the focus of this project therefore a custom system was not implemented and the options for veracity prediction were not studied in depth.

4 Results and Evaluation

In this section, first the strategy for our evaluation will be explained, detailing how we evaluated each specification, justifying which metrics to use and how they were calculated. Then the whole system will be evaluated according to these metrics. Key elements of the system will also be evaluated using comparative metrics.

4.1 Metrics

To evaluate each of the specification requirements, multiple separate evaluation metrics were chosen:

Recall

Recall is the measure of the fraction of relevant instances retrieved (Powers, 2020). It can be useful to help find out what proportion of relevant instances were actually retrieved, and what proportion of relevant instances were missed.

$$Recall = \frac{Relevant\ retrieved\ instances}{Relevant\ instances\ not\ retrieved}$$

In the context of document retrieval, the recall of the system can be determined as follows:

$$r_d = Relevant\ retrieved\ documents$$
$$n_d = Relevant\ documents\ not\ retrieved$$
$$Document\ Retrieval\ Recall = \frac{r_d}{(r_d + n_d)}$$

In the context of sentence retrieval, the recall of the system can be determined as follows:

$$r_s = Relevant\ retrieved\ sentences$$
$$n_s = Relevant\ sentences\ not\ retrieved$$
$$Document\ Retrieval\ Recall = \frac{r_s}{(r_s + n_s)}$$

Precision

Precision is the measure of relevant instances among all retrieved instances (International Organization for Standardization, 2023). It can be useful to try and determine what proportion of positive retrieved instances were indeed correct and what proportion of retrieved instances were incorrect.

$$Precision = \frac{Relevant\ retrieved\ instances}{All\ retrieved\ instances}$$

In the context of document retrieval, the precision of the system can be determined as follows:

$$r_d = Relevant\ retrieved\ documents$$
$$i_d = Irrelevant\ retrieved\ documents$$
$$Document\ Retrieval\ Precision = \frac{r_d}{(r_d + i_d)}$$

In the context of sentence retrieval, the precision of the system can be determined as follows:

$$r_s = Relevant\ retrieved\ sentences$$
$$i_s = Irrelevant\ retrieved\ sentences$$
$$Document\ Retrieval\ Precision = \frac{r_s}{(r_s + i_s)}$$

F1

The F1 score is a metric calculated from the precision and recall of a test, representing the harmonic mean between those two metrics. As a result, the F1 score is often used as a metric to measure how well a system balances both precision and recall. It can be calculated as follows:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

F1 scores range from 0 to 1, where the lowest possible value is 0 whereas a system with perfect precision and recall achieves a score of 1.

Oracle Experiments

Oracle evaluations are measures designed to analyse the performance of each component in a system by assuming each other component in the system knows the correct answer for every instance in the dataset. An "oracle" is an idealised component that always gives the correct answer.

In the context of an evidence retrieval system, two separate components can be the document retrieval, and the passage retrieval. When calculating the oracle accuracy of the document retrieval, the "oracle" is the passage retrieval component, which would be assumed to have 100% score. Therefore, only the score of the document retrieval component needs to be recorded. If the document retrieval component achieves a score of 90% then the oracle score of the system is said to be 90%.

FEVER and OFEVER

The OFEVER score is short for Oracle-FEVER score and is used to evaluate the effectiveness of an individual module at completing the FEVER shared task. It is done so by assuming each other section of the system is an "oracle". The FEVER shared task requires the submission of an evidence set, that justifies the label of each claim, each set being the minimum set of sentences that fully support the labelling of the claim. Correctly labelled claims that lack the complete evidence set do not receive a correct score. For data that contains multiple evidence sets, only one set is needed to achieve a correct FEVER score (Thorne, et al., 2018). It is worth noting that, for the task presented in this paper, the labelling of evidence is not required so any evidence set will be assumed to be correctly labelled when evaluating the quality of the evidence retrieval system.

Nielsen's Ten Usability Heuristics

The user-friendliness of the website will be assessed using Nielsen's Ten Usability Heuristics (Nielsen, 1994). These are a widely accepted set of heuristics, developed by Jakob Nielsen, that provide a broad set of principles for interface design and are particularly useful for identifying usability issues in web interfaces. They can be found in Appendix F.

4.2 Strategy

A strategy was developed to test each one of the specifications using the set of evaluation metrics mentioned in 4.1 as well as some more commonly used metrics.

Retrieval of Evidence – To evaluate the overall retrieval of evidence, the combined recall, precision, F1, and OFEVER metrics were calculated with both the document and passage retrieval modules working in combination. This means that evidence can only be regarded as correct if the correct passage within the document is found, not just the correct document. A single claim could have multiple evidence document-passage pairings; therefore, a point was given for recall for each one retrieved. When calculating precision, only incorrect document-passage pairings were regarded as false positives. We also evaluate our system in comparison to two other implementations, the FEVER baseline and

the Nie et al. system to benchmark the relative performance of our system against the established baselines and competitors for this task.

Retrieval of Documents – To evaluate how effective the document retrieval module was, we calculated the recall, precision, F1, and OFEVER of the document retriever only. An evidence document was treated as a document that contained an evidence passage for a claim. A single claim could have multiple evidence documents; therefore, a point was given for recall for each one retrieved. Like the Retrieval of Documents specification, we also benchmark our system compared to the FEVER baseline and the Nie et al. system.

Retrieval of Passages - To evaluate how effective the passage retrieval module was, we calculated the recall, precision, F1, and OFEVER of the passage retriever only. This meant that if the passage retriever received an irrelevant document, all results from this document would be discarded as the passage retriever has no way of retrieving the relevant passage from an irrelevant document. Only incorrect document-passage pairings that were within a relevant document to the claim were considered as false positives when calculating precision. Like the Retrieval of Documents specification, we also benchmark our system compared to the FEVER baseline and the Nie et al. system.

Presentation of Evidence – To evaluate the user-friendliness of the website design, the website was evaluated against Nielsen's Ten Usability Heuristics.

Performance of System - The system was run on the device specifications listed in Section 3.1.1, and the average runtime was calculated.

4.2.1.1 Test Dataset Creation

To run experiments with the evaluation metrics, a test dataset needed to be created. The entire FEVER dataset contains 184,445 claims, and a single run of any version of the system on the hardware specifications listed in Section 3.1.1 normally took between 20 and 60 seconds. Assuming the system only took 20 seconds to run, it would take almost 1025 hours to run on all claims. Therefore, the test set focused on claims that contained evidence within the FEVER dataset, so any claims that did not have any/full supporting evidence were removed. A random seed was created and using the seed 200 claim ids were selected from the dataset. These same 200 claims were run through each version of the system

whenever a significant change occurred, and these 200 claims were also used in the final evaluation.

It was thought that 200 claims were a reasonable number as on most versions of the system it took from 1-3 hours to fully run, which was not considered excessive, while maintaining a diverse enough range of claims to accurately represent the entire FEVER population of claims. The claims were randomly chosen for the following reasons:

- Given the subset size was sufficiently large, it helped make sure the claim and evidence passage structure were theoretically representative of the whole dataset. For example, some claims require the retrieval of one passage, however some require the retrieval of over four passages, a random selection ensure that the distribution of claims with one or four passages are the same distribution as found in the original dataset.
- Alternative strategies such as representative sampling is excessively time-consuming and may induce bias. Claims and evidence passages in the FEVER dataset cover a range of topics, vary in size, and possess many other unique properties. Trying to accurately work out the distribution of these properties in the FEVER dataset would be an inefficient use of time, trying to sample these properties could be difficult and time-consuming and may be prone to including sampling bias.

The FEVER dataset does contain a test dataset; however, we chose not to use this dataset as it did not come with claim, evidence pairings. Instead, it just came with a list of claims, requiring a custom module to be imported to mark the claims. Therefore, we decided to create our own marking script. It is important to bear in mind that, when comparing our system to the FEVER baseline and the implementation made by Nie et al., we are not performing a comparison on the same set of data as they were tested using the complete official test dataset, whilst we are only using 200 random samples from the concatenated training and development datasets. That being said, the results obtained from our analysis should proportionately reflect the results that would be obtained if we used the FEVER test dataset. The comparison is being made to help get an idea of how our system performs compared to others, but for an exact comparison a more controlled experiment would need to be made.

4.3 System Results and Evaluation

Metric	Final Version	FEVER Baseline	Nie et al.
Document Recall	77.37%	-	89.23%
Document Precision	6.24%	-	51.04%
Document F1	11.55%	-	64.94%
Passage Recall	60.84%	-	86.79%
Passage Precision	60.29%	-	39.39%
Passage F1	60.56%	-	51.58%
Combined Recall	69.27%	-	-
Combined Precision	11.13%	11.28%	42.27%
Combined F1	19.18%	18.26%	52.96%
OFEVER Document Score	79.66%	70.20%	92.82%
OFEVER Passage Score	59.86%	62.81%	91.19%
OFEVER Combined Score	35.59%	-	-
Average Execution Time	31.75s	-	-

Table 8: Comparison between our final system compared to the FEVER Baseline (k = 5) and Nie et al. (KM + dNSMN, sNSMN w. AS) implementations that achieved highest FEVER scores tested on the test dataset.

Retrieval of Evidence

Our full evidence retrieval pipeline achieved a recall of 69.27% within our chosen test set. This means that out of all the relevant document-passage pairings, our retriever was able to retrieve a correct document **and** passage 69.27% of the time. This is a solid foundational recall as it means that our system retrieves at least a single correct evidence passage for 69.27% of claims.

Despite achieving a relatively high recall, the combined OFEVER score was only 35.59%, meaning that the system retrieved a **complete set** of correct evidence documents and passages only 35.59% of the time. This could indicate that our system is effective at retrieving relevant documents and passages, however, it is not always effective at retrieving all necessary evidence to support or refute a claim comprehensively. The discrepancy between the high recall and

lower OFEVER score suggests that there are still improvements to be made in ensuring the completeness and precision of the evidence gathered. One possible reason for this discrepancy is that our system performs well on claims that contain multiple "easy" evidence passages, explaining why the recall is high, but worse on claims that only require the retrieval of one passage that is more semantically difficult to obtain, explaining the lower OFEVER score.

Our system also achieved a combined precision of 11.13%, leading the combined F1 score to be 19.18%. These results are to be expected, as we designed our system with the intention of displaying an array of potentially relevant passages to the user, with the user making the final decision which passages were relevant to their claim, therefore we decided to prefer a passage to be displayed, even if it may run the risk of not being relevant as opposed to potentially not displaying relevant passages. Our combined precision and F1 are comparable to the FEVER baseline, which achieved a combined precision of 11.28% and a combined F1 score of 18.26% but was significantly worse than the system made by Nie et al. which achieved a combined precision of 42.27% and a combined F1 of 52.96%. This does show that there is some room for improvement in narrowing down the retrieved passages.

Retrieval of Documents

Our document retriever module achieved a 77.37% recall, meaning that it was able to retrieve a correct document for a given claim 77.37% of the time. This was around a 12% lower recall than the final system made by Nie et al., which leaves some room for improvement, however a lower recall was expected given that we did not use a meticulously crafted rule-based span extraction system, opting to use an NER-base system instead. We could not find any document recall scores for the FEVER baseline system to compare our implementation to, however, it is likely that our system outperformed the FEVER baseline regarding document retrieval as we achieved a higher OFEVER document score than the FEVER baseline (79.66% vs 70.20%).

Our system achieved a 79.66% OFEVER document score, meaning that in 79.66% of claims, **all** relevant documents were retrieved, which is over a 9%

increase over the FEVER baseline, however it is 13% lower than the system made by Nie et al.

Several improvements can be made that may be able to increase the recall of this system. The first is through creating a custom model fine-tuned on FEVER data to extract spans to perform searches. Our system sometimes does not extract the correct keyword span or extracts too many different keyword spans. The second improvement can be made by simply fine-tuning the question generation, or answer extraction models that we use. Sometimes the system will generate a nonsensical question, which can be either caused by an irrelevant entity being extracted, or by the question generation model itself. To solve this problem the answer extraction model could be fine-tuned using claims from this dataset, paired with document titles found in the claim, and the question generation model could be fine-tuned with a set of manually approved questions (generated from claims in the FEVER dataset) and claims as context, ensuring that the model is accustomed to receiving short claims as context.

Our document retrieval achieved a relatively low precision and F1 score, however, as stated previously we designed our system with the intention of favouring a wider catchment of potential evidence.

Passage Retrieval

Our passage retrieval system achieved a surprisingly high precision and F1 score, whilst maintaining a lower recall value. Our passage recall was 60.84% which was significantly lower than the system made by Nie et al. which had a recall of 86.79%. It is fairly likely that our passage retriever's recall was a similar value to the FEVER baseline, as OFEVER passage scores were comparable (59.86% compared to 62.81%). This meant that the passage retrieval system was identifying too many passages as irrelevant, when in fact they could be relevant.

Our passage retriever received higher F1 and precision scores than the system made by Nie et al., and the FEVER baseline. Our precision was 60.29% compared to the system Nie et al. system which had a 39.39% precision, which indicates that our system made less incorrect predictions. This could be because we were classifying less passages altogether as relevant (shown in our lower recall), however we still achieve almost a 9% better F1 score, indicating that the balance of recall and precision of our passage retriever is better in these experiments. It

is worth noting that, despite our high F1 and precision, due to the recall being lower, we achieved a slightly lower OFEVER passage score than the FEVER baseline, and over a 30% worse passage score than the system made by Nie et al.

Presentation of Evidence

We evaluated our final website against Neilsen's Ten Usability Heuristics.

1. Visibility of System Status – Our website kept the users informed about the status of the system within a reasonable amount of time by including a loading screen. This screen provides updates every few seconds of the status of the retrieval process. The average execution time is around 30 seconds, therefore we found it necessary to include this screen.
2. Match Between the System and the Real World – Our website includes multiple sections in the homepage detailing the retrieval process, while certain jargon must be used such as "dense passage retrieval" or "named entities", the section aims to help the user understand any such jargon. The number of technical terms used is also kept to a minimum.
3. User Control and Freedom – If the user accidentally enters a claim and wants to go back and stop the retrieval process, they can click an exit button (pictured in Appendix D), taking them back to the homepage, allowing users to feel as if they remain in control of the system.
4. Consistency and Standards – The website is not overly difficult to understand in the first place, however we use a layout inspired by a Google search, containing a single text field with an enter button on the homepage, and displaying any results in a list format.
5. Error Prevention – We did not encounter any errors when running the website, this is probably due to its very simple design and functionality.
6. Recognition Rather than Recall – All available actions are made visible to the users at any point. These actions only include submitting a claim, cancelling the retrieval, and expanding or collapsing document text.
7. Flexibility and Efficiency of Use – No shortcuts were added for the expert user, however given the limited amount of features in the website it was not considered necessary to add any expert shortcuts.
8. Aesthetic and Minimalist Design – Our website has a simple colour scheme, only containing black text on a white background with some red

accented blocks as to keep the amount of visual noise low. In addition, each page only contains the content that is necessary for the user to see. For example, in the results page it does not contain excessive information about how the process works, instead it only focuses on displaying the claim, passages, and documents. The whole design is primarily structured to let a user enter a claim and see associated evidence.

9. Help Users Recognize, Diagnose, and Recover from Errors – We did not encounter any errors when using the website. There is very little scope for errors to occur given the limited functionality.
10. Help and Documentation – The homepage contains multiple dedicated sections explaining to the user how the evidence retrieval process works, and how to use the website.

Performance of System

Our goal of achieving an evidence retrieval that runs on low hardware specifications in reasonable time appears to have been achieved. On average, for the 200 claims tested, the system took around 30 seconds to complete execution. This would be considered on the upper bound of reasonable time, however no reductions needed to be made to the original dataset and the system is able to search all 5.45 million Wikipedia extracts.

It is worth noting that using a low-end GPU would likely make the system perform significantly better as most of the runtime is taken up through NLP tasks such as extracting entities from the text, generating questions, and performing embedding similarity calculations. All these aspects can be significantly improved by using a low-end GPU instead of the CPU (BUBER & DIRI, 2018).

5 Conclusion

This project, ESOTERIC, demonstrates the feasibility of creating an automated evidence retrieval system as a sub-module of an automated fact-checking system. It retrieves evidence from the entire FEVER document corpus, as to let the system receive a wide catchment of potential users due to its general applicability, promoting the general use of automated fact-checking systems. It does so on consumer hardware, democratising the ability to perform accurate fact-checking, with the evidence retrieval able to be executed entirely on a CPU, within usable runtime of around 30 seconds. It also wraps the evidence retrieval system inside a simple web application as a demonstration as to how this evidence retrieval system could be used as part of a full automated fact-checking process, allowing internet users immediate access to fact-checking facilities which in turn aims to curb the growing pertinence of online misinformation.

When tested on a randomly sampled small subset of claims, ESOTERIC is able to recall 69.27% of the target evidence document-passage combinations. Our document retriever module recalls 77.37% of relevant evidence documents and our passage retriever module recalls 60.84% of relevant evidence passages within those documents. Notably, when tested, our passage retriever module also achieves an F1 score of 60.56%. It is hoped that the relative success of this project will introduce another system to the broader academic field of fact-checking, thereby enhancing the tools available for combatting misinformation.

6 Future Work

While our system performed well in some areas, the improvements below are expected to enhance its overall performance.

Our final system makes use of a dense passage retriever to help rank how well the passages answer key questions about the input claim. This dense passage retriever has the ability to be fine-tuned to a specific dataset, normally resulting in a higher performance. We do not fine-tune the dense passage retriever as it was not achievable within the given time constraints. However, the **dense passage retriever can be fine-tuned** by first taking the input claims, then generating a set of "gold standard" manually approved generated questions. Then finding the answer to those questions in the document text, noting the passage, and manually approving those answers. This data can be formatted as follows to fine-tune the dense passage retriever (Haystack, 2023):

```
{
    "dataset": str,
    "question": str,
    "answers": list of str
    "positive_ctxs": list of dictionaries of format {
        'title': str,
        'text': str,
        'score': int,
        'title_score': int,
        'passage_id': str
    }
    "negative_ctxs": list of dictionaries of format {
        'title': str,
        'text': str,
        'score': int,
        'title_score': int,
        'passage_id': str
    }
    "hard_negative_ctxs": list of dictionaries of format {
        'title': str,
        'text': str,
        'score': int,
        'title_score': int,
        'passage_id': str}
```

```
        }
```

Our system makes use of a question generation system, however, does not filter or rerank these questions. As a result, some questions generated are nonsensical. The questions generated are then used to score documents on how well the document text answers these questions. In cases where poor questions are generated, this can artificially raise the score of irrelevant documents. To improve this, a question reranking model could be trained to assign a predictive score to questions to indicate the quality of the question (regarding how useful it is expected to be to retrieve evidence). This score could be used to filter out bad questions, or alternatively, it could be used as a weight so that documents retrieved using bad quality questions are given a lower final document score.

Additionally, we use the concatenation of an NER model and an answer extraction model to extract keywords from the claim. These models were not designed to extract keywords from short input claims. Training a custom model for keyword extraction from claims would likely improve performance as it would reduce the number of useless keywords being extracted. For example, the model will sometimes extract articles, leading to useless questions being generated, and making searching through document text slower as well as returning more irrelevant documents. This could be done by getting a set of claims found in the FEVER dataset, running them through our keyword extraction system, and removing "useless" keywords generated. Then this could be used as a dataset to fine-tune a language model.

A question generation model that generates questions without needing an answer could be used, cutting out the stage of extracting keywords to be used as answers to questions. To do this we would require training data from a dataset that contains claims, and useful fact-checking questions surrounding each claim. This type of question generation is generally seen as a more challenging NLP task (Do, et al., 2023), which is why it was not used in the final solution.

Different evaluation strategies were used to obtain the recall, precision, and OFEVER scores for our system, compared to the FEVER baseline and Nie et al. solution. As such, one future improvement that could be made is to evaluate the FEVER baseline and the Nie et.al solution using the same 200 claim subset mentioned in 4.2.1.1. This would likely produce a more accurate comparison.

7 Reflection

I began this task with absolutely no prior knowledge of machine learning or NLP. This meant that I was completely unaware of how to train a machine learning model, for NLP or for any other purpose, and I was also unaware of how machine learning worked in detail, especially in the field of NLP. Additionally, I did not have a fundamental idea of how any information/evidence retrieval systems worked. As such, many assumptions made during the development process are reflective of some naivety.

The development of this project gave me foundational knowledge of exactly how language models work, as well as an insight into a range of different NLP tasks such as tokenisation, question answering, named entity recognition, and dependency parsing. One of the most significant skills I acquired was learning to train NLP models. This involved not just the technical aspects of setting up and running training sessions but also the nuances of selecting appropriate data sets, preprocessing data, choosing the right model architectures, and fine-tuning parameters to improve model accuracy and efficiency. My understanding of information retrieval systems was also greatly enhanced. I learned how these systems work to index, search, and retrieve information from large datasets, which is crucial for building efficient fact-checking systems that rely on quick access to accurate data. This project also helped me to improve my report-writing skills. Having struggled with academic writing in other stages of my degree, I was forced to practice writing complex technical processes and explaining them in a clear and concise manner. I also gained valuable experience working with large datasets, including storing, querying, searching, and manipulating the FEVER dataset to fit my needs.

When performing this task, I ended up spending a significant amount of time on testing solutions that didn't work. This time could have been spent elsewhere if I had changed some initial assumptions and the overall approach of the project. Approaching the project, I assumed that it could be done in a "head-first" manner, meaning that I could start by cycling through many different techniques, evaluating how well they work to retrieve evidence from the FEVER dataset. For example, I tried to implement a TF-IDF system in Version 0.1, taking up

almost an entire week when this approach was not useful. Given that I now know how long it took, I would choose not to consider this approach at all. It turns out that this process was time-consuming as each system and evaluation script needed to be implemented to specifically work with my code-base and dataset. When approaching a project similar to this in the future, I would change the assumption that this project should be done "head-first" and instead adopt the assumption that, to undertake this project, a significant amount of research should be done into different methods as well as their overall effectiveness for this specific dataset, then one should be chosen and focused on, removing the need to evaluate different methods internally. This highlights the importance of doing adequate research in preparation for a task.

In a similar vein, I believe that this project could have achieved better results if more focus was attributed to fine-tuning one particular method as opposed to finding the optimal structure of different technique pipelines. The reason why I decided to structure the project as a concatenation of different techniques was because I operated under the initial assumption that good information retrieval systems needed different techniques to retrieve different types of information. The result of this assumption was many techniques were evaluated sacrificing a level of deep analysis. The alternative approach, which seems better retrospectively, would have been to simply focus on one method of retrieval, like dense passage retrieval, fine tuning this method and reporting the results in a higher level of detail. I would instead operate under the initial assumption that a good information retrieval system for this task probably involves a range of techniques, but mainly relies on one technique that should be focused on, with the other techniques acting as minor additions to capture niche information. This emphasises the importance of having a proper project structure identified before execution.

If I were to redo this project, I would also not have tried to focus so much on making it perform well on a low-end machine. Due to my lack of experience in NLP, I assumed that many NLP tasks were, in general, designed to perform with consumer hardware, with advanced hardware providing quicker execution time which I didn't think was needed in this project. However, when working with such a large amount of data, the hardware of the system made a lot more

difference than I initially thought. As a result, I severely underestimated the difficulty of creating an information retrieval system for a dataset this large on low-end hardware. This likely hindered my final solution as many models were chosen due to their ability to run within reasonable time on low-end hardware rather than their overall suitability to the task. Additionally, some steps such as filtering out documents that didn't contain any extracted keywords may not need to be applied if the system was not constrained to work on low-end hardware. The end system did end up having a better execution time, however different techniques have been entirely filtered out as they were not able to be executed on the system. In reality it may have been more interesting to pursue these without the hardware constraint to figure out their potential. In the future I would remove the assumption that an information retrieval system of this size should be performed on low-end hardware, so that the best version of the system could be found, with performance optimisation coming as a later stage in a separate project.

Approaching this task, I did not consider that the size of the dataset would affect the project in as significant of a way as it ended up doing. As a result, I wanted to try and give the evidence retrieval system the ability to search over as broad of a document corpus as possible. However, this obviously affected factors such as execution time as we performed keyword searches on every text in the corpus. As execution time was a factor that contributed to the evaluation of the performance each version, this influenced other changes made such as the size of language models used. This means that if we chose a smaller initial document corpus, we may have ended up with a significantly better performing system. There are some datasets which focus on specialised topics with a far smaller document corpus, which retrospectively seem to be a better fit for this project, given our hardware limitations. An example of this is CLIMATE-FEVER (Diggelmann , et al., 2021).

All else equal, given the way the task was undertook, I think that my method of testing and comparing to previous systems as we went was a relatively good approach. I made the initial assumption that I would be implementing a wide range of techniques, which may all have different interactions when combined to make an overall system. Therefore, I decided to test each version as soon as a

major change was made or a new feature was added, giving me the ability to see if a certain change was worth making.

This project has shown me firsthand the importance of proper planning, research, and evaluation, as well as giving me invaluable experience due to the sheer amount of new machine learning and NLP related skills and knowledge that I have learned in the process.

8 Bibliography

Bernhard, U. & Dohle, M., 2015. *Corrective or Confirmative Actions? Political Online Participation as a Consequence of Presumed Media Influences in Election Campaigns.* [Online]
Available at:
https://www.tandfonline.com/doi/full/10.1080/19331681.2015.1048918
[Accessed 02 02 2024].

BUBER, E. & DIRI, B., 2018. Performance Analysis and CPU vs GPU Comparison for Deep Learning. *2018 6th International Conference on Control Engineering & Information Technology (CEIT),* pp. 1-6.

Deepset, 2024. *Introduction to Haystack 2.0.* [Online]
Available at: https://docs.haystack.deepset.ai/docs/intro
[Accessed 02 05 2024].

Del Vicario, M. et al., 2016. The spreading of misinformation online. *Proceedings of the National Academy of Sciences ,* 04 01, 113(3), pp. 554-559.

Devendra Sachan, D. S. et al., 2023. Questions Are All You Need to Train a Dense Passage Retriever. *Transactions of the Association for Computational Linguistics,* Volume 11, p. 600–616.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K., 2019. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* [Online]
Available at: https://arxiv.org/pdf/1810.04805.pdf
[Accessed 20 03 2024].

Diggelmann , T. et al., 2021. CLIMATE-FEVER: A Dataset for Verification of Real-World Climate Claims.

Do, X. L. et al., 2023. Modeling What-to-ask and How-to-ask for Answer-unaware Conversational Question Generation. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics,* Volume 1, p. 10785–10803.

Guo, Z., Schlichtkrull, M. & Vlachos, A., 2022. *A Survey on Automated Fact-Checking.* [Online]
Available at: https://arxiv.org/abs/2108.11896
[Accessed 08 03 2024].

Haystack, 2023. *https://haystack.deepset.ai/tutorials/09_ dpr_ training.* [Online]
Available at: https://haystack.deepset.ai/tutorials/09_dpr_training
[Accessed 06 05 2024].

Hugging Face, 2024. 🤗 *Transformers.* [Online]
Available at: https://huggingface.co/docs/transformers/en/index
[Accessed 02 05 2024].

Hugging Face, 2024. *mistralai/Mistral-7B-v0.1.* [Online]
Available at: https://huggingface.co/mistralai/Mistral-7B-v0.1
[Accessed 08 05 2024].

Hugging Face, 2024. *Natural Language Processing.* [Online]
Available at: https://huggingface.co/learn/nlp-course/en/chapter1/2
[Accessed 11 04 2024].

iarfmoose, 2024. *iarfmoose/t5-base-question-generator.* [Online]
Available at: https://huggingface.co/iarfmoose/t5-base-question-generator
[Accessed 29 04 2024].

International Organization for Standardization, 2023. *Accuracy (trueness and precision) of measurement methods and results — Part 1: General principles and definitions.* [Online]
Available at: https://www.iso.org/obp/ui/#iso:std:iso:5725:-1:ed-2:v1:en
[Accessed 20 03 2024].

Jiang, A. Q. et al., 2023. Mistral 7B.

Karpukhin, V. et al., 2020. *Dense Passage Retrieval for Open-Domain Question Answering.* [Online]
Available at: https://scontent-lhr6-2.xx.fbcdn.net/v/t39.8562-6/240836864_1003700667032628_7482481494695845774_n.pdf?_nc_cat=104&ccb=1-7&_nc_sid=e280be&_nc_ohc=4LNOVIKPS90Q7kNvgH4Q3ch&_nc_oc=AdjEEJnlfOzNM0QcLUqWkCaVnHlzJ4FpLWp7AmPZTXbBMvFdwql1rX1o7pzkBwyjv6s&_nc_ht=sco
[Accessed 29 04 2024].

Karpukhin, V. et al., 2020. Dense Passage Retrieval for Open-Domain Question Answering. *2020 Conference on Empirical Methods in Natural Language Processing,* p. 6769–6781.

Kathare, N., Reddy, V. O. & Prabhu, V., 2021. *A Comprehensive Study of Elasticsearch.* [Online]
Available at: https://www.ijsr.net/archive/v10i6/SR21529233126.pdf
[Accessed 30 04 2024].

King, D. R., 1974. The binary vector as the basis of an inverted index file. *Journal of library automation,* 7(December 1974), pp. 307-314.

Kwiatkowski, T. et al., 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics,* Volume 7, p. 452–466.

Lee, M. L., Hsu, W. & Samarinas, C., 2021. Improving Evidence Retrieval for Automated Explainable Fact-Checking. *North American Chapter of the Association for Computational Linguistics,* Volume Human Language Technologies: Demonstrations, p. 84–91.

Lek, S. & Park, Y.-S., 2016. Artificial Neural Networks: Multilayer Perceptron for Ecological Modelling. *Developments in Environmental Modelling,* Volume 28, pp. 123-140.

Linden, S. v. d., 2022. Misinformation: susceptibility, spread, and. *Nature Medicine,* Volume 28, p. 460–467.

Liu, S. et al., 2017. Visual Exploration of Semantic Relationships. *IEEE,* 24(1), pp. 553 - 562.

Mathew, S. K. & T, S. M., 2022. The disaster of misinformation: a review of research in social media. *International Journal of Data Science and Analytics,* 13(4), p. 271–285.

Mistral AI, 2024. *Mistral AI API (0.0.2).* [Online]
Available at: https://docs.mistral.ai/api/
[Accessed 08 05 2024].

Mulla, N. & Gharpure, P., 2023. Automatic question generation: a review of methodologies, datasets, evaluation metrics, and applications. *Progress in Artificial Intelligence,* 12(1), pp. 1-32.

Nadeau, D. & Sekine, S., 2007. A survey of named entity recognition and classification. *Lingvisticæ Investigationes,* 30(1), pp. 3-26.

Nielsen, J., 1994. *10 Usability Heuristics for User Interface Design.* [Online]
Available at: https://www.nngroup.com/articles/ten-usability-heuristics/
[Accessed 02 05 2024].

Nie, Y., Chen, H. & Bansal, M., 2018. *Combining Fact Extraction and Verification with Neural Semantic Matching Networks.* [Online]
Available at: https://arxiv.org/pdf/1811.07039.pdf
[Accessed 20 03 2024].

Powers, D. M. W., 2020. *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.* [Online]
Available at: https://arxiv.org/pdf/2010.16061.pdf
[Accessed 20 03 2024].

Raffel, C. et al., 2023. *Exploring the Limits of Transfer Learning with a Unified.* [Online]
Available at: https://arxiv.org/pdf/1910.10683.pdf
[Accessed 11 04 2024].

Rajpurkar, P., Jia, R. & Liang, P., 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. Volume Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), p. 784–789.

Rajpurkar, P., Zhang, J., Lopyrev, K. & Liang, P., 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. Volume Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, p. 2383–2392.

Sanh, V., Debut, L., Chaumond, J. & Wolf, T., 2020. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper, and lighter.* [Online]
Available at: https://arxiv.org/pdf/1910.01108.pdf
[Accessed 20 03 2024].

Schmidt, R. M., 2019. Recurrent Neural Networks (RNNs): A gentle Introduction and Overview.

Sentence-Transformers, 2024. *Pretrained Models.* [Online]
Available at: https://www.sbert.net/docs/pretrained_models.html
[Accessed 19 04 2024].

Shvaiko, P., Yatskevich, M. & Giunchiglia, F., 2007. Semantic Matching: Algorithms and Implementation. *Journal on Data Semantics IX,* pp. 1-38.

Tedeschi, S. et al., 2021. *WikiNEuRal: Combined Neural and Knowledge-based.* [Online]
Available at: https://aclanthology.org/2021.findings-emnlp.215.pdf
[Accessed 01 05 2024].

Thorne, J., Vlachos, A., Christodoulopoulos, C. & Mittal, A., 2018. *FEVER: a large-scale dataset for Fact Extraction and VERification.* [Online]
Available at: https://arxiv.org/abs/1803.05355
[Accessed 11 03 2024].

Thorne, J. et al., 2018. *The Fact Extraction and VERification (FEVER) Shared Task.* [Online]
Available at: https://arxiv.org/abs/1811.10971
[Accessed 08 03 2024].

Vaswani, A. et al., 2017. *Attention Is All You Need.* [Online]
Available at: https://arxiv.org/pdf/1706.03762.pdf
[Accessed 20 03 2024].

Wang, L., Qian, L., Zheng, K. & Li, S., 2022. A Survey of Extractive Question Answering. *2022 International Conference on High Performance Big Data and Intelligent Systems (HDIS),* pp. 147-153.

Webster, J. J. & Kit, C., 1992. Tokenization as the Initial Phase in NLP. *COLING,* Volume COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics.

Wei, J. et al., 2022. Emergent Abilities of Large Language Models.

Zafar, A. et al., 2024. KI-MAG: A knowledge-infused abstractive question answering system in medical domain. *Neurocomputing,* Volume 571.

9 Appendix

Appendix A

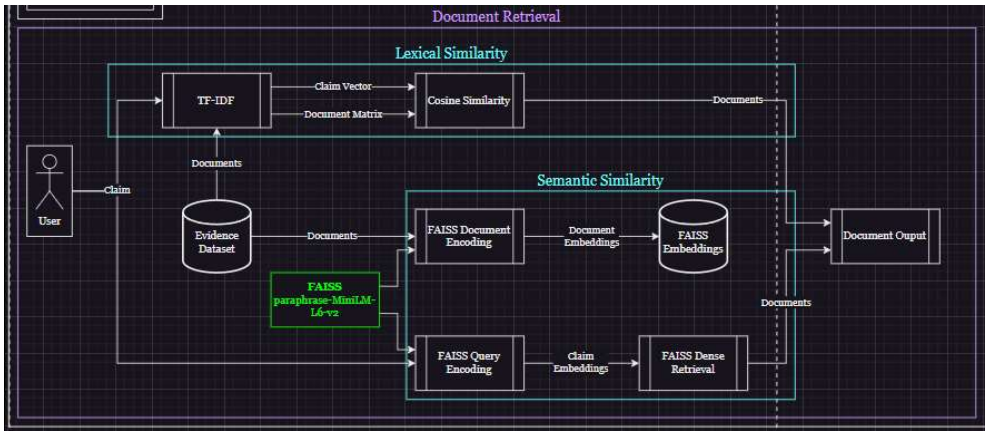
Version history

Metric	V0.1	V1.0	V1.1	V1.2	V1.3	V1.4	V1.5	V1.6	V1.7	Final
Title Match (as % of hits)	-	88.64%	61.66%	58.00%	88.46%	89.94%	85.64%	85.64%	80.84%	85.64%
Disambiguation (as % of hits)	-	5.11%	5.13%	4.46%	4.95%	5.03%	9.57%	9.57%	8.14%	9.57%
Text Match (as % of hits)	-	6.25%	33.20%	37.55%	6.59%	5.03%	4.79%	4.79%	11.02%	4.79%
Document Recall	-	72.42%	73.25%	75.31%	74.90%	73.66%	77.36%	77.36%	77.36%	77.36%
Document Precision	-	7.45%	4.78%	9.18%	9.98%	12.14%	6.24%	6.24%	6.24%	6.24%
Document F1	-	13.51%	8.98%	16.36%	18.61%	20.84%	11.55%	11.55%	11.55%	11.55%
Passage Recall	-	-	-	-	69.91%	-	-	21.33%	47.90%	60.81%
Passage Precision	-	-	-	-	59.85%	-	-	70.07%	60.86%	60.28%
Passage F1	-	-	-	-	64.49%	-	-	32.70%	53.61%	60.56%
Combined Recall	-	-	-	-	78.06%	-	-	24.01%	59.51%	69.27%
Combined Precision	-	-	-	-	14.55%	-	-	10.08%	5.33%	11.13%
Combined F1	-	-	-	-	25.53%	-	-	14.20%	9.78%	19.18%
OFEVER Document Score	-	74.01%	75.14%	76.84%	77.40%	74.01%	79.66%	79.66%	78.00%	79.66%
OFEVER Passage	-	-	-	-	74.52%	-	-	30.43%	52.27%	59.86%
OFEVER Combined	-	-	-	-	55.37%	-	-	19.77%	31.07%	35.59%
Average Execution Time	381928s	28.30s	39.04s	57.36s	67.36s	18.27s	11.53s	36.51s	36.24s	31.75s
Average Document Set Length	-	13.60	13.60	13.60	13.60	13.60	13.60	13.60	15.06	13.60
Average Position of Correct Document (in textually matched documents)	-	10.00	7.02	8.50	8.25	5.89	8.72	8.72	7.81	8.72

9.1.1 Version 0.1

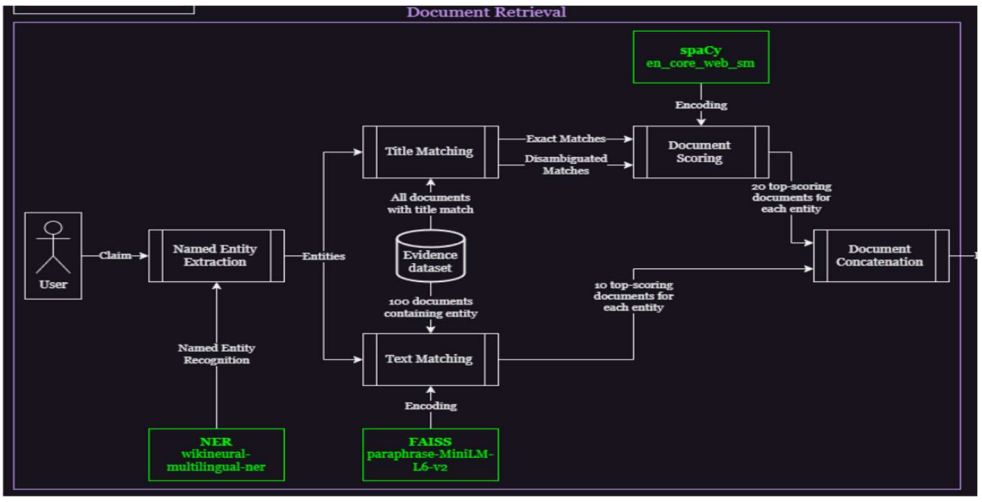
Version 0.1 attempted to calculate the TF-IDF matrix of the entire FEVER dataset beforehand, storing the matrix inside an SQLite3 database. It also tried to compute the embeddings of the entire FEVER dataset using the *paraphrase-MiniLM-L6-v2* model, storing them inside a large FAISS index. The system would then calculate the TF-IDF vector of the claim, finding the N closest documents in the TF-IDF matrix using cosine similarity. It would then represent

the claim in a high-dimensional vector format using *paraphrase-MiniLM-L6-v2* model, retrieving the N closest document vectors. Ultimately this system was completely rewritten due to the fact that it couldn't be run on the specifications listed in 3.1.1, requiring a V100 GPU to initialise. Even when running on the V100 GPU, a single claim was calculated to take 106 hours to retrieved evidence for.



9.1.2 Version 1.0

Version 1.0 restructured the entire project to follow a similar structure the Nie et al. solution. This included their title search and matching method involving disambiguation information. The system extracted keyword spans using wikineural-multilingual-ner, attempting to find titles that matched these spans (including those with "disambiguation" information). To widen the retrieval, we also searched for N documents containing the extracted keyword spans in the document text. We then calculated the vector embeddings of both the claim and the textually matched documents using *paraphrase-MiniLM-L6-v2*, calculating the N most similar documents by using FAISS dot product similarity.



9.1.3 Version 1.1

Version 1.1 was more effective than Version 0.1 however struggled to rank documents. The semantic similarity between *paraphrase-MiniLM-L6-v2* embeddings were not effective at ranking relevant documents, therefore we devised a ranking method to order documents by how well they answered questions about the claim. To do this we extracted answer spans using *t5-answer-extraction-small*, generating claims around these answers. We then used *t5-base-finetuned-question-generation-ap* to generate questions surrounding the claim and *tinyroberta-squad2* to attempt to answer these questions using each document text, using the confidence score as the final document score. This achieved better results than simply using semantic similarity as a ranking method.

9.1.4 Version 1.2

Version 1.2 used FTS5 to search for spans in document texts in the SQLite3 database, greatly improving the execution time.

9.1.5 Version 1.3

Version 1.3 implemented the first passage retriever. We trained a DistilBERT classification model to classify whether a sentence is relevant (i.e. supports or refutes) a given claim. We then split each document text into sentences and pass

it into the model alongside the claim, which classifies whether the sentence is relevant to supporting or refuting the claim.

9.1.6 Version 1.4

Version 1.4 changed database systems from SQLite3 with FTS5 to an Elasticsearch database. This greatly improved the execution time.

9.1.7 Version 1.5

Version 1.5 replaces the previous system of calculating the document score by using the confidence score given by *tinyroberta-squad2* with a dense passage retrieval system. To do this all documents in the database needed to be encoded using *dpr-ctx_ encoder-single-nq-base* and have their embeddings stored in the Elasticsearch database. The questions were encoded using *dpr-question_ encoder-single-nq-base* and the retrieved textually matched documents and disambiguative documents were then ranked according to how similar their context embeddings were to the set of question embeddings.

9.1.8 Version 1.6

Version 1.6 replaces Version 1.3's passage retriever with the QA reader FARMReader equipped with *tinyroberta-squad2*. This attempted to try and find which passages answered questions about the claim.

9.1.9 Version 1.7

Version 1.7 added a parallel pipeline to the passage retrieval designed to retrieve sentences that were semantically similar to the claim. This was done by splitting each document text into sentences, then calculating the sentences with the highest BM25 similarity score to the claim.

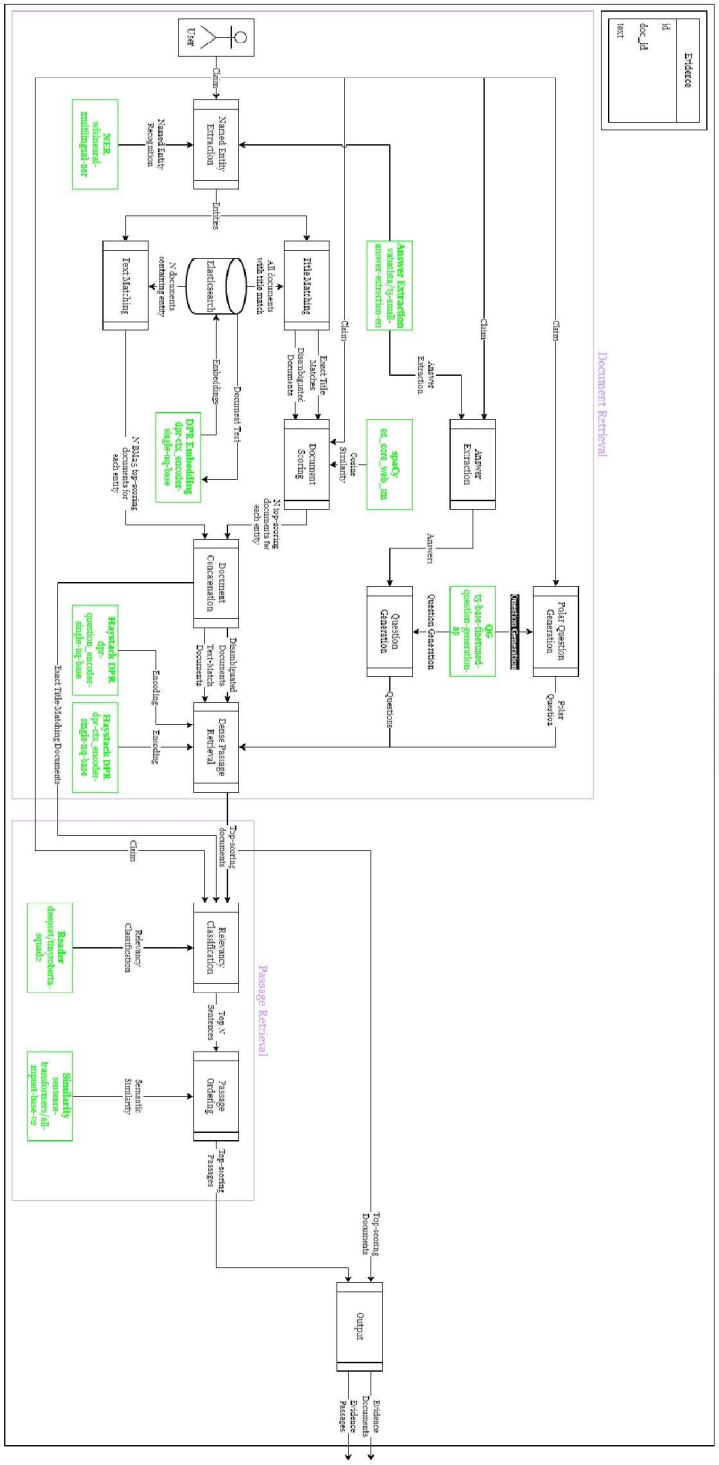
9.1.10 Final Version

The final version of the system replaces the passage retrieval module in Version 1.7 with the passage retrieval module in 1.3, after finding that the relevancy classification model in 1.3 was more effective at retrieving passages than 1.7. It

also adds a final step to the passage retrieval process, sorting the passages by semantic similarity to the claim.

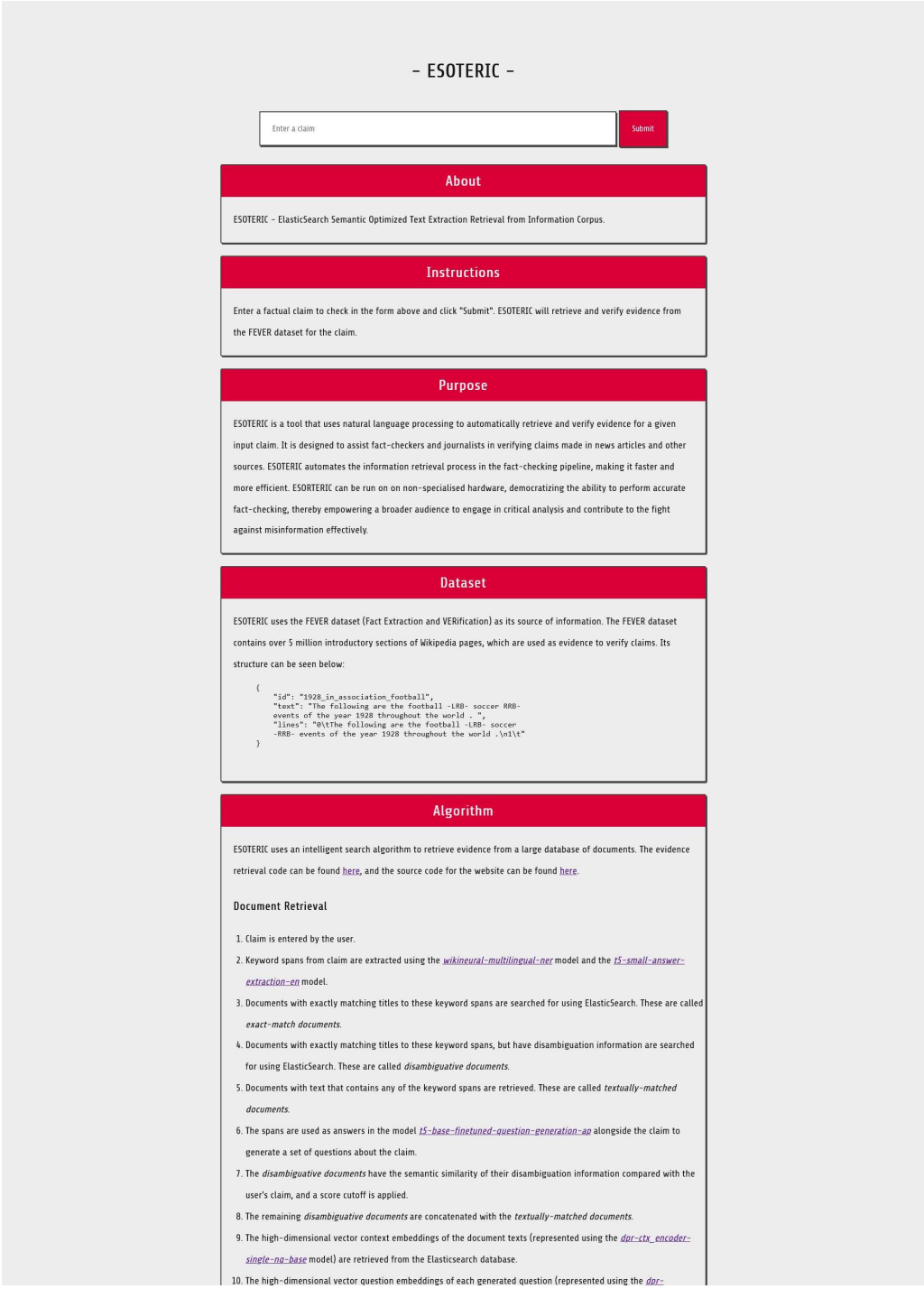
Appendix B

Data flow diagram



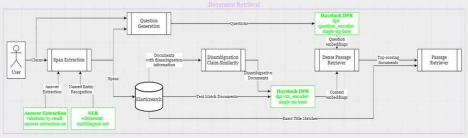
Appendix C

Screenshot of Homepage



question_encoder-single-nq-base model) are calculated.

11. The angle in the vector space is measured between each document text embedding and each question embedding using a *Dense Passage Retriever*. The closest angle for each document is saved as its document score.
12. A cutoff score is applied to the documents, having the effect of retrieving the closest documents to each question.
13. The final set of documents is passed into the passage retriever.



Passage Retrieval

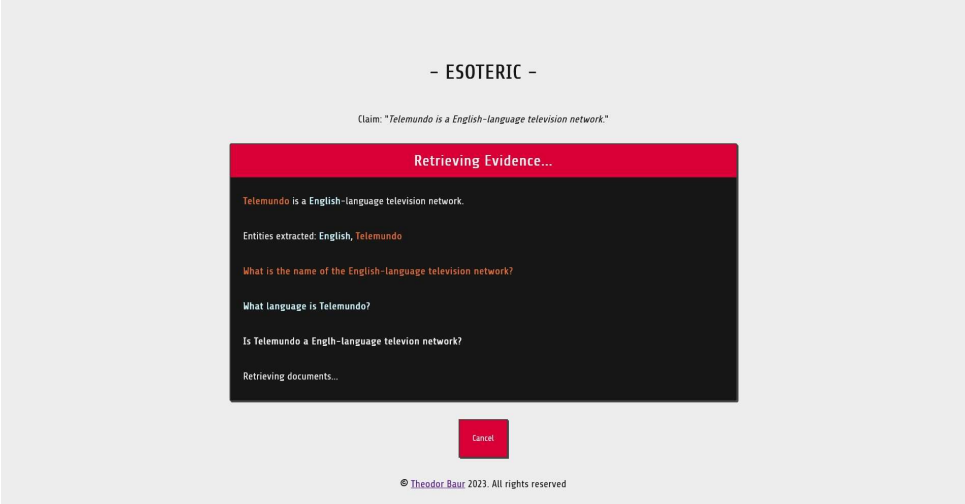
1. All document texts are split into individual sentences.
2. Each sentence is passed through our classification model, which is fine tuned with data from the FEVER dataset.
3. The model classifies each sentence as either irrelevant or relevant to the claim.
4. The relevant sentences are returned as evidence for the claim.

References

- FEVER: a large-scale dataset for Fact Extraction and VERification
- FEVEROUS: Fact Extraction and VERification Over Unstructured and Structured information (arXiv)
- wikineural-multilingual-ner
- t5-small-answer-extraction-en
- t5-base-finetuned-question-generation-ap
- dpr-ctx_encoder-single-nq-base
- dpr-question_encoder-single-nq-base
- relevancy_classification_FEVER

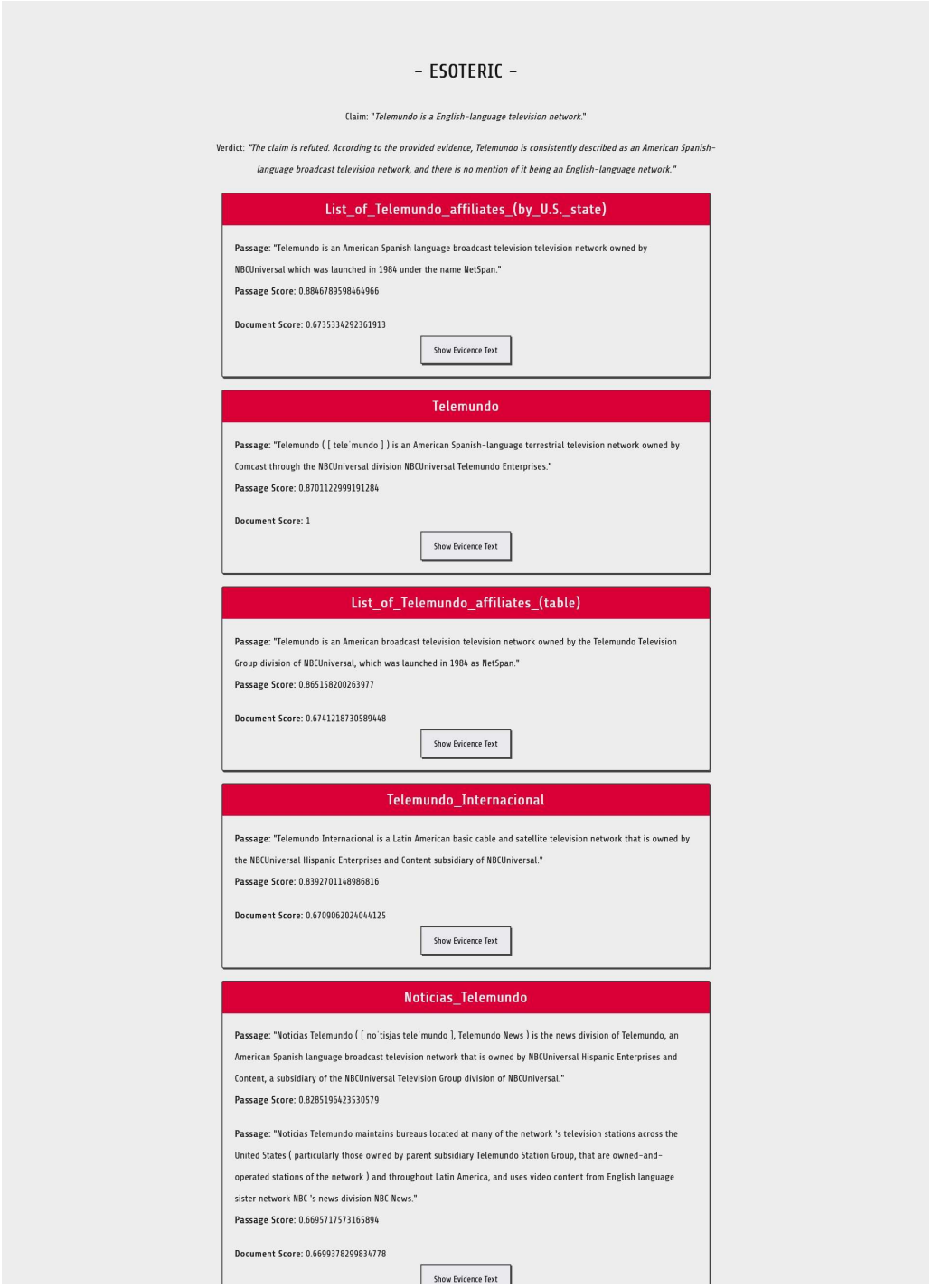
Appendix D

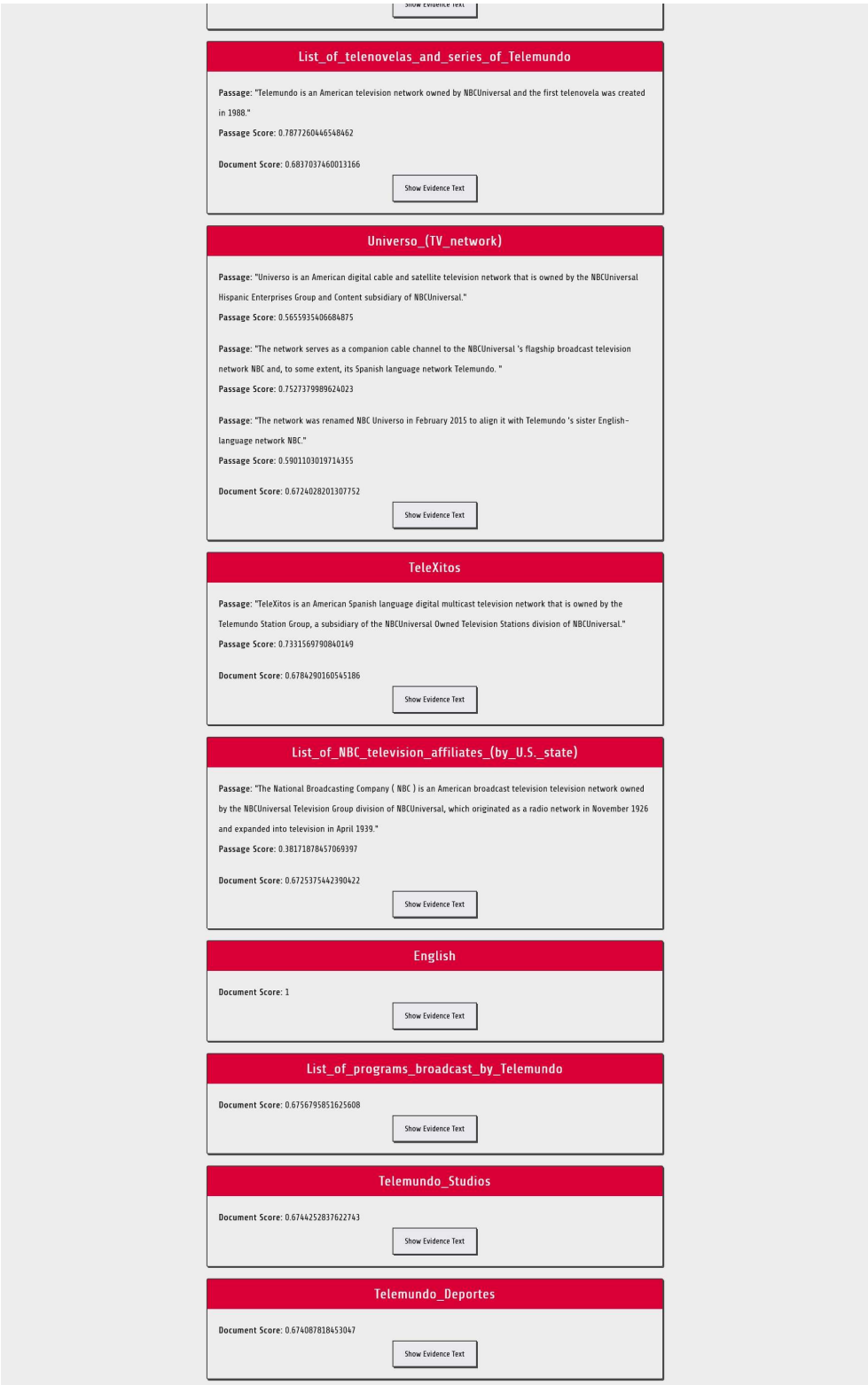
Screenshot of Loading Page



Appendix E

Screenshot of Results Page




List_of_telenovelas_and_series_of_Telemundo
Passage: "Telemundo is an American television network owned by NBCUniversal and the first telenovela was created in 1988."
Passage Score: 0.78772604465484 62
Document Score: 0.6837037460013166
Show Evidence Text
Universo_(TV_network)
Passage: "Universo is an American digital cable and satellite television network that is owned by the NBCUniversal Hispanic Enterprises Group and Content subsidiary of NBCUniversal."
Passage Score: 0.5655935406684875
Passage: "The network serves as a companion cable channel to the NBCUniversal 's flagship broadcast television network NBC and, to some extent, its Spanish language network Telemundo. "
Passage Score: 0.7527379989624023
Passage: "The network was renamed NBC Universo in February 2015 to align it with Telemundo 's sister English-language network NBC."
Passage Score: 0.5901103019714355
Document Score: 0.6724028201307752
Show Evidence Text
TeleXitos
Passage: "TeleXitos is an American Spanish language digital multicast television network that is owned by the Telemundo Station Group, a subsidiary of the NBCUniversal Owned Television Stations division of NBCUniversal."
Passage Score: 0.7331569790840149
Document Score: 0.6784290160545186
Show Evidence Text
List_of_NBC_television_affiliates_(by_U.S._state)
Passage: "The National Broadcasting Company (NBC) is an American broadcast television television network owned by the NBCUniversal Television Group division of NBCUniversal, which originated as a radio network in November 1926 and expanded into television in April 1939."
Passage Score: 0.38171878457069397
Document Score: 0.6725375442390422
Show Evidence Text
English
Document Score: 1
Show Evidence Text
List_of_programs_broadcast_by_Telemundo
Document Score: 0.6756795851625608
Show Evidence Text
Telemundo_Studios
Document Score: 0.6744252837622743
Show Evidence Text
Telemundo_Deportes
Document Score: 0.674087818453047
Show Evidence Text

Appendix F

1. **Visibility of System Status** – The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time.
2. **Match Between the System and the Real World** - The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order.
3. **User Control and Freedom** - Users often perform actions by mistake. They need a clearly marked "emergency exit" to leave the unwanted action without having to go through an extended process.
4. **Consistency and Standards** - Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions.
5. **Error Prevention** - Good error messages are important, but the best designs carefully prevent problems from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
6. **Recognition Rather than Recall** - Minimize the user's memory load by making elements, actions, and options visible. The user should not have to remember information from one part of the interface to another. Information required to use the design (e.g. field labels or menu items) should be visible or easily retrievable when needed.
7. **Flexibility and Efficiency of Use** - Shortcuts — hidden from novice users — may speed up the interaction for the expert user so that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
8. **Aesthetic and Minimalist Design** - Interfaces should not contain information that is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.

9. **Help Users Recognize, Diagnose, and Recover from Errors** - Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.
10. **Help and Documentation** - It's best if the system doesn't need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.

Appendix G

https://github.com/theobaur13/ESOTERIC

Appendix H

https://github.com/theobaur13/ESOTERIC-website