Assignment 3, Applied Natural Language Processing

Li Fu Chuen
The University of Adelaide
a1865844@student.adelaide.edu.au

Ka Hing Chan
The University of Adelaide
a1867200@student.adelaide.edu.au

Manish Adhikari
The University of Adelaide
a1876371@student.adelaide.edu.au

1. Executive summary

This project aims to implement the Closed domain factoid question-answering system (QA) on Open Research Dataset Challenge (CORD-19) to answer a question. This project uses only 10,000 random documents from COVID-19 Open Research Dataset Challenge (CORD-19) as a corpus due to the limited computation power.

The process of our QA system involves several steps, including document and query processing, document query matching, document retrieval, document summarization, answer extraction, and text to speech output. Our QA system uses various techniques such as coreferencing, name entity recognition, word embedding generation, article indexing, BART summarization, BERT answer extraction, and Google Text to Speech.

The performance of an intermediate information retrieval system (IR) and a question-answering system (QA) are evaluated using Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP) metrics and Accuracy respectively. The results indicate that the IR system in Assignment 3 outperforms Assignment 2, with MRR and MAP scores of 0.733 and 0.393, respectively, compared to 0.303 and 0.147 for Assignment 2. However, the improved performance comes at the cost of a threefold increase in execution time. We also compare the accuracy different BERT models. The best performance model is BERT fine-tuned with SQuAD 2.0 dataset. The QA system demonstrates a accuracy 0.5 when comparing its output to the ground truth in 10 questions. It can provide plausible, albeit incorrect, answers. The identified bottleneck in the QA system is the performance of the named entity recognition. Future work could explore the use of fine-tuned or primed pre-trained models to enhance the system's accuracy.

2. Analysis of Methods

2.1. Coreference

In linguistics, coreference refers to the phenomenon where two or more expressions refer to the same object. NeuralCoref, developed by Thomas Wolf, is used for performing coreference in this project and is based on a neural network [15]. The algorithm implemented in NeuralCoref first collects the mentions and their context word embedding. The word embedding for each mention is then augmented with the word embedding for words around it to incorporate contextual information [14]. Furthermore, features of the mention such as its length and location are also added to its word embedding [14]. The neural network architecture of NeuralCoref is relatively simple, consisting of two separate fully-connected neural networks with ReLU activation functions [14]. Figure 1 has shown the overview of the neural network architecture. The first neural network(The upper part of Figure 1) takes a pair of mention word embeddings as input and evaluates the likelihood that the two mentions are coreferent. The second neural network(The lower part of Figure 1 takes a mention word embedding as input and computes the likelihood of the mention having no coreference [14]. The neural networks produce scores in arbitrary units rather than probabilistic outcomes [14]. The candidate with the highest score is the predicted outcome amongst all possible candidates.

Generally, both Rule-based and neural network-based coreference tools are used for coreferencing. Rule-based tools are limited by their design rules which can lead to high precision but low recall. In contrast, neural network-based approaches learn coreference patterns directly from the corpus. As a result, their overall performance is superior. Thus, the neural network approach is selected for this project.

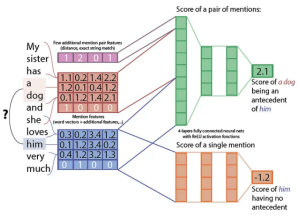

Figure 1. The architecture of the neural network for coreference

2.2. Name Entity Extraction

Name Entity Extraction (NER) function in Spacy is based on a neural network that consists of a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) variant, specifically a Long-Short Term Memory (LSTM).

The architecture of the CNN is shown in Figure 2. The CNN is responsible for extracting character-level features from the words. A randomly initialized character embedding, along with additional character features such as capitalization and punctuation, is fed into a convolutional layer followed by a max pooling layer [5]. The resulting vector in the max pooling layer represents the CNN-extracted character features of the word.

The character feature vector is then fed into the Bi-directional LSTM along with the word embedding and additional word features such as capitalization and lexicon information, such as single-token entry, as shown in Figure 3. The input as a whole is then fed into the Bi-LSTM. The hidden states of the forward-LSTM and backward-LSTM are then fed into a linear layer and a softmax layer separately, to yield the likelihood score of the class of the word [5]. The two scores are added up, and the class with the highest score is assigned to the word.

When performing NER, rule-based approaches, CNN-LSTM neural networks, and transformers are all popular choices. However, CNN-LSTM neural networks are generally more robust to noise than rule-based approaches, which may not be able to capture all variations in complex sentences. While transformers are also robust to noise, they are much more computationally expensive.

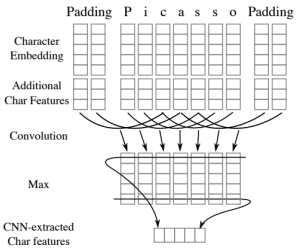

Figure 2. A CNN for extracting character level features in NER

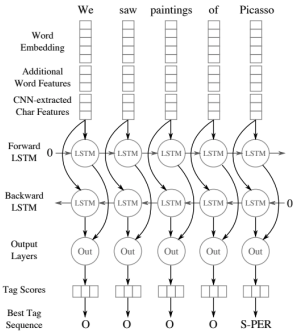

Figure 3. A LSTM for generating entity tag in NER

2.3. Word Embedding

Fasttext is a word embedding tool that considers both words and subwords [3]. The word embedding is trained using the continuous skip-gram model, which is based on the distributional hypothesis that words occurring in similar contexts tend to have similar meanings. The skip-gram model is trained to predict the context words of a target word given the target word vector as input, to maximize the likelihood of the context words given the target word [3]. The objective of skipgram is illustrated in Equation 1.

The skip-gram model is trained to maximize the probability of predicting the context words w_c given a center word w_t. The objective function of the skip-gram model is given by:

$$\arg\max_{w_c} \sum_{t=1}^{T} \sum_{c \in C_t} \log p(w_c|w_t) \tag{1}$$

where, w_c is the context words, w_t is the target word and C_t is the set of context word.

The subword model is built to consider the subwords that constitute a word, as subwords often represent meaningful units. The subword model first breaks down the word into n-grams and trains a vector representation for each subword. For example, the word "where" would be broken down into ¡wh, whe, her, ere, re¿ if using 3-grams while a subword model often breaks the word into multiple n-grams [3]. The word embedding of a word is then the weighted sum of all the subword vectors that constitute the word, with the weight being the frequency of the subword happening in the word [3]. During training, the model works similarly to the skip-gram model in Equation 1, aiming to maximize the likelihood of the context word given the target word, but with the target word representation being constituted by subwords rather than a word vector [3].

Two-word vectors are generated from the skip-gram model and the subword model, and are concatenated to form the word embedding of the word.

While Word2Vec and TF-IDF are strong tools for training word embeddings, they only consider word-level embedding, whereas Fasttext considers both word-level and subword-level embedding. This makes Fasttext better suited for handling out-of-vocabulary words in queries, as the subwords can be used to generate an embedding for the out-of-vocabulary word.

2.4. BART fine-tuned with CNN-news (Summarization)

BART is a language model that uses a combination of Bidirectional and Auto-Regressive Transformers for pre-training as shown in Figures 4, 5, and 6. It is built with a denoising autoencoder sequence-to-sequence model that can be applied to various natural language processing tasks [11]. To generate a summary of a given input text, BART follows a step-by-step process. First, the input text is fed into the model in the form of tokens. Then, BART uses its bidirectional encoder to analyze the input text and understand its context. Once the context is captured, BART employs its auto-regressive decoder to generate a summary of the input text, predicting one token at a time based on the previous ones generated [11]. The summary generation process continues until a stopping criterion is met, such as a predefined length or an end-of-summary token.

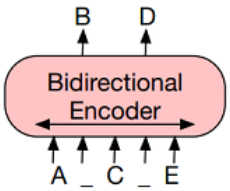

Figure 4. The Bidirectional encoder structure of BERT (Lewis et al. 2019)

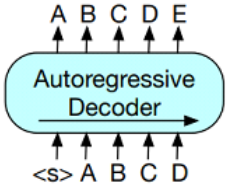

Figure 5. The Auto-Regressive decoder structure of GPT (Lewis et al. 2019)

BART is pre-trained by corrupting documents using a random noise function, such as randomly masking words or randomly deleting words [11]. The corrupted text is then fed into the BART model and optimizes the cross-entropy between the decoder's output and the original document [11]. As a result, the model learns to reconstruct the original text using its sequence-to-sequence autoencoder architecture.

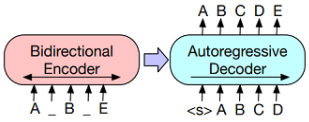

Figure 6. The encoder-decoder structure of BART (Lewis et al. 2019)

BART can be fine-tuned on summarization tasks using a supervised learning approach, where the model is trained on pairs of input texts and their corresponding summaries. During training, the BART model learns to generate summaries that are semantically meaningful and linguistically well-formed. In our model, we fine-tuned BART with the CNN Daily Mail dataset.

There are several summarization models available such as Generative Pretrained Transformer (GPT), Bidirectional Encoder Representations from Transformer (BERT), and Bidirectional and Auto-Regressive Transformers (BART). GPT uses an autoregressive decoder for generating summary output, while BERT uses a bidirectional encoder for extracting summary. BART combines the bidirectional encoder in GPT and the autoregressive decoder from BERT. Therefore, it can generate high-quality summaries and handle long and complex documents so BART is chosen to be implemented in this project.

2.5. BERT fine-tuned with SQuAD 1.0 vs SQuAD 2.0

Bidirectional Encoder Representations from Transformers are a family of masked-language models introduced in 2018 by researchers at Google. BERT utilizes a transformer to learn contextual relations between words in a text. The input embedding in BERT is a combination of the following three types of embeddings:

Token embeddings: Token embedding is the contextual embedding of each token in the sentence. It is initialized using the WordPiece algorithm and then trained in Bert by considering the surrounding context of the token within the sentence [7]. The WordPiece algorithm is applied to the entire training corpus to create a fixed vocabulary of subword units [7]. Each subword unit is assigned an embedding vector. And those vectors are used to initialize the token embeddings in BERT.

Segment embeddings: BERT also learns unique embedding for distinguishing different sentences. As shown in Figure 7, all tokens marked as A belong to the question, and those marked as B belong to the paragraph [7].

Position embeddings: Positional embeddings are used to learn the information of order in the embeddings. As in transformers the information related to the order is missed,

and positional embeddings are used to capture the sequence of words in a sentence [7]. The input embedding of BERT is the sum of Token embeddings, Segment embeddings, and Position embeddings [7].

Figure 7. The input embedding of BERT is the sum of Token embeddings, Segment embeddings, and Position embeddings. [7]

BERT provides two strategies to learn contextual information, Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) as illustrated in Figure 8. In Masked Language Modeling, some words in the text are randomly masked. The context words surrounding a [MASK] token are used to predict the [MASK] word. BERT randomly selects a subset of tokens in each input sequence and replaces them with a special [MASK] token. The model is then trained to predict the original value of the masked tokens based on the surrounding context. Specifically, it is trained to minimize the cross-entropy loss between the predicted probability distribution and the true distribution over the original tokens. The loss function in BERT only considers the prediction of the masked values, the prediction of the non-masked words is ignored [7]. To prevent the model from simply memorizing the masked tokens, BERT uses a mixture of masked and unmasked tokens during training [7]. In particular, it randomly selects 15% of the tokens in each input sequence to mask, 80% of the masked tokens are replaced with [MASK], 10% are replaced with a random token, and the remaining 10% are left unchanged [7]. This enables the model to understand the relationships between words in a sentence.

Next Sentence Prediction allows BERT to understand how different sentences in a text corpus are related to each other. During the training of the BERT model, the sentence pairs are taken as input. It then predicts if the second sentence is the subsequent sentence of the first sentence. The input sequence is constructed by concatenating the two sentences with a special [SEP] token in between them and adding a special [CLS] token at the beginning of the sequence. The output is a single value (0 or 1) indicating whether the second sentence is a consecutive sentence or not. In practice, 50% of inputs are taken such that the second sentence is the subsequent sentence as in the original document, whereas the other 50% comprises the pair where the second sentence is chosen randomly from the document [7]. The Masked Language Modeling and Next Sentence Prediction are trained simultaneously.

The Stanford Question Answering Dataset (SQuAD) consists of questions asked on a set of Wikipedia articles. The answer to each of the questions is either a text segment

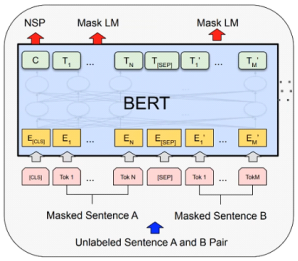

Figure 8. The pre-training procedure of BERT [7]

or a span from the passage. In the fine-tuning phase, BERT extracts words from a question and a passage and combines them as input as shown in Figure 9. It begins with a special token [CLS] indicating the start of a sentence and uses [SEP] to separate the question and passage. BERT also uses segment embeddings to distinguish between the question and the passage containing the answer. Two segment embeddings are created, one for the question and the other for the passage. These embeddings are added to a one-hot representation of tokens to differentiate between the question and the passage. After combining the embedded representation of the question and passage, we feed it as input to the BERT model. The last hidden layer of BERT is modified to use softmax and generate probability distributions for the start and end index of a substring that will form the answer. This substring will be selected from the input text sentence.

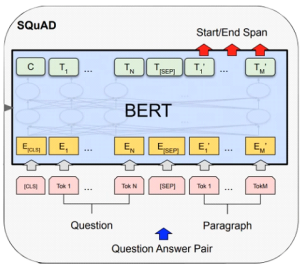

Figure 9. The fine-tuning procedure of BERT using SQuAD dataset [7]

DistilBERT is a compressed version of BERT that requires much lower computational power. The idea behind it is to transfer the knowledge learned by the larger model to the smaller model, making it more efficient and easier to deploy on devices with limited computational resources. DistilBERT achieves it by utilizing a combination of knowledge distillation and parameter pruning techniques. In knowledge distillation, a smaller model is trained to mimic the predictions of a larger, more complex model. In parameter pruning, certain parameters in the model are removed to reduce its size. The resulting DistilBERT model has a much lower computational time.

After researching the models for the question-answering task, it is clear that there are many neural network models such as RNN, LSTM, and CNN. However, the state-of-the-art neural network models are Bidirectional Attention Flow (BiDAF) and Bidirectional Encoder Representations from Transformers (BERT). BERT was used to extract answers from the snippets in this project due to the following reasons:

BERT is pre-trained on large amounts of text data that generates higher-quality contextualized embeddings while BiDAF uses Bidirectional LSTM to generate contextualized embeddings, which can't capture long-term dependencies very well. Moreover, While BiDAF is built on top of several bidirectional LSTMs, BERT is built on top of Transformers. Therefore, BERT has no recurrence architecture and is possible to work parallel.

2.6. Text-to-Speech

Text-to-speech (TTS) is the process of converting written text into spoken words. It helps the program to read out texts using simulated human speech. Text-to-speech involves several steps, including text analysis, linguistic processing, and audio synthesis.

In the text analysis phase, the text is examined and converted to phonetic spelling to determine the appropriate pronunciation, stress, and intonation patterns. After that, the layer of grammar and syntax is applied to ensure the spoken text is understandable. And the model or program generates the spoken text using artificial intelligence or pre-recorded sound databases [12].

There are several APIs available to convert text to speech such as VoiceDream, Wideo, Google Cloud Text-to-Speech, Amazon Polly, and iSpring Suite. Google text to speech is selected for this project as it is free and open source. Google Text-to-Speech is a service offered by Google that uses advanced artificial intelligence and machine learning techniques to generate high-quality spoken text from written text [13]. Google Text-to-Speech supports a wide range of languages and voices. It generates speech that sounds natural and expressive.

2.7. Evaluation metrics

Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP) are adopted to evaluate the document retrieval system of this project. MRR identifies the rank of the first relevant item in the list of recommendations. The reciprocal rank is 1 divided by the rank of the first relevant item. The MRR for the model is the average of the reciprocal ranks for all queries. The MRR formula is calculated as follows:

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{R_i} \tag{2}$$

N is the total number of queries
R_i is the rank of the first relevant item for the i-th query.

MAP takes into account both the rank and precision of relevant items in the list of recommendations for a given query or user. Average Precision (AP) is the average of the precision values at each rank where a relevant item is found.

$$AP = \frac{1}{K}\sum_{k=1}^{K}P(k)\cdot rel(k) \tag{3}$$

K is the number of items in the list of recommendations
$P(k)$ is the precision at rank k
$rel(k)$ returns 1 if the item is relevant, and 0 otherwise.

$$MAP = \frac{1}{N}\sum_{i=1}^{N}AP_i \tag{4}$$

N is the total number of queries or users in the dataset
AP_i is the average precision for the i-th query or user

There are many other metrics for evaluating document retrievals such as precision, recall, and F1 score. However, precision, recall, and F1 score are not adequate metrics for evaluating document retrieval performance as they do not consider the rank of the documents. MRR rewards systems that place relevant items higher in the list of recommendations. MAP considers the precision values at each rank where a relevant document is found and assigns a weight to each precision value based on the rank of the relevant document. Therefore, MRR and MAP are the best choices for evaluating information retrieval.

The Exact Match (EM) and accuracy are adopted to evaluate the final answer. If the words in the predicted answers are exactly matched with the ground truth answer for each pair of questions and answers, then the EM score is 1. Otherwise, it will be 0. Then we calculate the number of correct answers (EM score 1) divided by the total number of predictions to get the accuracy.

Several other metrics are available for evaluating the final answer such as F1 score, and Exact match. Exact Match is a strict criterion that measures the percentage of queries for which the system provides a complete and exact match with the query. And it is particularly useful in the scenario where the system is required to provide a specific answer to a query. Since the goal is to get the answer with the highest accuracy from the system, Exact Match and accuracy are used as evaluation metrics instead of the F1 score. Therefore, the F1 score and Exact Match are the best metrics.

3. Implementation

In this study, we present a closed-domain, extractive, and information retrieval-based question-answering (QA)

system. The framework of the system can be categorized into two primary components: the information retrieval system and the question-answering system. Upon receiving an enquiry, the information retrieval system retrieves the five most relevant snippets of articles from a designated corpus. Subsequently, the question-answering system extracts an appropriate answer from the relevant snippet. The details of implementation are presented step by step in this section.

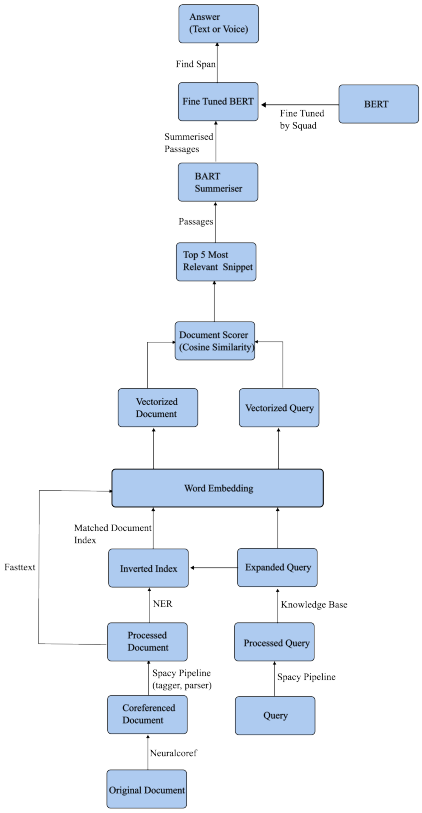

Figure 10. A flowchart describing the mechanism of the question-answering system

3.1. Dataset

The dataset used in this assignment is a subset of the ”COVID-19 Open Research Dataset Challenge (CORD-19)” from Kaggle [2]. CORD-19 is a resource of over 1,000,000 scholarly articles, including over 400,000 with full text, about COVID-19, SARS-CoV-2, and related coronaviruses [2]. This dataset is provided to the global research community to apply recent advances in natural language processing and other AI techniques to generate new insights in support of the ongoing fight against this infectious disease [2]. Due to the insufficient computation power, 10,000 samples are randomly sampled from the original dataset to implement in this assignment. Samples with too short a corpus, less than 100 characters, give no sensible contents, and samples with too long a corpus, more than 100,0000 characters, take an exceptionally long time to process them. Thus, from those 10,000 samples, samples with too short a corpus and too long a corpus are filtered out.

3.2. Pre-processing

The Pre-processing section is composed of coreference and data cleaning. Coreference is to replace all pronouns and representations of one entity with its canonical form. The data cleaning part is to clean unnecessary contents to reduce noise for downstream tasks. The implementation of coreference and data cleaning is illustrated below:

3.2.1 Coreference

The original corpus is processed to replace the pronouns and representations of one entity with its canonical form to enable coreference checking. To illustrate, the sentence ”covid is spreading all over the world, and it is highly infectious” would be resolved to ”covid is spreading all over the world and covid is highly infectious” by following this process.

The NeuralCoref library is employed in conjunction with a Spacy pipeline to accomplish coreference resolution [15]. NeuralCoref is a pre-trained deep neural network that is trained on a vast corpus of annotated data, specifically the OntoNotes Corpus [14]. Given the original corpus as input, NeuralCoref outputs a corpus that has been annotated with possible coreferences in certain words. Then, this annotated corpus passed through the coreference function that replaces all pronouns and entity representations with their canonical forms.

It is important to note that the Spacy pipeline used in the coreference section differs from that used in the rest of the article. It is because NeuralCoref is not a well-maintained library and an outdated version of Spacy was necessary to support it.

3.2.2 Data Cleaning

1. The in-text citations of IEEE and APA styles were attempted to remove. IEEE style is easier to remove because the format is well-maintenance such as [1]. However, in-text citations of the APA style have a lot of variations. Only a simple and frequently appear in-text format is cleaned. A sample is as follows, ”(Peterson et al., 2009)”.

2. The URLs in the corpus were also attempted to remove. Since URLs are not useful in judging the relevance of an article to some topics, any string starting with ”http://” and ”www.” is removed.

3. The non-English characters were also attempted to remove. The Spacy NLP pipeline cannot process non-English characters so they are not helpful in further analysis.

Punctuation and capitalization are not removed in the data cleaning process because they are useful in the latter part for named entity recognition. For example, "U.S." would become "us" without capitalization and punctuation. Also, it cannot be recognized as a named entity.

3.3. Name Entity Extraction

Once the corpus has been coreferenced, it is passed through the Spacy pipeline for entity extraction. The Spacy pipeline is a series of processing steps, including part-of-speech tagging, semantic parsing, and more. Using information from the part-of-speech tags and semantic parses, the pipeline extracts named entities from the corpus.

It is worth noting that the language model employed in this step is en_core_sci_sm [1]. This model has been specifically trained on scientific articles. And it is capable of extracting named entities exhaustively and extensively. As a result, each token in the corpus is linked to a named entity and can be a phrase or a single word. This arrangement will be valuable in later sections, such as word embedding.

3.4. Inverted Index

An inverted index is constructed to enable a quick lookup of articles that mention named entities. Creating an inverse index involves recording all the entities present in the corpus. And also the documents that they appear in. The resulting inverted index is implemented as a dictionary named inverted_index, with the named entity serving as the key and the value being a list of article indices that mention the entity.

3.5. Word Embedding

To capture the semantic meaning of the phrases in the corpus, a Fasttext model named fasttext_model is trained based on the tokens extracted in the previous section, which can be phrases or single words [4]. The input of the model is a list of tokenized sentences. It is trained on a 10-gram window, and the output is a 300-dimensional word vector.

In the vector space generated by the model, words that have similar meanings will be closer to each other than unrelated words. This means that word embedding can effectively capture the semantic meaning of the phrases present in the corpus.

3.6. Knowledge Base (Entity Linking)

The primary purpose of the knowledge base in this project is to establish links between named entities and their aliases or alternative names, such as "Covid" and "Covid-19". For this, a dictionary-based knowledge base is created. In this knowledge base, the key is the canonical form of the named entity, and the value is the corresponding alias.

After extracting the name entities from the corpus, Wikidata is leveraged to find the canonical form of each named entity to perform entity linking [10]. This process enabled the program to link named entities with their aliases, thus enriching our knowledge base. However, Wikidata may not always be up-to-date, and some critical name entities may not exist in the database. For example, "Covid" did not exist in Wikidata, so a part of the knowledge base is manually added to link it to "Covid-19". Thus, it is crucial to check for such scenarios and handle it manually. The knowledge base stores under knowledge_base which contains around 60,000 entity-alias pairs.

3.7. Query-Document Matching

The goal of query-document matching is to identify potential documents in the corpus that match the query. To this end, the entered query is first processed using the preprocessed_query function. Then it is cleaned by removing stop words and punctuation. And named entities are extracted from the query. These extracted named entities are then searched against the knowledge base to retrieve their corresponding aliases that are added to the query. As a result, the changed query contains words with rich contextual meaning and expanded name entities.

Next, the expanded query looks for potential articles by matched_article function. Each token in the query is used to search the inverted index to find the documents that mention it. A collection of article indices that mention the tokens in the query is retrieved. And only those articles that contain all the tokens in the query are considered potentially relevant documents. Then, these potential articles are used for further analysis.

3.8. Document Scorer

The cleaned and expanded query is then passed to the rank function for further processing. The query and the potentially relevant articles are vectorized using the Fasttext model trained in the word embedding section.

The average of the word vectors is taken for each word in the cleaned and expanded query to generate the query mean word vector. The paragraph means word vectors for the articles are generated using the same approach.

Then, the cosine similarity between the query mean word vector and each paragraph mean word vector is calculated for each article. And the resulting cosine similarity score helps to determine how relevant each article paragraph is to the query. The higher the similarity score, the more similar the article paragraph is to the query. And it implies that the content of the article is more relevant to the query.

3.9. Summarisation

Before extracting answers from the five snippets, a summarization of the five snippets using BART is done to further increase the accuracy. The input is a piece of string formed by concatenating five snippets. The output is the

summary of those snippets. The minimum and maximum length of the summary is 30 and 250 characters respectively. The pre-trained BART is fine-tuned with the CNN Daily Mail dataset. The CNN Daily Mail is a large collection of text-summary pairs, which is commonly used for text summarization tasks in natural language processing (NLP). This dataset is comprised of news articles from the CNN and Daily Mail websites. On top of that, it also contains corresponding summaries created by professional journalists. As a result, BART gives a high-quality summary of the snippets. A credit is given that the BART used is fine-tuned by its creator from hugging face before implemented in this project [8].

3.10. Bert Models fine-tuned with SQuAD 1.0 vs SQuAD 2.0

The pre-trained BERT and DistilBERT models were downloaded from hugging face [9]. Then, the model was fine-tuned with the SQuAD dataset. The SQuAD dataset contains questions and paragraphs as inputs and contains answers and answer start/end indexes as labels. We fine-tune the model with a learning rate 2e-5, and a batch size of 16 for 3 epochs. The training minimizes the cross-entropy loss between the output and the labels. There are two versions, SQuAD 1.0 and SQuAD 2.0. Squad 1.0 contains over 100,000 question-answers from Wikipedia articles while SQuAD 2.0 contains 140,000 question-answer pairs. SQuad 2.0 also has 50,000 unanswered questions to cater to situations where no answers to questions are supported by a paragraph for a question. Due to computational power limitation, we sample 20000 question-answer pairs from both SQuAD 1.0 and SQuAD 2.0. We fine-tune BERT and DistilBERT models with both SQuAD 1.0 and SQuAD 2.0 and compare their performance on 5 questions. The result will be shown in the experiment session.

3.11. Question-Answering

After fine-tuning the BERT model using the SQuAD dataset for question-answering tasks and obtaining a summarised snippet from a pre-trained summarization model, this snippet is subsequently passed as an input to the ”question_answering” function. The fine-tuned BERT model identifies the text span most likely to contain the correct answer within the snippet. The output of this function comprises the extracted answer, the corresponding answer span, and the softmax probability score associated with the identified span. This method demonstrates the potential for integrating pre-trained models in a manner that effectively addresses complex question-answering tasks.

3.12. Output utility and Text-to-Speech

The Ipython widgets are utilized to make the output utility of the program. The interface has two checkboxes for the sound and paper details, a text field to insert the query, and a Submit button which can be seen in Figure 11. These checkboxes are added to provide more control to the user about the output format. Upon a query submission, the user can select if the program will search for the relevant documents and from those files return the most relevant answer.

The query and returned answer will be converted to sound utilizing Google Text-to-Speech API. gTTS generates sound from those texts by analyzing the text, performing linguistic processing, and audio synthesis on it. The generated audio file is saved in an mp3 format and is autoplayed using threading.

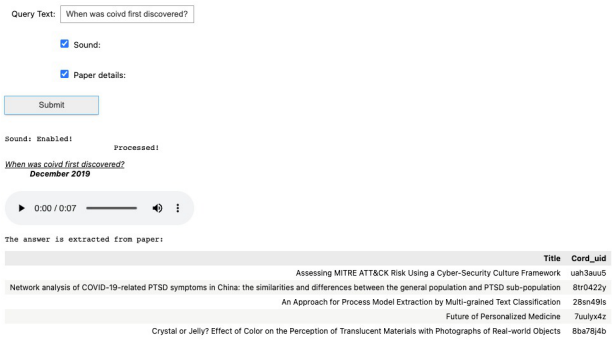

Figure 11. User Interface

4. Experiments and Results

4.1. Information Retrieval Comparision of Assignment 2 and Assignment 3

The aim of this subsection is to evaluate the Information Retrieval (IR) performance in Assignments 2 and 3. The information retrieval system underwent several improvements in Assignment 3 based on the findings from Assignment 2. The primary enhancements include:

- Coreference resolution (replacing pronouns/representation to its canonical form)
- Knowledge expansion (from 10 to 60,000 entity linkages)
- Query pre-processing (extraction of crucial words)
- Fasttext word embedding (consider sub-word)

More details on how the improvement works are presented in Methodology.

Ten questions designed to test and compare the IR system in Assignment 2 and Assignment 3 are shown in Table 1. A comparison of their IR performance of them in Mean Reciprocal Rank (MRR) and Mean Average Precision(MAP) metric is summarized in Table 2 and Table 3 respectively. Figure 12 and 13 shows a comparison of their

performance in bar plots. Their performance is evaluated based on the MRR and MAP. While MRR only considers the rank of the first relevant article, MAP considers the relevance of all the ranked articles. Since relevant articles should be at the highest rank, the rank of the first relevant article is important for an IR system. Thus, MRR is a good metric to measure it. MAP gives a comprehensive picture of how the IR performs across all the ranks. The MRR and MAP for the IR system in Assignment 2 are 0.303 and 0.147, while the MRR for Assignment 3 is 0.733 and 0.393. Both metrics indicate that the IR performance in Assignment 3 is superior to that of Assignment 2. IR in Assignment 3 can retrieve relevant articles in a higher rank and it can retrieve more relevant articles than IR in Assignment. These performance differences can be attributed to the coreference resolution, which clarifies ambiguity, the expansion of the knowledge base that enables the retrieval of more relevant articles, and the advanced word embedding that captures greater semantic meaning.

Some questions demonstrate that the IR system in Assignment 3 performs significantly better than its counterpart in Assignment 2. For instance, the IR system in Assignment 2 could not find any relevant article for question 4, while the IR system in Assignment 3 identified the pertinent article with the highest rank. The articles retrieved by the Assignment 2 system mostly presented death figures rather than the cause of death from COVID-19. In contrast, the IR system in Assignment 3 accurately presented the cause of death. One potential reason for this difference could be the improved query pre-processing, which retains important words in the query, while the query pre-processing in Assignment 2 discards words like "cause" in question 4. Consequently, the IR system in Assignment 3 can identify the cause, while the IR system in Assignment 2 fails to do so.

Nonetheless, neither IR system could find relevant articles for question 1. For question 1, both IR systems were unable to present relevant articles that included the number of infection cases in 2022. However, the IR system in Assignment 3 was able to find the number of infection cases in 2020. This discrepancy may be associated with insufficient data, leading the IR system in Assignment 3 to select the most similar snippet while the IR system in Assignment 2 is not proficient enough to find a comparable snippet.

Questions 6 and 7 were intentionally designed to be similar to confuse the IR system. Question 6 asks about the origin of COVID-19, while question 7 asks about the location of the origin. The articles retrieved from question 6 should be about the virus, while those retrieved from question 7 should be about a location. However, during query pre-processing, "What" and "Where" are removed, resulting in the same set of articles being retrieved for both questions. Although the MRR and MAP for these questions are reasonable, the retrieved articles do not accurately address the needs of the question.

Table 1. A list of query evaluate the performance of systems

No.	Query for system performance evaluation
1	How many people have been infected by covid in 2022?
2	What are the pathways for COVID-19 to spread?
3	What are the symptoms of infecting covid?
4	What causes death from Covid-19?
5	What is social distancing?
6	Where is the origin of covid?
7	What is the origin of covid?
8	What kind of virus is Covid?
9	What types of rapid testing for Covid-19 have been developed?
10	When was covid first discovered?

Table 2. The performance of IR in Assignment 2 and 3 in MRR metric

Query	MR(Assignment 2)	MR(Assignment 3)
1	0	0
2	0.333	1
3	0.5	0.333
4	0	1
5	1	1
6	0	0.5
7	0.2	1
8	1	1
9	0	1
10	0	0.5
MRR	0.303	0.733

It is important to note that the execution time for Information Retrieval (IR) in Assignment 3 is significantly longer than that in Assignment 2. The retrieval process for all 10 questions in Assignment 2 took 12.6 seconds, while it required 38.2 seconds for IR in Assignment 3, increasing by approximately 203%. This extended execution time can be attributed to the expansion of the knowledge base. As the knowledge base grows, more potential articles are included in the query-document matching process, leading to a larger number of articles and their respective snippets being evaluated by the document scorer. Consequently, the time required for the IR process increases substantially in Assignment 3.

4.2. BERT models performance comparison

In this experiment, the performance of BERT and DistilBERT fine-tuned with SQuAD 1.0 and SQuAD 2.0 is compared by using 5 queries. We calculate the number of Exact

Table 3. The performance of IR in Assignment 2 and 3 in MAP metric

Query	AP(Assignment 2)	AP(Assignment 3)
1	0	0
2	0.287	0.2
3	0.180	0.287
4	0	0.760
5	0.2	0.333
6	0	0.320
7	0.04	0.483
8	0.760	0.4
9	0	0.6
10	0	0.543
MAP	0.147	0.393

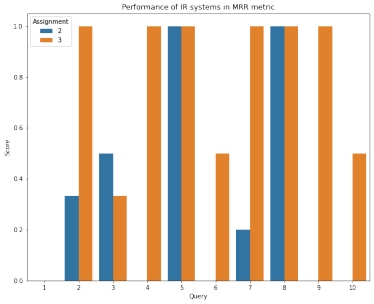

Figure 12. A barplot showing the performance of IR systems in MRR metric

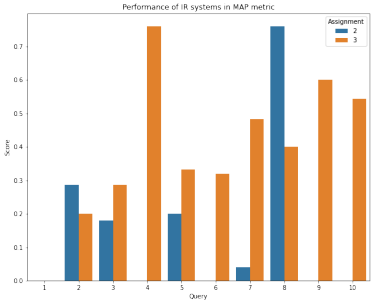

Figure 13. A barplot showing the performance of IR systems in MAP metric

matches between the prediction and the ground truth. Then, the accuracy is the number of Exact Match divided by the total number of queries. Table 4 and 5 display the queries used for the test and the accuracy of the BERT models respectively. Figure 12 shows the accuracy of BERT models against 5 test queries.

We also compare the computational time of BERT and

Table 4. The 5 queries for evaluating different BERT models

No.	Query for BERT models comparison
1	What are the symptoms of infecting covid?
2	What causes death from Covid-19?
3	Where is the origin of covid?
4	What is the origin of covid?
5	When was covid first discovered?

Table 5. The number of Exact Match and accuracy comparison for different BERT models

BERT models accuracy comparison		
Models	Number of Exact Match	Accuracy
BERT SQuAD 1.0	3	0.6
BERT SQuAD 2.0	4	0.8
DistilBERT SQuAD 1.0	2	0.4
DistilBERT SQuAD 2.0	2	0.4

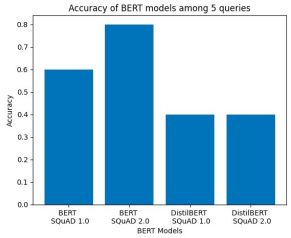

Figure 14. The accuracy of BERT models among 5 queries

DistilBERT fine-tuned with the SQuAD 1.0 and SQuAD 2.0 dataset. It can be clearly seen from Figures 12 and 13 that BERT SQuAD 2.0 has the highest accuracy (0.8), but the longest computational time (53.18). Although the DistilBERT models have a shorter computational time, the accuracy is much lower than BERT models. The performance of BERT SQuAD 1.0 is medium. Therefore, the best BERT model is BERT SQuAD 2.0.

4.3. Question-Answering Result

After extracting the top 5 documents, they are concatenated and passed through a summarizer. The resulting summary is input to BERT SQuAD 2.0 to output the final answer. The final answers for the 10 queries are compared with the ground truth answers. There are 5 out of 10 answers that exactly match the ground truth answer. The accuracy of our system is 50%. Figure 16 shows some of the queries and answers from our QA system. For question 6

Table 6. The fine-tuning execution time comparison of BERT models

BERT models Execution Time Comparison	
Models	Execution Time
BERT SQuAD 1.0	51.03
BERT SQuAD 2.0	53.18
DistilBERT SQuAD 1.0	30.77
DistilBERT SQuAD 2.0	36.95

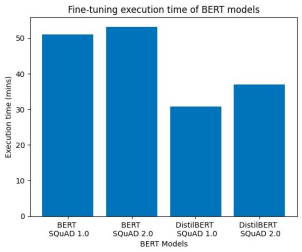

Figure 15. The fine-tuning execution time of BERT models

(Where is the origin of covid?) and question 7 (What is the origin of covid?), the outputs are (Wuhan, China) and (SARS-CoV-2) respectively. This shows that the system can successfully distinguish between (Where) and (What) in the queries and extract the corresponding answer. For question 3 (What are the symptoms of infecting covid?), the output is (Cough (68%), fever/chills), while the ground truth answer is (Cough (68%), fever/chills (58%), and shortness of breath (37%)). The system can successfully extract part of the answer. However, since we only count the Exact Match between prediction and ground truth, this output is classified as the wrong answer. For question 1 (How many people have been infected by covid in 2022?), the output is (59.7 million). The output is a wrong answer, but it is the number of people infected by covid in 2020. The wrong answer is due to the lack of relevant data in 2022 in the dataset for this project. But the model can extract the number of people infected by covid.

5. Code

Please visit our github repository for the coding files. https://github.com/fuchuenli/NLPAssignment3.git

6. Conclusion

6.1. Learnings

Using the whole document embedding to find the relevant documents will provide a list of documents that have

```
question6 = "Where is the origin of covid?"
'answer': 'Wuhan, China'
question7 = "What is the origin of covid?"
'answer': 'SARS-CoV-2'
question3 = "What are the symptoms of infecting covid?"
'answer': 'Cough (68%), fever/chills'
question1 = "How many people have been infected by covid in 2022?"
'answer': '59.7 million'
```

Figure 16. Some of the queries and answers from our QA system

similar semantics/scores to the query. In these cases, the document will be irrelevant as those words might not be together in the document. In order to solve this problem, an average fragment embedding of 10 sentences was used to get the document embedding, however, the relevant fragment ended up with partial information regarding the query. Since each passage has information about a topic in English writing, splitting the document into fragments of 10 sentences will divide the information about the topic into several fragments. And all of those fragments might not have a high cosine similarity score. Thus, each document was split into passages using an NLP passage tokenizer.

Fasttext is a good word embedding technique to generate word embedding as it considers both words and subwords. Thus, Fasttext can handle out-of-vocabulary and rare words. On comparing Fasttext to Word2Vec, FastText has shown better performance on similarity tasks. For example, (covid↔Covid-19) is closer in Fasttext word embedding than Word2Vec word embedding in the vector space. However, Fasttext is much more computational expensive. it took 6 hours to train the respective model while Word2Vec only took 11 minutes.

The main task of this project is to answer a query. Due to the limited corpus, it is clear that the top relevant snippets might not have a relevant answer for each query. Thus, the top 5 snippets were summarized using Bart to increase the chance of giving the relevant answer to a query. And that summary was utilized by Bert to answer the question.

The SQuAD dataset is a popular choice to fine-tune the Bert model to learn how to answer questions about the text. This Fine-tuned model can then be used to answer questions on new unseen text. This combination of the BERT and SQuaD datasets improves the ability of BERT to handle new or unseen questions.

Threading is a technique that allows multiple threads to run concurrently within a single process. The implementation of the threading function on Jupyter Notebook enables it to execute multiple sub-process on different threads. And this way program can handle multiple tasks smoothly on Jupyter Notebook [6].

6.2. Limitations

Since the project is a closed domain QA and the dataset of the project only limited to 10000 articles due to computational limitation, it is only possible to return the relevant answer for a query if the extracted dataset has information related to that query.

A critical bottleneck in enhancing the QA system is the performance of the Named Entity Recognition (NER) component. The NER tool utilized in this study is SciSpacy, a language model tailored for scientific articles. Although it can exhaustively extract named entities, the accuracy of the extracted entities may be suboptimal. NER plays a crucial role in building the inverted index and preprocessing queries, serving as the foundation for subsequent tasks. Consequently, a high-quality NER is indispensable for ensuring the proper functioning of downstream processes.

The performance of the IR system, as demonstrated in Query 6 and 7 of Table 1, reveals that the current query-document algorithm output documents related to both "Where" and "What". The output from IR system is not specific enough regarding to the query. However, since we use BERT to extract the answer from the summary of the related documents, we can accurately output the final correct answer. Our system still require a better query-document algorithm to better address factoid questions.

Our BERT models has limitation on the span of the answer. They can only output one span as the answer. If the ground truth answer consists of serveral words existing in different word spans of the summary, BERT can only choose the most relevant word span as the answer. This may due to the architacture of BERT and SQuAD dataset as SQuAD dataset has only one word span as label for fine-tuning.

This project utilizes the free service gTTS by Google. Thus, the sound option for an answer has the following limitations:

- For each execution, the query and answer combined can only have a maximum text length of 5,000 characters.
- This project can only convert only 1,000 answers to audio per day.

6.3. Challenges

A primary challenge in this study was effectively performing coreference resolution. Coreference is a vital task, as it clarifies subjects and objects in sentences, potentially improving the performance of subsequent tasks. However, coreference resolution is inherently complex, encompassing cataphora, demonstrative pronouns, and other linguistic phenomena beyond anaphora. Several rule-based and neural network-based approaches were employed, but none of them yielded satisfactory results or required excessively long processing times. Ultimately, the neural network-based tool, NeuralCoref, was utilized. Although not perfect, NeuralCoref efficiently handles coreference resolution, including cataphora, within a reasonable timeframe.

The snippets in the Assignment 2 are composed by 10 consecutive sentences. The 10 sentences extracted may not represent the whole contents or one idea may divided into multiple snippets. The relevant information is hard to retrieve. Therefore, a paragraph is chosen to be a snippet instead of 10 sentences, so the idea will be complete and consistent.

Another challenge was to expand the Knowledge Base. Manual expansion of the Knowledge Base was a tedious job that require a lot of time and it is also impractical to compare over 2 millions name entities across 10000 articles. On the other hand, using per-trained model alone to connect the name entities between articles are extremely time-consuming.As a result, an approach to look up Wikidata to find corresponding name entity to their canonical form is implemented which described in Implementation section. Though this method does not create a knowledge base as exhaustive as using pre-trained model but it substantial reduced the execution time.

Before adopting BART summarizer, only the top 1 ranked document snippet is used to extract the final answer. The accuracy of the answer heavily depends on the accuracy of IR system. Therefore, we adopt BART to summarize the top 5 relevant snippets. This increases the chance to extract the correct answer and lower the strictly accurate requirement for IR system.

In this project, the answer and query are converted to a single sound file utilizing Google Text to Speech. However, the audio file was not being played in the Jupyter even when it is in autoplay mode. On analyzing the code and Jupyter Notebook, the problem was due to Jupyter Notebook running a single cell at a given time. After researching ways to tackle it, threading seemed like a good option. Incorporating the threading function into the program, the program became responsive enough to handle the new query even when the audio clip is being played.

7. Future works

As outlined in the limitations section, the NER performance is a bottleneck in the project due to its imperfect extraction of named entities. To address this issue, a pre-trained model such as BERT, fine-tuned with a customized entity extraction dataset from CORD-19, could potentially replace the current NER tool. Since the fine-tuned pre-trained model is tailored to a COVID-19 dataset, the extraction of named entities is expected to be more accurate, thereby enhancing overall system performance.

In the answer extraction part, we can adopt variants of BERT specifically for medical text answer extraction such

as BioBERT. BioBERT has been pre-trained on a large corpus of biomedical text. Then, we can fine-tuned BioBERT with Cord-19 dataset. This can improve the accuracy of answer extraction in COVID-19 related context.

References

[1] AI2. scispacy. https://allenai.github.io/scispacy/, Nov. 2022. [Online; Accessed: Apr. 18, 2023]. 7

[2] ALLEN INSTITUTE FOR AI AND 8 COLLABORATORS. Covid-19 open research dataset challenge (cord-19). https://www.kaggle.com/datasets/allen-institute-for-ai/CORD-19-research-challenge, Mar. 13 2020. [Online; Accessed: Apr. 01, 2023]. 6

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. [Accessed: Apr. 06, 2023]. 2

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. [Accessed: Apr. 06, 2023]. 7

[5] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370, December 2016. [Accessed: Apr. 06, 2023]. 2

[6] DataNoon. Multithreading in python. https://datanoon.com/blog/multithreading_in_python/, Nov. 2018. [Online; Accessed: Apr. 06, 2023]. 11

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. [Accessed: Apr. 06, 2023]. 3, 4

[8] Hugging Face. facebook/bart-large-cnn · hugging face. https://huggingface.co/facebook/bart-large-cnn, n.d. [Online; Accessed: Apr. 06, 2023]. 8

[9] Hugging Face. Question answering — transformers documentation. Technical report, Hugging Face, n.d. [Online; Accessed: Apr. 18, 2023]. 8

[10] E. Gerber. spacy-entity-linker: Linked entity pipeline for spacy. https://pypi.org/project/spacy-entity-linker/, n.d. [Online; Accessed: Apr. 18, 2023]. 7

[11] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, ..., and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019. [Accessed: Apr. 06, 2023]. 3

[12] A. Scott. Explain the working of text-to-speech solutions. https://www.datasciencecentral.com/a-guide-to-how-text-to-speech-works/, Aug. 2022. [Online; Accessed Apr. 6, 2023]. 5

[13] Read the Docs. Module (gtts) – gtts documentation. https://gtts.readthedocs.io/en/latest/module.html#module-gtts.tts, n.d. [Online; Accessed: Apr. 06, 2023]. 5

[14] Thomas Wolf. State-of-the-art neural coreference resolution for chatbots. *Medium*, Sep. 02 2020. [Online; Accessed: Apr. 06, 2023]. 1, 6

[15] T. Wolf and S. V. Landeghem. Neuralcoref 4.0: Coreference resolution in spacy with neural networks. https://github.com/huggingface/neuralcoref, Jan. 2022. [Online; Accessed: Apr. 18, 2023]. 1, 6