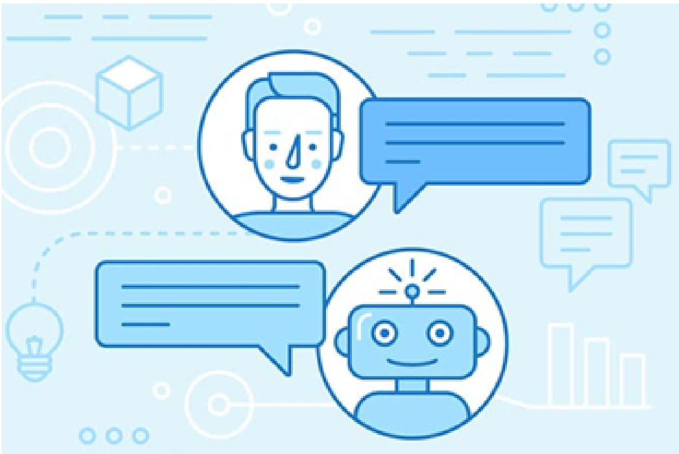
Simplifying Social Conversations: Building a Digital Assistant Using NLP



Pablo De Dios Aguilar
3rd Year Project Final Report
UCL Department of Electronic and Electrical Engineering
Supervisor: Professor Miguel Rio

Contents

Note: This report contains 33 pages and 7400 words (excluding this page and the appendices).

Abstract

This paper discusses the creation of a system aimed at having *"social conversations"* with users. The architecture of the system is explained, which consists of an Understanding model (capable of classifying input sentences into *"objective"* or *"subjective"* intention), Objective Response *Retrieval* and Subjective Response *Generation* models. All the different parts were combined through the usage of speech-to-text and text-to-speech libraries and the context of the conversation was tracked.

This simple system was concluded to allow for casual conversations to be had and some future improvements have been identified (e.g., expanding the objective/subjective dataset, retraining the subjective response generation model on a different dataset to attempt to reach a higher accuracy and treating long and open responses to objective questions separately).

It is believed this approach has showcased the possibility to simplify these kinds of conversations, which can make this field more approachable for future researchers.

1 Introduction and Literature Review

1.1 Context and Goal of the Project

Ever since the creation of the first computers, scientists and engineers have been trying to simplify human-computer interactions. These, which initially relied on the usage of different commands on a command-line interface (CLI), have evolved into what we have today: a graphical user interface (GUI) with a series of icons and a cursor that allows users to navigate through them.

One of the fields which has been greatly explored, is the usage of voice to communicate with computers. In the 1960s, the *IBM Shoebox*, which could understand digits from 0 to 9 and could perform operations between them [1], became the first attempt at creating a voice assistant. In 1990, the first speech recognition product for consumers came out: *Dragon dictate*. It was a dictation system that required users to stop between spoken words (this problem was fixed in 1997, when the same company introduced *continuous* speech recognition) [2]. More recently, digital assistant products such as *Apple Siri*, *Google Now*, *Cortana* and *Amazon Alexa*, whose first versions were launched in the early 2010s [3], have showcased a wide range of abilities from understanding and performing commands, to *replying* to users. These systems have a *"task-oriented"* approach, which means that they are constantly seeking an instruction within the user utterance which will reveal the purpose of the interaction.

Despite how far voice recognition and assistance has come, there is still one glaring flaw: these systems cannot be used to have a *"social conversation"* (i.e., conversations that more closely resemble the typical human-human interaction, like small talk). This type of conversation does not necessarily have a specific goal, other than passing the time. This project will be focusing on creating a system which is capable of having these "social/casual conversations", which can be used to help old people living alone (i.e., digital companion), to aid in children education (i.e., digital educator), etc.

1.2 Related Work

1.2.1 Casual Conversations

The first thing that must be looked into is what the structure of "causal conversations" is like. An in depth examination of these conversations is conducted in [4], where the main stages of a conversation are identified (greeting, address, identification, approach, centering, leave-taking and goodbye) and the relationships between them are explained. Taking inspiration from this journal, [5] explains the difference between "chat" (interactive stage of the conversation) and "chunk" (part of the conversation in which there is a dominant speaker and the rest of the people involved are listening).

These sources have attempted to define the main features and structure of social conversations. They suggest that there are multiple stages to be considered when creating conversational systems. Furthermore, datasets like SWBD-DAMSL [6] and Daily Dialog [7], which present conversations that have been annotated to include information about the specific user intention for each of the sentences, provide an opportunity to create complex conversational systems that analyze multiple conversational factors in order to generate the most appropriate response.

Nevertheless, a system that classifies utteances into different stages/intentions would require multiple classifiers to operate. The conversational system presented in this report will follow a simpler classification process (i.e., objective or subjective). It is postulated that, if a sufficiently large dataset were used to train an advanced response generation model, it would be exposed to enough conversations to be able to learn about different intentions and stages throughout the training process. While a response generation model that is advanced enough to learn all of these features may not exist in the present day, NLP is considered a relatively new field which means there is room for improvement.

1.2.2 Question-Answering and Response Generation

In order to answer objective questions, the BERT model (it will be explained further in section 3.4) was tested on the Stanford Question Answering Dataset (SQUAD) [8]. This dataset provides a paragraph (context) with a set of questions and answers related to it. The experiments on this report, which obtained an accuracy of 86.3% on the test set, led to the decision to use a pre-trained BERT model to answer any objective question the user may have. Nevertheless, a piece of code had to be created to extract the context from google search results, since [8] required the context to be given directly.

On the other hand, the Natural Language Processing with Attention Models course [9] trained a reformer model on the MultiWOZ dataset (i.e., task-oriented conversations). While this course is not free, the methodology followed was recreated at [10] (67.15% accuracy obtained). The code that was implemented in this project has a high influence from these sources, even if a different dataset was applied.

1.2.3 High Level Architecture of a Conversational System

Conversational systems can have different configurations:

- A rule-based approach (i.e., the system takes into account hand-crafted rules for analyzing incoming sentences and generating reponses).
- A retrieval-based configuration, where the response for a particular sentence is obtained by taking the top result (i.e., analyzing similarity) from a database containing response-answer pairs.
- A system which is entirely based on NLP techniques (i.e., employing artificial intelligence both for utterance analysis and generation).

[11] not only discusses the structure of a conversational system based entirely on NLP (which is split into three stages: utterance understanding, dialogue control and utterance generation), but it also compares its "naturalness" to the one achieved using retrieval-based and rule-based systems. In that paper, it was concluded that the NLP system was considered more natural than the retrieval-based one. However, it did not reach the same level as the rule-based approach.

Before developing our system, its high-level architecture must be designed. The one defined on [11] was used as reference to construct our own architecture (i.e., the main three parts of the system defined on that paper were considered for our first end-to-end version). However, the mentioned report was elaborated by numerous people, which allowed the proposed system to use a wide range of models. For my project, due to the fact that it is individual, I will not be able to experiment with that many diverse models.

1.2.4 Information Searching

Sometimes, the user asks an open-domain question for which the system will require to search information before it is able to properly answer. [12] has proposed a system which is able to tackle these situations. The architecture of said system has: a conversational query rewriting transformer, passage re-ranking transformer and abstractive search-answer generation transformer.

This report is relevant to my project because my digital assistant must be capable of answering open-domain questions asked by the user. However, the approach that I ended up using is simpler, as I utilized google search to extract information about the question asked and then a pre-trained BERT model will be able to extract the answer. I obtained positive results using this approach, even if it is not as advanced as the one presented in [12]. Nevertheless, the simplicity of my approach is one of its advantages.

2 Goals and Objectives

1. To design a Sentence Classification Subsystem

    a. To define the different types of potential user intentions.

    b. To find a data set/s that contain sentences labelled with the defined user intentions. If not possible, it might be necessary to modify existing sets.

    c. To create and train a model using the found data set/s. This model will be utilized to determine the likelihood that a sentence has a specific intention.

2. To design a Response Generation Subsystem

    a. To find a variety of data sets that have records of human-human or human-machine interactions, so that the responses can be modelled after them.

    b. To produce different models that generate a response based on the user intention. Depending on the specific sentence produced by the user, one of the models will be used.

3. To combine both subsystems together

    a. To integrate a speech-to-text model so that the the user speech gets converted to text.

    b. To convert the generated text responses back to speech using a text-to-speech model.

    c. To keep track of the context to help the response generation subsystem to be more accurate.

Please note: An additional subsystem was defined in the project proposal. It was believed that multiple responses would have been generated and ranked according to coherency with respect to the context (I referred to it as the Decision subsystem). However, it has been concluded that only one response will be generated, hence the Decision subsystem as it had been defined no longer made sense.

3 Theoretical Background

3.1 Embedding Layers

3.1.1 Introduction

Note: All information presented in section 3.1.1 has been extracted from [13].

Machine Learning models cannot process text directly. A vectorization step must be carried out in the pre-processing stage. One of the most basic approaches is one-hot encoding, in which each word will be represented by a vector that contains a one in the index corresponding to that word, and zeros for all other indices. The length of this vector will be equal to the number of words in the vocabulary. This approach could be utilized to represent words and even sentences (e.g., by concatenating the vectors of each word that appears on a sentence).

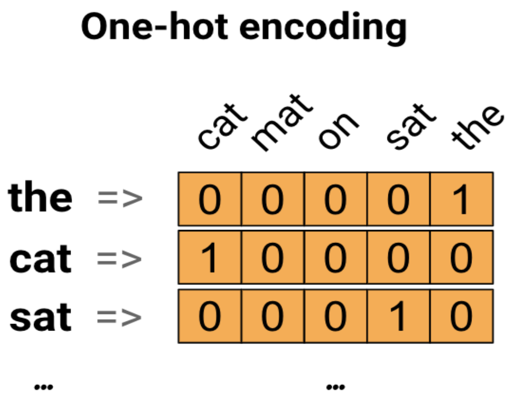


Figure 1: One-Hot Encoding of Different Words, extracted from [13]

Nevertheless, one-hot encoding does not provide information about the meaning of words. In order to obtain that information, word embeddings will be introduced. They are dense vectors of floating values that contain a deeper understanding of what each word means, because words that are alike will have similar embedding values. Obtaining the embedding vector for a particular word is done by using a pre-trained model. The length of these vectors varies depending on the size of the dataset on which the model was trained.

3.1.2 Learning the Meaning of Words

Note: All information displayed in section 3.1.2 has been extracted from [14].

There are different vector embedding learning methods (e.g., Skipgram Negative Sampling, GloVe). The one that was chosen for our project is Swivel.

Swivel, along with most of these methods, relies on a metric named Point-wise Mutual Information (PMI), which is used to measure the association of two different events:

$$\log \frac{P(i,j)}{P(i)P(j)} \tag{1}$$

When applied to language, this metric can be estimated as follows (where x_{ij} is the amount of times that the focus word i and the context word j co-occur, $x_{i*} = \sum_j x_{ij}$ is the number of instances that the focus word i is present in the text, $x_{*j} = \sum_i x_{ij}$ is the amount of times the context word j is present in the text, and $|D| = \sum_{i,j} x_{ij}$ is the overall number of co-occurrences).

$$\log \frac{x_{ij}|D|}{x_{i*}x_{*j}} \tag{2}$$

For the training process, X is initially constructed (i.e., the co-occurrence matrix, which contains all the x_{ij} values sorted in descending frequency order). Once it has been constructed, it will be divided into different shards. Once those shards have been created, an iterative training process will begin: A specific shard will be randomly chosen, the row and column vectors for that shard will be used to estimate the PMI value, which will then be compared to the observed PMI value for that shard. The error obtained will be used to compute the new gradient values (i.e., using a gradient descent algorithm).

This process can be better understood by looking at Figures 2 and 3:

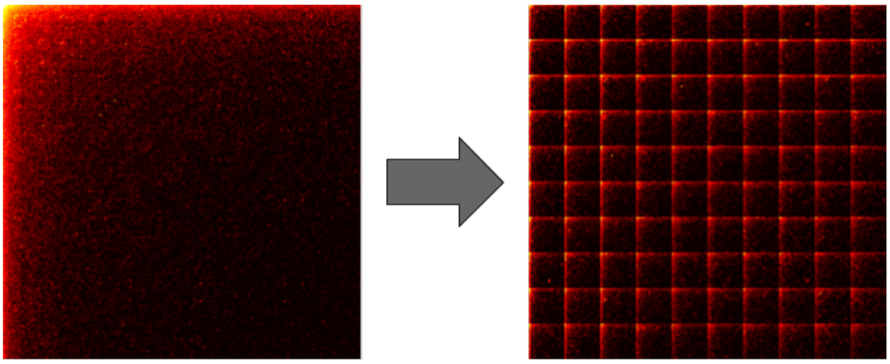


Figure 2: "The matrix re-organization process creates shards with a mixture of common and rare features, which naturally leads to a mixture of large and small co-occurrence values (brighter and darker pixels, respectively)." [14]

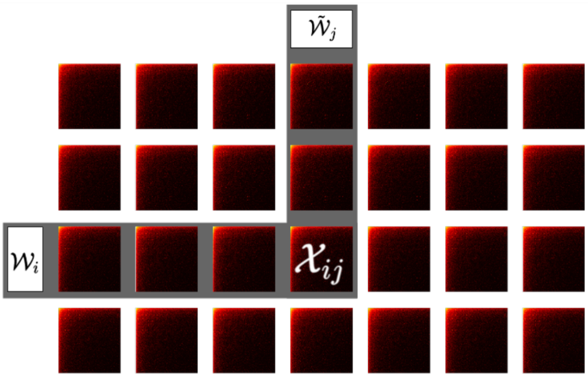


Figure 3: "A shard X_{ij} is selected for training. The corresponding row vectors W_i and column vectors $\tilde{W}_j$ are multiplied to produce estimates that are compared to the observed PMI derived from the count statistics. Error is computed and back-propagated." [14]

Additionally, Swivel distinguishes between two different kinds of unobserved co-occurrences. If both words are very rare then the lack of co-occurrence is not significant, whereas if both words are common, the fact that they are never observed together is significant.

Through this process (which allows for parallel computation due to the multiple shards generation), the model learns the embedding of different words by observing their association with other words.

3.2 Transformer

3.2.1 Introduction

Simple Machine Learning models are given a set of data points from which the model will make a prediction. However, working with language is more complex because of the context. Any conversation has an underlying context that humans are aware of and can reference at various points throughout the conversation.

Therefore, when text is being generated, the model must be aware of the context so that it can reference previous words that are relevant to the newly generated word. Network architectures like Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) can allude to previous words but have a limited context window.

Transformers, introduced in [15], have a significant advantage over these other models, because they do not have limitations when it comes to the amount of context they can consider for generating new words. These models are based on a concept known as self-attention.

3.2.2 Self-Attention

Note: The information displayed on sections 3.2.2 and 3.2.3 has been obtained through a combination of sources [15] and [16].

Self-attention can be defined as a process that determines how relevant each word in a sequence is with respect to other words. The specific attention discussed is called scaled dot-product attention (displayed on figure 4). This type of attention utilizes three different vectors: queries (Q), keys (K) and values (V), which are all derived from the input sequence. While figure 4 displays an individual attention layer, multiple of these layers can be run in parallel to create what is known as Multi-Headed Attention (this kind of attention layer is what a transformer utilizes).

Scaled Dot-Product Attention

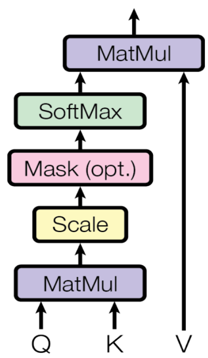


Figure 4: Scaled Dot-Product Attention, extracted from [15]

To illustrate how attention works, let ”Hello, I love you” be an input sequence that will be analyzed using self-attention. The computed matrix representing the attention weights of the input sequence is displayed on figure 5. This matrix presents probability values between the different words in the sequence. These scores indicate how relevant a specific word is with respect to other words.

Self-attention
Probability score matrix

	Hello	I	love	you
Hello	0.8	0.1	0.05	0.05
I	0.1	0.6	0.2	0.1
love	0.05	0.2	0.65	0.1
you	0.2	0.1	0.1	0.6

Figure 5: Example Self-Attention Matrix for the Sentence ”Hello, I love you”, extracted from [17]

3.2.3 Transformer Architecture

Once self-attention has been introduced, the transformer architecture can be defined (displayed on figure 6). The transformer architecture has two main parts: encoder and decoder (indicated on figure 6 with colors red and blue, respectively).

The encoder process can be broken down into different stages:

1. Input Embedding: The encoder will begin by using an embedding layer that will provide information about the meaning of the words in the input, as previously discussed in section 3.1.

2. Positional Encoding: Information about the position of the words within the sentence will be introduced into the word embeddings.

3. Multi-Headed Attention: After the input sequence has been embedded, it will go through an attention layer to compute the attention weights for the input words

4. Addition-Normalization & Feed Forward: Lastly, the transformer has a couple of Add-Norm layers with a Feed Forward layer in between. Firstly, the addition layers, which are defined as residual connections (i.e., both the input and the output of the previous layer are added together), help the training process (direct flow of gradients through the network). Secondly, the normalization layers are aimed at model stabilization. Lastly, the Feed Forward layer is mainly used for further processing.

On the other hand, the decoder utilizes similar layers to the encoder. To avoid repetition, the decoder will not be analyzed layer by layer. Nevertheless, two main things about its functionality will be explained:

- The first multi-headed attention layer does not operate on the input sequence, but rather the output one.
- The second multi-headed attention layer takes the output of the encoder as queries and keys, while the values are extracted from the output of the first multi-headed layer. This layer is paramount for the overall transformer's operation, as it is responsible for determining which of the input sequence words should be referenced when generating a response.
- The final combination of linear and softmax generates the probabilities for each individual word in the vocabulary. Let's say our system has a vocabulary with 10000 words, the final output of the decoder would produce values between 0 and 1 (i.e., probabilities) for each of the those words. Then, the word with the highest probability will be the one predicted by the output (i.e., the transformer generates one word at a time until it predicts the end of the output sequence).

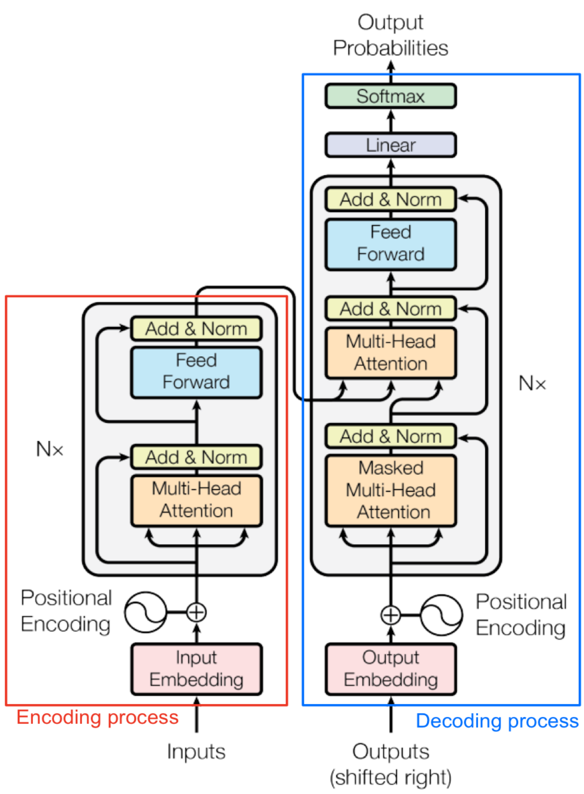


Figure 6: Transformer Model Architecture, modified from [15]. Note: Nx indicates that the shown layers appear N times in the network.

3.3 Reformer

Note: The information used in section 3.3 was extracted from [18]. Additionally, [19] was also used since it provides a good summary of the longer report.

While transformers have been proven to be effective for language modelling, they have two problems when it comes to efficiency:

- When analyzing a sequence using self-attention, all the word pairs within it are considered. This can be extremely inefficient, especially when analyzing texts that contain a high number of words.
- Transformers have multiple layers whose activation values need to be stored in memory. A single layer can require a few GB of storage, which means that memory can quickly run out.

In order to solve these issues, the reformer was introduced [18]. It is a network architecture based on transformers, that introduces a couple of approaches to improve efficiency. Firstly, this report explained a concept known as Locality-Sensitive Hashing (LSH). LSH computes a hash function that is capable of grouping similar vectors (i.e., words in the case of language modelling) together.

This process can be seen on figure 7, where each color represents a different hash (as similar words will have the same hash, they are represented by the same color). Once the hashes have been assigned, words with the same hash will be brought together and the sequence will be split into chunks. Attention will be applied inside those chunks (and the word that is adjacent to that chunk). This process can help to reduce the computational load considerably.

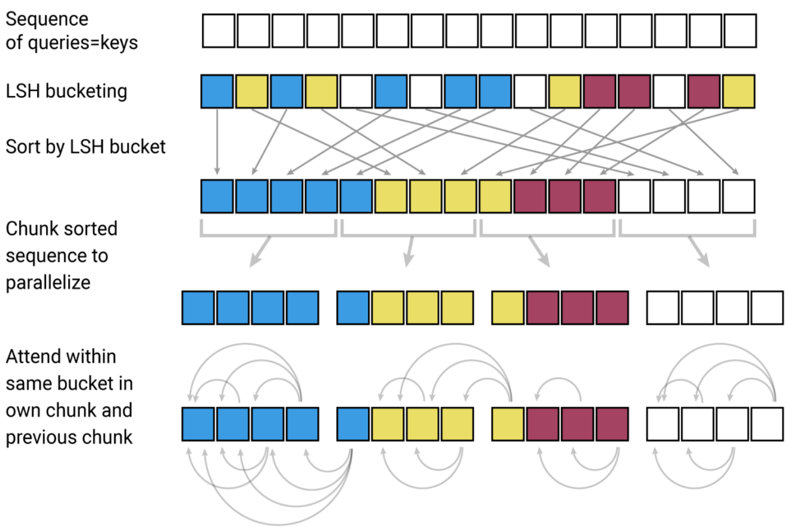


Figure 7: "Simplified depiction of LSH Attention showing the hash-bucketing, sorting, and chunking steps and the resulting causal attentions" [18]

The second technique introduced in this report is the application of reversible layers (introduced in [20]) in the transformer. In your typical layer, the activations within it are used to update the vectors that go through the layer. However, reversible layers have two sets of activations: one which updates the value of the vectors and another one that computes the changes that have been made. Therefore, instead of needing to store the activation values of any intermediate layer, the network can be run in reverse and those values are calculated by subtracting the changes that were made.

Reversible layers have the following equations (where x_1 and x_2 are inputs, and y_1 and y_2 are outputs):

$$y_1 = x_1 + F(x_2) \qquad y_2 = x_2 + G(y_1) \tag{3}$$

Subtracting the residuals allows these types of layers to be reversed:

$$x_2 = y_2 - G(y_1) \qquad x_1 = y_1 - F(x_2) \tag{4}$$

3.4 Bidirectional Encoder Representations from Transformers (BERT)

BERT [8] is a language understanding model based on the previously explained transformer architecture (i.e., as its name suggests, it is an encoder only). The pre-training process of BERT has two main parts, which differentiate it from other language models:

1. This model is bidirectional. That means that the analysis of the text is more powerful than other models that employ a left-to-right or right-to-left analysis. In order to generate a bidirectional representation, a process known as **Masked LM** is used. Through it, a percentage (15% in the case of the experiments in [8]) of the tokens are masked randomly and the model is asked to predict those masked words. This way, the words that come before and after the masked token are considered as context.

2. For tasks like Question-Answering (QA), understanding the relationship between different sentences is key. To train the model to achieve this, sentence pairs will be extracted from the utilized corpus (in the BERT paper, the BooksCorpus [21] and English Wikipedia corpus were utilized). When choosing sentence pairs A and B, 50% of the time B will be the sentence that comes after A while the other 50%, B will be a randomly chosen sentence from the corpus. The model will then be trained to predict whether the sentences go right after each other or not. This process is known as **Next Sentence Prediction**.

4 Understanding Model

4.1 Explanation and Setup

While there are models that classify input user sentences based on sarcasm, emotional state, topic, etc. our system will have a binary classification.

It is assumed that the any user utterance can be split into one of two classes: Objective or Subjective. Those classes/intentions are referring to whether the response to be generated requires our system to search information (objective) or not (subjective). Another way to think about these two types is that objective utterances have a correct/factual answer, while subjective ones require a more personal response. Here is an example to show what is meant by each of the intentions:

- Objective Class

  - USER: When was *penicillin* discovered?
  - SYSTEM: *1928*.

  In this example, the system would not be able to generate a response unless it finds information about the penicillin. As has previously been mentioned, objective sentences have one correct answer, 1914 in this particular case.

- Subjective Class

  - USER: Would you like to have a pizza?
  - SYSTEM: Sure, thanks.

  This example, however, shows an input sentence where the user does not expect the system to look up any information to generate a response. Furthermore, the system could both respond positively or negatively to the suggestion (i.e., personal answer).

A data set containing the objective/subjective classification discussed was not found, so it was created. For objective statements, the Natural Questions set was used [22], which is comprised of natural questions anonymized from google search queries, such as: *"Who was the first Royal to live in Buckingham Palace?"* and *"What type of warfare was going on in the battle of Britain?"*. For subjective utterances, sentences from the CONVAI2[1] set were utilized. Some examples of sentences included in this data set are: *"I'm 40 years old"* and *"On weekends I do live action role playing events"*. The created data set contains 5204 sentences, with a distribution of objective/subjective of 50/50 (even though more objective sentences could have been added, the distribution was kept at 50/50 to ensure the set was balanced).

Nevertheless, the generated dataset had a problem. All objective statements included were questions, while subjective ones were not. Therefore, our model would likely label any question as objective and any non-question sentence as subjective (this is could lead to some incorrect classifications). In order to solve this problem, the SubjQA dataset [23] was utilized. This dataset is comprised of non-factual questions about different "domains" (e.g., movies, books) [2].

From the questions found in the book folder, the ones that contained *'you'* were extracted, obtaining a total of 197 personal questions like *How was the action in the book that you completed reading?* and *Do you want to have a long life?*. Those questions were then included in the initial dataset (i.e., the 197 newly generated utterances will substitute the first 197 subjective statements on the initial dataset, to maintain a 50/50 distribution).

The generated set was used to train the understanding model (built using TensorFlow) whose structure is illustrated on figure 8. The embedding layer used was extracted from https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1. In the figure, the bias value is referring to the number of units (neurons) and the red segments are indicating which activation function each Dense layer has.

[1]Conversational Intelligence Challenge. Conversations between 1,000 volunteers and 10 chatbots were collected.
[2]The sentences were extracted from reviews from Amazon, TripAdvisor and Yelp.

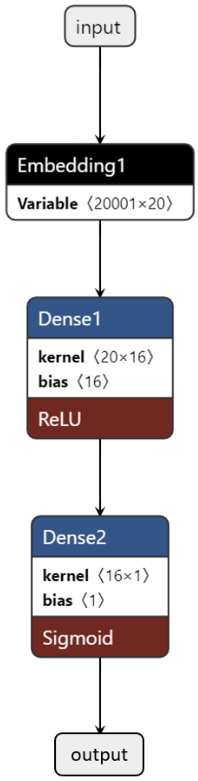


Figure 8: Understanding Model Architecture

The model was compiled using the *Adam* optimizer, and was trained for 200 epochs and 10% validation (this will be explained further on section 4.2). The objective/subjective dataset was split into a training and a testing set (consisting of 4160 and 1044 sentences, respectively). After training, the model will provide an output between 0 and 1, with 0 meaning *"subjective"* and 1 meaning *"objective"*. A threshold will be defined at 0.5. Therefore, any value obtained below 0.5 will be labelled as subjective and any value above it will be objective.

Note: The code used for the model training can be found at Appendix A.

4.2 Discussion of Results

Figure 9 displays the loss experienced by the model. Two different values are being displayed: training and validation loss. The training loss is calculated on 90% of the training set (3744 sentences), while the remaining 416 sentences from the training set are used as validation (i.e., to help with the tuning process and avoid overfitting the model to the dataset). The training set started with a loss of 0.68 and ended with 0.07, whereas the validation set started with 0.49 and ended with 0.11. When evaluating the model on the test set, a loss of 0.11 was obtained.

On the other hand, figure 10 shows the model accuracy. The model had an initial accuracy of 64.9% on the training set, and finished with 97.8%. With regards to the validation set, it initially predicted 80.3% of the sentences correctly and reached a 96.6% accuracy in the end. When trying to predict the label for the utterances on the test set, the model achieved an accuracy of 96.4%.

Figure 9: Understanding Model Loss

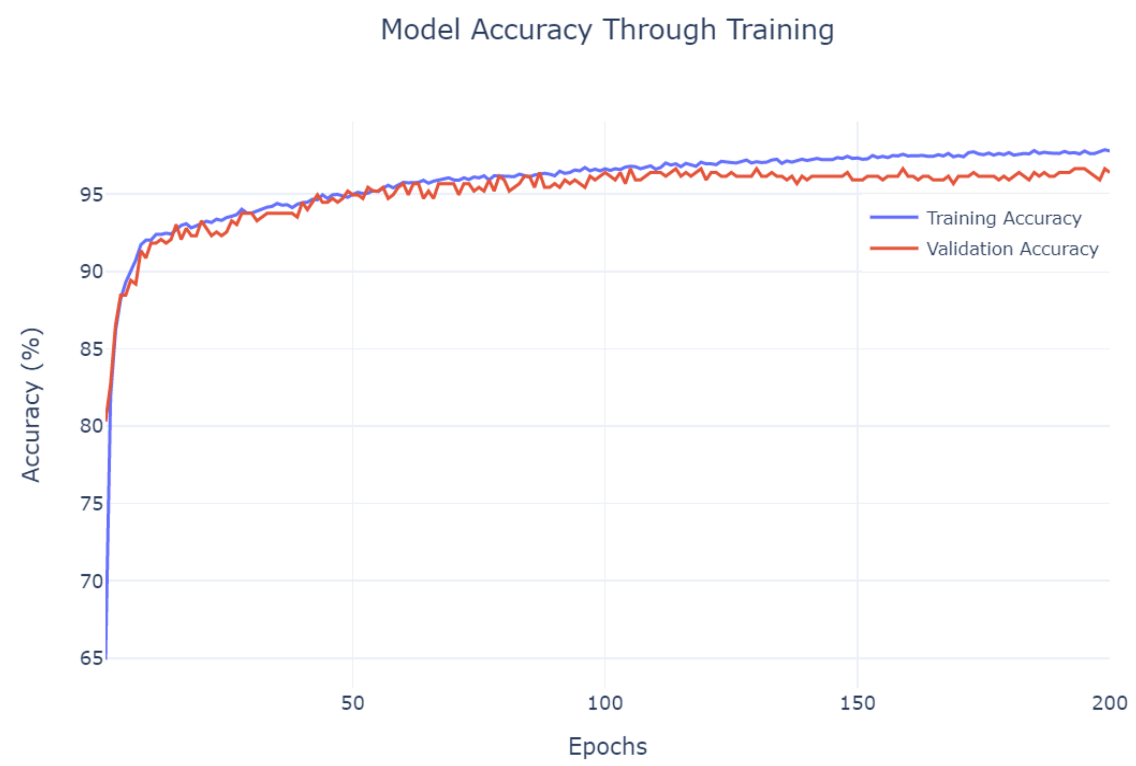


Figure 10: Understanding Model Accuracy

These results indicate that the model is able to properly distinguish between the different kinds of sentences that have been defined. Nevertheless, some sample sentences will be fed to the model and analyzed for further understanding of the meaning of the results. The sentences and their predictions are displayed on table 1.

From the sample sentences, a series of conclusions were extracted:

1. All the labels were properly determined. This means that the model can be utilized for the classification task described in this report, and it performs well.

2. Non-question statements obtain a predicted value close to 0. This happens because in the generated dataset, objective utterances were always questions. Therefore, the model has a tendency to classify statements that are not inquiries as subjective.

3. While the question *"What is the density of water?"* obtained a value close to 1, the rest of the questions did not obtain values that were that clear (especially *"When was penicillin discovered?"* and *"What do you think of me?"*, as they both were relatively close to the 0.5 threshold). This suggests that the model does not fully understand the difference between these two types of sentences. One way to solve this problem would be to create a larger dataset, which would ensure the model could have more sentences to learn from (the utilized 5204-sentence set is relatively small).

Sample Sentence	Expected Label	Predicted Label	Predicted Value
"What is the density of water?"	Objective	Objective	0.999
"When did you have breakfast?"	Subjective	Subjective	0.212
"I like your shirt"	Subjective	Subjective	0.002
"When was penicillin discovered?"	Objective	Objective	0.572
"What do you think of me?"	Subjective	Subjective	0.406
"I will get a sandwich for dinner"	Subjective	Subjective	0.000

Table 1: Sample Sentence Classification Using the Understanding Model

However, there is one issue that must be discussed. There is a type of sentences which combines both objective and subjective elements. As an example, the sentence *"What is your favourite part of Rome?"* would be labelled as subjective by our model. This makes sense, since it is a personal question. Nevertheless, because the city of Rome is mentioned, the model would need to extract some information about it in order to mention what "its favourite part" is.

One way to solve this would be to include a third class in our classification. This type of sentence could be named "Mixed". This new class would contain any sentence for which the system must look up information about a specific topic and generate a "personal" response using that information. Sentences like these one would need to be generated and inserted into the dataset so that the model[3] can differentiate them.

[3]If a longer and more complex dataset was used, a more advanced model would be required.

5 Objective Response Retrieval

5.1 Explanation and Setup

For question-answering tasks, there are two main approaches: *Open-book* (i.e., the system can get data from external sources) and *Closed-book* (i.e., no external data is allowed). The objective class defined is not limited to a specific conversation topic, which means that a closed-book approach cannot be utilized (i.e., while the model could be prepared to answer questions about specific topics, the number of topics it could discuss would be limited unless an open-book approach is used. Our system must be able to answer any question regardless of the topic).

Once open-book has been chosen, there are two different types of models that could be used: *retriever* or *generator*. The former *extracts* the answer from a given context and presents it as the response, while the latter *extracts* the answer from the context and *generates* a response that contains the extracted answer. An example is provided to further illustrate the difference between these two models:

- Question: *"When did WWI start?"*

- Context: *"World War I, also known as the Great War, began in **1914**..."*.

a) **Retrieval model**: *1914*. b) **Generator model**: WWI started in *1914*.

Even though the generator model will create a more complete result, the retrieval model would be simpler, which makes it more appealing for my system. Hence a retrieval model will be utilized to obtain the answers to the user's queries.

However, it is necessary to extract the context of the question before the answer could possibly be obtained. A simple way to do this is to search the query on google, and extract the information from one of the top links that are given as results.

The python **googlesearch** library has a function called *search* which will obtain the different URLs for the top results of a query. It will be utilized to get the first 10 links, and then a for loop will try to open one of the URLs and extract its text (i.e., through a combination of the libraries **urllib** and **BeautifulSoup**). If the URL cannot be opened (e.g., 404 error), if it contains an image whose text cannot be extracted or if it comes from an unwanted website (e.g., wikipedia, facebook, quora), the code will go to the following URL provided by the search.

Once one of the links has been opened and its text content was obtained, a *BERT* model trained on the Stanford Question Answering Dataset (SQuAD)[4] [24] will be utilized, to read that text and extract the answer that the user is looking for. The model used is available at https://huggingface.co/deepset/bert-base-cased-squad2.

Note 1: The code utilized for the Objective Response Retrieval system can be found at appendix B.

Note 2: For a deeper understanding of the Objective Response Retrieval System, an activity diagram for the query context extraction has been sketched. It can be seen on figure 14, which is located in appendix E.

5.2 Discussion of Results

For this part, a pre-trained model that had achieved a high accuracy on the SQUAD dataset was utilized. Therefore, no graphs have been produced. The way to analyze the results is by asking a set of different objective questions and seeing how accurate the model is. The sample questions and their answers are displayed on table 2.

The first 5 example questions showcased were properly answered. The questions are varied, asking about people, dates, even for a physics concept (density). This illustrates that the pre-trained model used is highly accurate for question-answering tasks. Nevertheless, the last two questions can help to explain some of the shortcomings of the system.

When asking "Who was the first pick in the last NBA draft?" the model obtained a completely wrong answer, since it did not even give the name of a person. The system obtained the first 10 links to that query, and the URL that it opened was https://www.nba.com/stats/draft/history/. When opening it, the problem can be spotted. This site does not provide a piece of text that the model can analyze. The site displays a list of players, but the system was not able to extract that list properly.

[4]Paragraphs from Wikipedia articles were extracted, and different questions were created for each paragraph.

This is one of the flaws with the system. So far, if the URL does not come from facebook, wikipedia or quora; does not contain an image in png, jpeg, jpg formats; and it can be opened, then it is assumed that the text from that link will be properly extracted. Nevertheless, there are many different websites, the content of some of which will not be properly taken and the model will not have a good context to work with.

The other flaw can be seen in the last question. If the system is asked an open question like ”How do you improve your mental health?”, which does not have a short and precise answer, it will not be able to extract a useful answer. The URL retrieved by the system is https://medlineplus.gov/howtoimprovementalhealth.html, which contains a useful guide with numerous recommendations (e.g., staying positive, practicing gratitude, taking care of your physical health). However, the model expects a short answer, which is why it is not able to summarize the information presented in the context.

The way to solve the first issue is by having a better screening process. The system must be able to determine if the information extracted from the link is good enough to answer the question. There are some models that analyze coherency, which could be combined with our current system to determine if the context is coherent enough to be able to provide an accurate answer.

With regards to the second problem, this model is just not appropriate for summarization tasks. Once a statement has been labelled as objective, a second classification process could take place, which would determine if the the question is asking for a precise short answer or for an open and longer response. If the latter is true, then a content summarizing model would be utilized to condense the information in the given URL.

Sample Sentence	Retrieved Answer	Is the Answer Correct
”What is the density of water?”	1 g/cm3	✓
”What is the capital of France?”	Paris,	✓
”Who was the prime minister of the UK during WWII?”	Winston Churchill,	✓
”Who was the president of South Africa in the year 2000?”	Thabo Mbeki:	✓
”When did the American Civil War begin?”	1861,	✓
”Who was the first pick in the last NBA draft?”	2	✗
”How do you improve your mental health?”	Realize your full potential	✗

Table 2: Sample Question-Answering Using the Objective Response Retrieval System

6 Subjective Response Generation

6.1 Explanation and Setup

In order to reply to "subjective" sentences, the response generation model must have been trained on casual/daily conversations. That way, the model will understand the utterances human beings produce when participating in "small talk" conversations. There are plenty of data sets that are used to train models for response generation tasks (e.g., Cornell Movie-Dialog Corpus, Reddit data set, Amazon QA). However, most of those datasets are not comprised of conversations that could be categorized as casual.

Therefore, the DailyDialog set will be utilized [7]. It contains 13118 manually labelled multi-turn dialogues, in which there are annotations about the topic of conversation, emotion and act. These annotations, however, will not be used for the training of the subjective response generation model, only the actual conversation texts. The creators of the dataset mentioned that they extracted the conversations from different websites aimed at non-English speakers learning the language for daily situations (even though the sites have not been specified). Here are two example conversations that will illustrate the kind of content present on this set:

Conversation 1

- A: Hi Mark.
- B: Hi.
- A: What are you planning to do today?
- B: I'm not sure yet.
- A: Would you like to have lunch with me?
- ...

Conversation 2

- A: Hey John, nice skates. Are they new?
- B: Yeah, I just got them. I started playing ice hockey in a comminuity leage. So, I finally got myself new skates.
- A: What position do you play?
- B: I'm a defender. It's a lot of fun. You don't have to be able to skate as fast on defense.
- ...

This part of the project was influenced by an online course named Natural Language Processing with Attention Models [9]. The example given in the course was very similar to what I needed to do for my project, with two main differences: they used the MultiWOZ dataset, which is focused on task-oriented conversations, and they utilized a pre-trained model, instead of training it from scratch.

The steps followed in the training code are specified below. Trax is the library that was utilized for the creation of the reformer model and data pipeline, and the training process is as follows:

1. The content of the train and test files will be read. Once it has been read, the conversations will be extracted and two delimiters (i.e., Person 1 and Person 2) will be utilized to differentiate between the different conversation turns.
2. A data pipeline will be defined. This pipeline will take the data stream perform the following tasks: shuffling, tokenization, length filtering (2048 being the maximum length), bucketing by different lengths (the defined boundaries are 128, 256, 512 and 1024) and adding the loss weights.
3. The reformer model used will be specified. A 6-layer 33000-word reformer will be used. This reformer will be utilizing self-attention.
4. Once the model has been initialized, a training loop will begin. The model was trained for **20660** training steps throughout 6-7 days[5], using the Adam optimizer with a learning rate of 0.001. A learning rate schedule was used to vary the lr slightly to optimize the training process.

Note: The code for the Reformer Model training is displayed in appendix C.

[5]I tried utilizing a GPU to speed up the training process. Nevertheless, an out-of-memory error forced me to have to train it relying solely on my laptop's CPU.

6.2 Discussion of Results

Figure 11 displays both the training loss (i.e., evaluated on the train set) and evaluation loss (i.e., calculated on the test set) described by the Reformer model throughout the 20660 training steps. The training loss started with a value of 10.42 and had a final value of 2.12, whereas the evaluation loss started with a value of 10.44 and ended with 3.87.

Even though the training loss can still be reduced, the evaluation loss fluctuates and is no longer being altered significantly. Therefore, the training process was stopped to avoid overfitting the model to the train set.

On the other hand, figure 12 displays the evaluation accuracy (i.e., calculated on the test set) of the model. The model started with an accuracy of 0.17% and that accuracy increased significantly until step 2200 (where it reached 47.7%). Beyond there, the accuracy increased slower and there was a high level of fluctuation[6]. The final accuracy value obtained was 34%, and the peak accuracy obtained was 58.16%.

Let's compare my results to the ones obtained by the same reformer trained on the MultiWOZ dataset. The coursera course does not provide any accuracy value. Nevertheless, a similar project was found on GitHub [10], and the obtained accuracy value was 67.15% on 20,000 training steps. One reason which could explain why the model did not achieve that level of accuracy is that the DailyDialog set contains casual conversations, which are more complex than task-oriented conversations.

In order to improve this accuracy in the future, there is a few of things that can be done. Firstly, the hyperparameter values (e.g., learning rate, optimizer, batch size) must be varied to find the best results. Secondly, the same model could be trained on a different social conversations dataset. Lastly, a different model could be trained on the DailyDialog set.

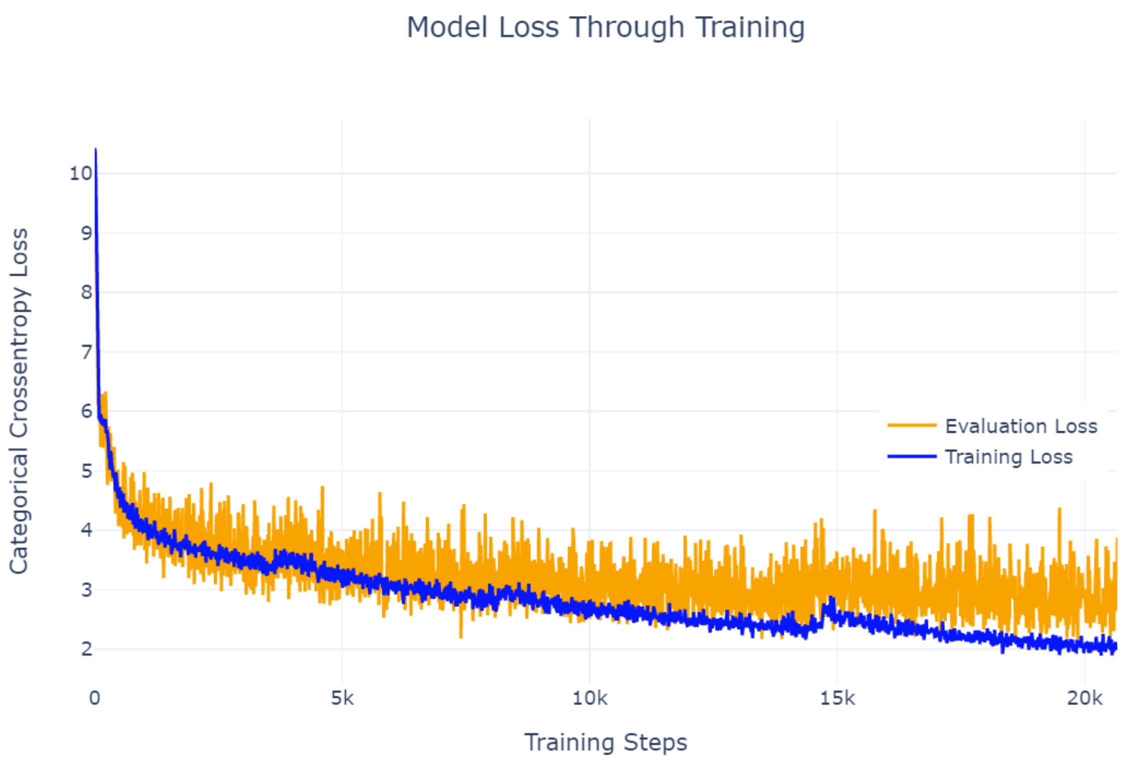


Figure 11: Response Generation Model Loss

[6]One of the reasons for this fluctuation is that a test generator is being used. This means that random test sentences are being chosen for the evaluation, instead of consistently evaluating on the same sentences.

Figure 12: Response Generation Model Evaluation Accuracy

Lastly, a couple of sample sentences will be fed into the model and the output will be displayed. The model has been trained on full conversations, which is why the generated output contains multiple conversation turns (the maximum length of the output has been set to 100 characters).

As can be seen, the generated outputs can be quite repetitive, especially in examples 2 and 3. This, however, is not a problem because in section 7 (where all the different parts of the system are put together), when having full conversations only the first conversation turn generated will be taken. Furthermore, since the model has been trained on full conversations, more than one conversation can be provided as input. This will help the system to keep track of the conversation context.

Example 1

Input: *Person 1: Would you like to have a beer?*

Output: *? Person 2: Yes, please. Person 1: Would you like to have dinner with me? Person 2: Yes, please. Person 1: I'd like to make a reservation for tonight. Person 2: That's good. Person 1: Do you have anything to declare? Person 2: No, thanks. Person 1: You're welcome. Person 2: I'm sorry. I*

Example 2

Input: *Person 1: What is your favourite sport?*

Output: *? Person 2: I like reading novels. Person 1: What do you think of it? Person 2: I like it very much. Person 1: I like it very much. Person 2: I like it. I think it's a good idea. Person 1: I like it. Naturally it's worth it. Person 2: I like it. Person 1: I like it. Person 2: I like it. Person 1*

Example 3

Input: *Person 1: I enjoy talking to you.*

Output: *. Person 2: I'm glad you like it. Person 1: I'm glad you like it. Person 2: I'm glad you like it. Person 1: I'm glad you liked it. Person 2: It's nice of you. Person 1: I'm glad you like it. Person 2: It's nice of you. Person 1: I hope you're right*

7 Creating the Overall System

7.1 Explanation and Setup

The last step in the project is combining the sentence classifier, objective response retrieval and subjective response generation. Additionally, speech-to-text and text-to-speech systems will be included to allow the user to have spoken conversations with the system, instead of having to type the conversations. Figure 13 showcases the diagram for the overall system[7].

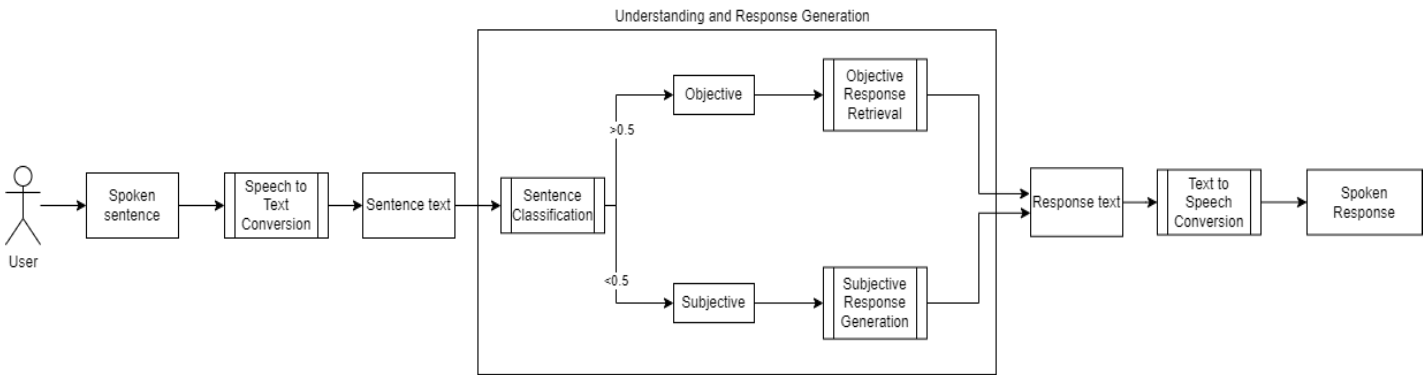


Figure 13: Overall System Diagram

To convert from speech to text, the speech_recognition library will be utilized, as it allows for real time conversion. The recognize_google_cloud method will be used, hence the google cloud credentials must be stored in a json file within the system. Once a text response has been generated, the pyttsx3 library will be used to convert the text to speech.

The understanding, objective response retrieval and subjective response generation[8] models will be loaded and they will be combined in a loop function that will keep the model running until the user stops the program.

Another important point is the context of the conversation. Every newly generated sentence will be stored inside a list. This list will be updated regularly to contain the last two conversation turns (i.e., the user input and the previous statement). Those two elements in the list will be used as inputs for the reponse generation part[9].

Note: The code for the overall system is included in appendix D.

7.2 Discussion of Results

In this section, four different sample conversations will be displayed. These sentences combine both objective and subjective utterances, to allow the user to understand the capabilities of the system. From these conversations, a series of conclusions have been extracted from the generated responses:

- The responses can sometimes be repetitive (e.g., there is plenty of repetition in sample conversation 1).
- The responses are sometimes be random (e.g., in sample conversation 2 the system mentions going to the hospital, which has nothing to do with the input "I'm really tired").
- The objective response generation part works rather well when looking for precise and short answers (as had already been mentioned in section 5.2).
- The responses are reasonable. The system created for this project has plenty of room for improvement, especially in the subjective response generation part. Nevertheless, the system is able to provide sufficiently appropriate responses to allow for some form of discussion.

[7] The sentence classification part displays < 0.5 and > 0.5 values, because the classifier returns a value between 0 and 1 (threshold set at 0.5).

[8] For the subjective response generation model a combination of functions will be defined to generate the dialogue.

[9] The system was also tested with three sentences as context. This, however, yielded negative results because the responses generated were not very accurate.

Sample Conversation 1

- User: Do you know how to drive?
- System: Well, I'm not sure. I'm not sure what to do.
- User: Would you like me to teach you how to drive?
- System: I'm sorry, but I'm not sure. I'm not sure what I can do.
- User: Driving is easy, trust me.
- System: You are right. I'm so sorry. I'm so sorry. I'm terrified.

Sample Conversation 2

- User: I'm really tired.
- System: I'm sorry. I'm going to the hospital.
- User: Is everything okay?
- System: I'm not sure. I'm going to be late.
- User: Good luck with everything!
- System: I'm glad you are going to be late.

Sample Conversation 3

- User: Would you like to go for a walk with me?
- System: I'd like to.
- User:What place would you like to go to?
- System: I'd like to have a look at the airport.
- User: Okay. Then, we will go to the airport.
- System: Thank you. I'll be there on time.
- User: What is the closest airport in Madrid?
- System: Madrid - Cuatro Vientos Airport[10].

Sample Conversation 4

- User: Who was the president of the USA during the Civil War?
- System: Abraham Lincoln.
- User: When was Abraham Lincoln assassinated?
- System: April 14, 1865.
- User: How old was Abraham Lincoln when he was assassinated?
- System: 56.

[10]Which is actually the closest airport to my current location. Nevertheless, these types of questions are not usually answered properly since the model does not have any information about your location unless otherwise specified.

8 Conclusion and Future Work

This project was aimed at creating a conversational system that simplifies the approach to social conversations. In order to do that, a simple objective/subjective classification system was implemented, which obtained a 96.4% testing accuracy. For objective response retrieval, a pre-trained BERT model trained on a question-answering dataset was used (in combination with a piece of code which extracts the context for any question from google search results), and it obtained positive results. Lastly, for subjective statements, a reformer model was trained on the DailyDialog set, obtaining a peak evaluation accuracy of 58.16%.

When combining everything together, it was observed that the system is able to have casual conversations with users, but there are a few areas for improvement in the future. Firstly, the objective/subjective dataset should be expanded and modified to include a "mixed" class, which combines elements of both objective and subjective statements (as explained in section 4). Secondly, the system should distinguish between two types of objective questions (i.e., the ones that are looking for a short and precise answer, and those looking for an open and longer response). Lastly, subjective response generation model should be trained on different datasets or could be altered to seek an improvement in the accuracy of its responses.

Casual conversations are complex and difficult to replicate using NLP. This report does not have the aim to introduce new NLP models and explore their potential. Nevertheless, through the implementation of currently available networks, a simple and innovative approach (i.e., in the initial classification process) has been defined which I hope can help to showcase the possibility of simplifying these kinds of conversations, with the aim of making this promising technology more approachable for future researchers.

References

[1] IBM, "IBM Shoebox." [Online]. Available: https://www.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html

[2] A. Bell, "The History of Speech Recognition – Part 1," June 2019. [Online]. Available: https://whatsnext.nuance.com/en-gb/dragon-professional/history-speech-recognition/

[3] A. Mutchler, "Voice Assistant Timeline: A Short History of the Voice Revolution," July 2017. [Online]. Available: https://voicebot.ai/2017/07/14/timeline-voice-assistants-short-history-voice-revolution/

[4] E. Ventola, "The structure of casual conversation in english," *Journal of pragmatics*, vol. 3, no. 3-4, pp. 267–298, 1979.

[5] E. Gilmartin, C. Saam, C. Vogel, N. Campbell, and V. Wade, "Just talking-modelling casual conversation," in *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, 2018, pp. 51–59.

[6] D. Jurafsky and E. Shriberg, "Switchboard swbd-damsl shallow-discourse-function annotation coders manual," 1997.

[7] Y. Li, H. Su, X. Shen, W. Li, Z. Cao, and S. Niu, "Dailydialog: A manually labelled multi-turn dialogue dataset," *arXiv preprint arXiv:1710.03957*, 2017.

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

[9] Y. B. Mourri, L. Kaiser, and E. Shyu, "Natural Language Processing with Attention Models," 2020. [Online]. Available: https://www.coursera.org/learn/attention-models-in-nlp

[10] S.-M. Liao, "Chatbot-Reformer," November 2021. [Online]. Available: https://github.com/sliao-mi-luku/Chatbot-Reformer

[11] R. Higashinaka, K. Imamura, T. Meguro, C. Miyazaki, N. Kobayashi, H. Sugiyama, T. Hirano, T. Makino, and Y. Matsuo, "Towards an open-domain conversational system fully based on natural language processing," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, 2014, pp. 928–939.

[12] R. Ferreira, M. Leite, D. Semedo, and J. Magalhaes, "Open-domain conversational search assistant with transformers," *arXiv preprint arXiv:2101.08197*, 2021.

[13] TensorFlow, "Word embeddings," February 2022. [Online]. Available: https://www.tensorflow.org/text/guide/word_embeddings

[14] N. Shazeer, R. Doherty, C. Evans, and C. Waterson, "Swivel: Improving embeddings by noticing what's missing," *arXiv preprint arXiv:1602.02215*, 2016.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[16] M. Phi, "Illustrated Guide to Transformers - Step by Step Explanation," April 2020. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

[17] N. Adaloglou, "How Transformers work in deep learning and NLP: an intuitive introduction," December 2020. [Online]. Available: https://theaisummer.com/transformer/

[18] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[19] N. Kitaev and Ł. Kaiser, "Reformer: The Efficient Transformer," January 2020. [Online]. Available: https://ai.googleblog.com/2020/01/reformer-efficient-transformer.html

[20] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," *Advances in neural information processing systems*, vol. 30, 2017.

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

[22] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, M. Kelcey, J. Devlin, K. Lee, K. N. Toutanova, L. Jones, M.-W. Chang, A. Dai, J. Uszkoreit, Q. Le, and S. Petrov, "Natural questions: a benchmark for question answering research," *Transactions of the Association of Computational Linguistics*, 2019.

[23] J. Bjerva, N. Bhutani, B. Golahn, W.-C. Tan, and I. Augenstein, "SubjQA: A Dataset for Subjectivity and Review Comprehension," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2020.

[24] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

A Code: Understanding Model Training

```
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
import pandas as pd
import openpyxl


sentences_df = pd.read_excel('sentence_classification_dataset.xlsx', engine='openpyxl')

# Split the sentences into test and train sets
train_df = sentences_df[:4160]
test_df = sentences_df[4160:]

# Load the embedding layer from tfhub
embed = hub.load(handle="https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1")

# Build the TensorFlow model
model = tf.keras.Sequential([
        tf.keras.layers.Dense(16, activation='relu', input_shape = (20,)),
        tf.keras.layers.Dense(1, activation='sigmoid') # Output layer
    ])

model.compile(optimizer='adam',
             loss='binary_crossentropy',
             metrics=['accuracy'])

# Train the model for 200 epochs, with a 0.1 validation split
history = model.fit(embed(train_df['Sentences']),
                  train_df['Label'],
                  epochs=200,
                  validation_split=0.1
                  )

# Evaluate the model on the test set
eval_results = model.evaluate(embed(test_df['Sentences']), test_df['Label'], verbose=0)
for metric, value in zip(model.metrics_names, eval_results):
    print(metric + ': {:.3}'.format(value))


# Once the model has been trained, define a function to make predictions
def predict_label(sentence):
    """predict if the sentence is objective or subjective"""
    prediction = model.predict(embed([sentence]))[0][0]
    if prediction < 0.5:
        print("Predicted label is: Subjective")
    else:
        print("Predicted label is: Objective")

# Test model on a sample sentence
sentence = "do you like pizza"
predict_label(sentence)

# Save the model
model.save('second_understanding_model.h5')
```

B Code: Objective Response Retrieval

```
import googlesearch
from googlesearch import search
import urllib.request
from bs4 import BeautifulSoup as bs
from transformers import pipeline, BertTokenizer, BertForQuestionAnswering
import numpy as np

# Extract the BERT tokenizer from
modelname = 'deepset/bert-base-cased-squad2'
tokenizer = BertTokenizer.from_pretrained(modelname)
model = BertForQuestionAnswering.from_pretrained(modelname)

# Define a question-answering pipeline
qa = pipeline('question-answering', model=model, tokenizer=tokenizer)

def check_if_unwanted(url):
    """Check if the URL contains an image or comes from an unwanted site"""
    unwanted_formats = ["png", "jpeg", "jpg", "facebook", "wikipedia", "quora"]
    return any(image_format in url for image_format in unwanted_formats)

def get_context(question):
    """Extract context for a specific question"""
    for url in search(question, num_results=10): # Go through the top 10 result URLs
        if check_if_unwanted(url):
            continue # If unwanted, go to the the next URL
        else:
            try:
                # Get the data from the url
                open_url = urllib.request.urlopen(url)
                data = open_url.read()

                # Clean the content (remove the html formatting)
                soup = bs(data, features="html.parser")
                for script in soup(["script", "style"]):
                    script.extract()
                context = soup.get_text()

                # Perform further cleaning (extracted from stackoverflow)
                lines = (line.strip() for line in context.splitlines())
                chunks = (phrase.strip() for line in lines for phrase in line.split(" "))
                context = ' '.join(chunk for chunk in chunks if chunk)

                return context
            except:
                continue # If the URL could not be opened, go to the next

def extract_answer(question, context):
    """Extract answer for a specific question, given a context"""
    ans = qa({
        'question': question,
        'context': context})
    return ans['answer']

def answer_question(question):
    """Extract answer for a specific question, without given context"""
    context = get_context(question)
    return extract_answer(question, context)

# Test the Objective Response Retrieval System
answer_question("Who was the president of the USA during the Civil War?")
```

C Code: Reformer Model Training

To be noted: The code presented here is very similar to the one shown at [9].

```
import json
import random
import numpy as np
import trax
from trax import layers as tl
from trax.supervised import training

VOCAB_FILE = 'en_32k.subword'
VOCAB_DIR = 'path/to/vocab/file/'

def read_json_file_content(file_path):
    """Read content from file in given path"""
    content = []

    # Extract the lines from the file
    with open(file_path, 'r') as f:
        content_lines = f.readlines()

    # Append each line to content
    for line in content_lines:
        content.append(json.loads(line))
    return content

def yield_delimiters(number_of_sentences):
    """Yield the conversation delimiters"""
    for i in range(number_of_sentences):
        if i%2 == 0: # Number is even or 0
            yield "Person 1: "
        else:
            yield "Person 2: "


def extract_conversations(content):
    """Function that will extract the conversation from the file content and
    clean them."""
    # Characters that are preceded by a white space in the corpus, which should not be the case
    characters_without_space = [',', '?', '!', '.', "'"]

    all_conversations = []
    for conversation in content: # Go through the different conversations
        conversation_text = ""
        number_of_sentences = len(conversation['dialogue'])

        # Get the delimiters for the different conversation turns
        delimiters = list(yield_delimiters(number_of_sentences))

        sentence_counter = 0
        for sentence in conversation['dialogue']:
            split_sentence = sentence['text'].split()
            reconverted_sentence = delimiters[sentence_counter]
            for i in range(len(split_sentence)):
                # If the word contains any of the characters that do not require space, do not include a space
                if any(split_sentence[i] == character for character in characters_without_space):
                    reconverted_sentence = reconverted_sentence + split_sentence[i]
                # If the previous word is the apostrophe character, do not include a space afterwards
                elif i != 0 and split_sentence[i-1] == "'":
                    reconverted_sentence = reconverted_sentence + split_sentence[i]
                else: # For all other cases, a space must be included
```

```
                reconverted_sentence = reconverted_sentence + " " + split_sentence[i]

            if sentence_counter == 0: # If this is the first sentence in the conversation
                conversation_text = conversation_text + reconverted_sentence
            else: # If there is another sentence before, include a space
                conversation_text = conversation_text + " " + reconverted_sentence
            sentence_counter = sentence_counter + 1

        all_conversations.append(conversation_text) # Append the new conversation to the
            list
    return all_conversations

def stream(data):
    """Generate a data stream from the given data"""
    # loop over the entire data
    while True:
        d = random.choice(data)

        # yield a pair of identical values
        yield (d, d)

def ReformerLM(vocab_size=33000, n_layers=6, mode='train',
     attention_type=tl.SelfAttention):
    """Define the reformer model to be used"""
    model = tl.Serial(
                trax.models.reformer.ReformerLM(
                vocab_size=vocab_size,
                n_layers=n_layers,
                mode=mode,
                attention_type=attention_type
            ),
            tl.LogSoftmax()
        )

    return model

def training_loop(ReformerLM, train_gen, eval_gen, output_dir = "path/to/model/"):
    """Define the training loop that will be used to train our model"""

    # Define the learning rate schedule
    lr_schedule = trax.lr.warmup_and_rsqrt_decay(
        n_warmup_steps=200, max_value=0.001)

    # Define the train task
    train_task = training.TrainTask(
        labeled_data=train_gen,
        loss_layer=tl.CrossEntropyLoss(),
        optimizer=trax.optimizers.Adam(0.001),
        lr_schedule=lr_schedule,
        n_steps_per_checkpoint=10
    )

    # Define the eval task
    eval_task = training.EvalTask(
        labeled_data=eval_gen,
        metrics=[tl.CrossEntropyLoss(), tl.Accuracy()]
    )

    # Define the training loop
    loop = training.Loop(ReformerLM,
                        train_task,
                        eval_tasks=[eval_task],
                        output_dir=output_dir)
    return loop
```

```python
# Extract the train and test data
train_data = extract_conversations(read_json_file_content('train.json'))
test_data = extract_conversations(read_json_file_content('test.json'))

# Define a data pipeline
data_pipeline = trax.data.Serial(
    trax.data.Shuffle(),
    trax.data.Tokenize(vocab_dir=VOCAB_DIR,
                       vocab_file=VOCAB_FILE),
    trax.data.FilterByLength(2048),
    trax.data.BucketByLength(boundaries=[128, 256, 512, 1024],
                             batch_sizes=[16,  8,    4,   2, 1]),
    trax.data.AddLossWeights(id_to_mask=0)
)

# Apply the data pipeline to our train and test sets
train_stream = data_pipeline(stream(train_data))
test_stream = data_pipeline(stream(test_data))


# Define and run the training loop
loop = training_loop(ReformerLM, train_stream, test_stream)
loop.run(20660) # Change this depending on desired number of steps
```

D Code: Overall System

Note 1: The objective response retrieval model (defined in appendix B) has not been included, to avoid repetition. Note 2: Again, part of the code has been extracted from [9].

```
import speech_recognition as sr
import os
import tensorflow as tf
import tensorflow_hub as hub
from googlesearch import search
import urllib.request
from bs4 import BeautifulSoup
from transformers import BertTokenizer, BertForQuestionAnswering
import trax
from trax import layers as tl
import numpy as np
import pyttsx3

VOCAB_FILE = 'en_32k.subword'
VOCAB_DIR = 'path/to/vocab/file/'

# Define the path to the Google Cloud credentials, to be able to use
# the speech_recognition method recognize_google_cloud
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'path/to/cloud/credentials'

# Initialize a recognizer and a microphone
r = sr.Recognizer()
mic = sr.Microphone()

# Initialize pyttsx3 and configure it
engine = pyttsx3.init()
engine.setProperty('rate', 125)
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[1].id)


def listen_to_user(microphone, recognizer):
    """Main listening loop"""
    with microphone as source:
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)
    return recognizer.recognize_google_cloud(audio)

def is_subjective(model, sentence):
    """Predict if the sentence is subjective or not using the
    understanding model"""
    prediction = model.predict(embed([sentence]))[0][0]
    if prediction < 0.5: # Subjective
        return True
    else: # Objective
        return False


def tokenize(sentence):
    """Tokenize a given sentence"""
    return list(trax.data.tokenize(iter([sentence]), vocab_file=VOCAB_FILE,
         vocab_dir=VOCAB_DIR))[0]

def detokenize(tokens):
    """Detokenize a list of given tokens"""
    return trax.data.detokenize(tokens, vocab_file=VOCAB_FILE, vocab_dir=VOCAB_DIR)


def ReformerLM_output_gen(ReformerLM, start_sentence, temperature, tokenize=tokenize):
    """Create a generator which will yield the next symbol generatd by the model"""
    input_tokens = tokenize(start_sentence)
```

```python
    # Add batch dimension to array
    input_tokens_with_batch = np.array(input_tokens)[None, :]

    # call the autoregressive_sample_stream function from trax
    output_gen = trax.supervised.decoding.autoregressive_sample_stream(
        ReformerLM,
        inputs=input_tokens_with_batch,
        temperature=temperature
    )
    return output_gen

def generate_dialogue(ReformerLM, model_state, start_sentence, max_len, temperature):
    """Generate a dialogue for a given start sentece"""
    delimiter_1 = 'Person 1: '
    delimiter_2 = 'Person 2: '

    sentence = '' # Initialize the output
    counter = 0
    result = [tokenize(': ')] # Output tokens
    ReformerLM.state = model_state # reset the model state when starting a new dialogue
    output = ReformerLM_output_gen(ReformerLM, start_sentence, vocab_file=VOCAB_FILE,
         vocab_dir=VOCAB_DIR, temperature=temperature)


    # Generate new tokens until the maximum length is reached
    for o in output:
        result.append(o)

        sentence = detokenize(np.concatenate(result, axis=0))
        if sentence.endswith(delimiter_1):
            sentence = sentence.split(delimiter_1)[0]
            print(f'{delimiter_2}{sentence}')
            sentence = ''
            result.clear()

        elif sentence.endswith(delimiter_2):
            sentence = sentence.split(delimiter_2)[0]
            print(f'{delimiter_1}{sentence}')
            sentence = ''
            result.clear()

        counter += 1

        if counter > max_len:
            print(sentence)
            return sentence

def extract_response(output_sentence):
    """From a generated conversation, only return the first sentence
    (both with and without the leading delimiter)"""
    delimiter1_pos = output_sentence.find('Person 2:')
    delimiter2_pos = output_sentence.find('Person 1:')
    if delimiter1_pos < delimiter2_pos:
        response = output_sentence[delimiter1_pos+len('Person 2:'):delimiter2_pos]
        response_and_delimiter = output_sentence[delimiter1_pos:delimiter2_pos]
    else:
        response = output_sentence[delimiter2_pos+len('Person 1:'):delimiter1_pos]
        response_and_delimiter = output_sentence[delimiter2_pos:delimiter1_pos]
    return response.lstrip(), response_and_delimiter

def clean_context(context):
    """If more than two sentences are kept inside context, keep the last two"""
    while len(context) > 2: # Remove until length is 2
        context.pop(0)
    return context
```

```python
def loop():
    """Main loop that will allow the user to have a conversation with the system"""
    context = [] # Initialize the context list
    while True: # Keep having the conversation until user stops the program
        user_input = listen_to_user(mic, r)
        if is_subjective(understanding_model, user_input.lower()): # Subjective statement
            # Set delimiter and introduce the new input in the context
            if len(context) == 0:
                subj_input = 'Person 1: ' + user_input
                context.append(subj_input)
            else:
                delimiter = 'Person 1: ' if 'Person 2:' in context[-1] else 'Person 2: '
                input_with_delimiter = delimiter + user_input
                context.append(input_with_delimiter)

            # If there are more than two sentences in the context, keep the last two
            context = clean_context(context)
            # If there are two sentences in the context, combine them for the input
            if len(context) != 1:
                subj_input = ' '.join(context)

            output = generate_dialogue(ReformerLM=model, model_state=STARTING_STATE,
                                       start_sentence=subj_input, vocab_dir=VOCAB_DIR,
                                       vocab_file=VOCAB_FILE, max_len=100, temperature=0.2)

            response, response_and_delimiter = extract_response(output)
            context.append(response_and_delimiter)
        else: # Objective question
            user_input = user_input + "?"
            engine.say('I will look it up, please give me a few seconds')
            engine.runAndWait()
            response = answer_question(user_input)
            engine.say('The answer that I found is')

        engine.say(response)
        engine.runAndWait()


# Load the understanding model and the embedding layer
understanding_model = tf.keras.models.load_model('second_understanding_model.h5')
embed = hub.load(handle="https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1")

# Load the trained reformer model
def attention(*args, **kwargs):
    kwargs['predict_mem_len'] = 120 # max length for predictions
    kwargs['predict_drop_len'] = 120 # never drop old stuff
    return tl.SelfAttention(*args, **kwargs)
shape11 = trax.shapes.ShapeDtype((1, 1), dtype=np.int32)
model = ReformerLM(
    vocab_size=33000,
    n_layers=6,
    mode='predict',
    attention_type=attention,
)
model.init_from_file('model/model.pkl.gz', weights_only=True, input_signature=shape11)
STARTING_STATE = model.state

loop() # Run the system loop
```

E Activity Diagram for the Query Context Extraction

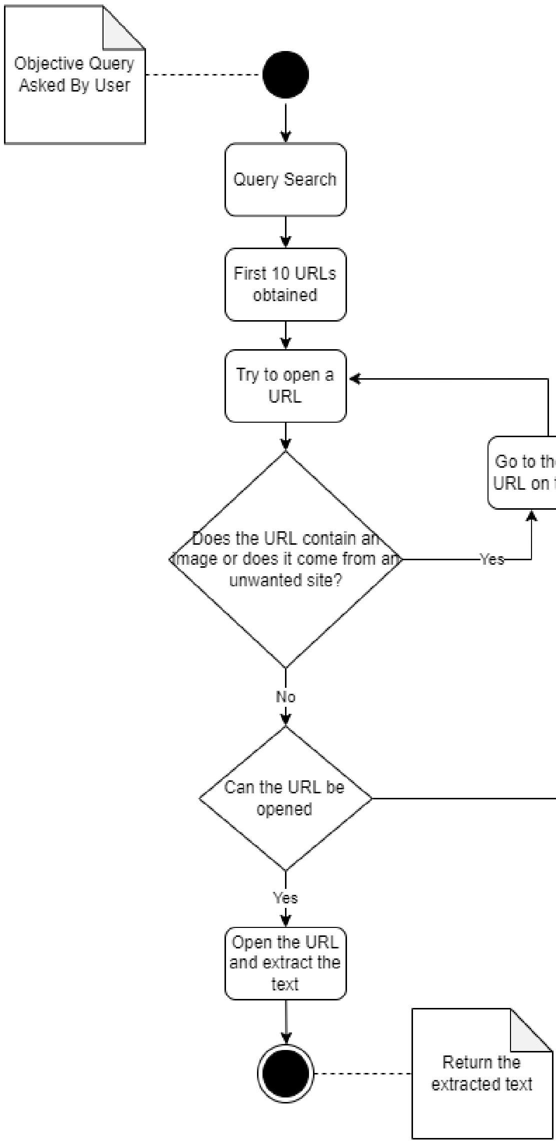


Figure 14: Query Context Extraction