**Empathic Companion: “An AI-Powered Chatbot for Emotional Support”**

**Group Name: Group A**

**Group Members:**

|  |  |  |
| --- | --- | --- |
| **First name** | **Last Name** | **Student number** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Submission date:**

Table of Contents

[**ABSTRACT** 4](#_Toc173895743)

[**INTRODUCTION** 4](#_Toc173895744)

[**PROBLEM STATEMENT** 5](#_Toc173895745)

[**OBJECTIVE** 5](#_Toc173895746)

[**LITERATURE REVIEW** 6](#_Toc173895747)

[**METHODS** 6](#_Toc173895748)

[Data Collection 7](#_Toc173895749)

[Data Preprocessing 7](#_Toc173895750)

[**SENTIMENT AND EMOTION DETECTION MODEL DEVELOPMENT (SED)** 10](#_Toc173895751)

[⁠ ⁠**CHATBOT ARCHITECTURE** 10](#_Toc173895752)

[TensorFlow, Keras, LSTM-Chatbot Architecture (non-transformer based) 11](#_Toc173895753)

[BERT /GPT-2 Chatbot Architecture (transformer-based) 13](#_Toc173895754)

[**CHATBOT RESPONSE GENERATION** 14](#_Toc173895755)

[TensorFlow, Keras, LSTM-Chatbot Architecture (non-transformer based) 14](#_Toc173895756)

[BERT /GPT-2 Chatbot Architecture (transformer-based) 14](#_Toc173895757)

[**GRAPHICAL USER INTERFACE** 14](#_Toc173895758)

[**DEPLOYMENT** 16](#_Toc173895759)

[**ETHICAL CONSIDERATIONS** 16](#_Toc173895760)

[**RESULTS** 16](#_Toc173895761)

[Sentiment and Emotion Detection 16](#_Toc173895762)

[Naïve Bayes 17](#_Toc173895763)

[Random Forest Classifier 17](#_Toc173895764)

[Support Vector Machine 18](#_Toc173895765)

[Chatbot (non-transformer based) 18](#_Toc173895766)

[GPT-2 Chatbot (transformer based) 24](#_Toc173895767)

[**CONCLUSIONS AND FUTURE WORK** 25](#_Toc173895768)

[**REFERENCES** 26](#_Toc173895769)

[Figure 1: Data visualization of the distribution of sentence sequence length using Seaborn’s boxplot 9](#_Toc173895770)

[Figure 2: Chatbot Architecture 10](#_Toc173895771)

[Figure 3: Loss curve plotted using Matpotlib’s plot method 12](#_Toc173895772)

[Figure 4: Graphical User Interface 15](#_Toc173895773)

[Figure 5: Classification report of Naive Bayes 17](#_Toc173895774)

[Figure 6: Classification report of Random Forest Classifier 17](#_Toc173895775)

[Figure 7: Classification report of Support Vector Machine 18](#_Toc173895776)

[Figure 8: Loss curve for training session run 2 20](#_Toc173895777)

[Figure 9: Loss curve for training session run 4 21](#_Toc173895778)

[Figure 10: Loss curve for training session run 10 22](#_Toc173895779)

[Figure 11: Loss curve for training session run 14 23](#_Toc173895780)

[Figure 12: Loss curve for training session run 17 23](#_Toc173895781)

[Figure 13: Interaction with Chatbot 24](#_Toc173895782)

[Table 1: A matrix showing model performance during training 19](#_Toc173895783)

# **ABSTRACT**

Addressing loneliness and the need for emotional companionship are some of the most widespread challenges in our society; our project addresses these widespread assumptive issues by developing an "Empathic Companion": “An AI-powered Chatbot for Emotional Support.” The project aims to reconstruct how users communicate with their digital companions. Getting help by traditional means has limitations in terms of availability and accessibility. Getting this support is sometimes difficult due to a lack of confidence or stigma. Therefore, to help those who do not have this kind of support, it has been vital to devise a chatbot, that is, an interactive program able to suggest a valuable and empathetic conversation to its user so that he/she will have a companion.

Our chatbot is built based on cutting-edge advancements in Artificial Intelligence (AI) and Natural Language Processing (NLP), which allows it to become a user's helpful friend, interlocutor, and assistant. The primary and perhaps foremost goal is to gain knowledge and establish an environment that can engage users and, at the same time, act as a virtual comforter. Utilizing advanced machine learning algorithms helps the bot develop the best and most comfortable way to interact. Indeed, there is a strong need for tools that offer emotional support and companionship so that our mental well-being can be further improved.

The empathic chatbot can be considered a reliable source to promote the user’s well-being through an immediate, persistent, non-judgmental point of support and a friend for its user. Moreover, this project is not only thought of as a tool with immediate advantages for the users, but mostly it can be considered a specific contribution to the generic field of AI-based companionship solutions, and as such, can pave the way to future inventions that try to tackle, scalable and accessible, the issue of addressing a wide range of emotional support tools.

# **INTRODUCTION**

Since introducing the *World Wide Web(www.)* to the public, each technological leap has resulted in an increasingly fast-paced and demanding life. According to (Hillyer, 2020), since the start of the millennium (year 2000), there have been 740 million cell phone subscriptions worldwide; since then, that number has exceeded 8 billion. This example demonstrates the magnitude of impact and reach that technology has today; many activities once done by a human, even as little as 5 years ago, are now done by some technological offshoot.

Home or personal “chatbots” are one such offshoot and have established themselves for many as a necessity to living an optimum life. Apple’s Siri and Amazon’s Alexa can schedule appointments, give directions, select and create music playlists, recommend and select television shows to watch and conduct cursory or extensive research on topics of your choice. Humans have, without knowing, become quite acquainted and comfortable with engaging with artificial agents and have confidence in these personal “chatbots” to execute the tasks given even better than their human counterparts.

The social effects on life are apparent, but the economic impacts are sometimes noticed too late. As mentioned earlier, life is becoming increasingly fast-paced due to these technological advancements. Jobs are no longer confined to a set time of the day or location, and with each new development, the expectations for productivity and efficiency yield, resulting in increased tasks for individuals and, in some cases, loss of jobs.

A recent study by HR consulting firm Robert Half reported that more than 4 out of every 10 Canadian professionals felt ‘burnout,’ and 52% reported that this was due to heavy workloads and understaffed teams (Barghiel, 2024). Also, as of June 2024, 1.6 million Canadians were reported as currently unemployed, which is projected to continue to rise ("Labour Force Survey, June 2024", 2024).

This creates the perfect storm for many individuals to feel disconnected and overwhelmed with life and could result in various mental health challenges. According to (Stephenson, 2023), 6 out of every 10 persons who meet the criteria for some type of mental health challenge, 57% do not receive some type of counseling or psychotherapy.

This presents an opportunity for developing an artificial empathetic companion to serve as a preliminary form of counseling. However, counseling legally is an activity that a licensed professional conducts. (American Psychological Association, 2018) has defined counseling as “*professional assistance in coping with personal problems, including emotional, behavioral, vocational, marital, educational, rehabilitation, and life-stage (e.g., retirement) problems. The counselor uses such techniques as active listening, guidance, advice, discussion, clarification, and the administration of tests”*.

The development of this artificial agent is not to replace a counselor but to provide a safe space for individuals who are suffering from various mental health issues to discuss their issues with an agent who actively listens and engages in thoughtful discussion. This will serve as a preliminary phase in providing help to those with mental health issues and will direct individuals to licensed professionals or institutions for further support if needed.

# **PROBLEM STATEMENT**

Almost 600 thousand (14%) Canadians have indicated that they perceive their mental state to be fair or poor (Statista, 2024), and often, accessibility to resources for counseling, companionship, and even social support is non-existent. Traditional methods of counseling and therapy, while effective, often face limitations such as but not limited to cost, scheduling conflicts, and social stigma. There is a great need for a solution to limitations that healthcare professionals in the mental health industry face that offers a low barrier for entry can be accessed at any time, and provides safe, nonjudgmental, and instant preliminary help when needed.

The "Empathic Companion: An Interactive Chatbot for Emotional Support" aims to fill this gap by leveraging the capabilities of AI and NLP. The chatbot aims to converse with consumers compassionately and provide them company and empathetic responses. This innovative solution not only addresses the immediate emotional needs of users but also contributes to their overall mental well-being.

# **OBJECTIVE**

The primary objective of this project is to design an application that can entertain users and serve as a valuable companion by applying Artificial Intelligence (AI) and Natural Language Processing (NLP) technologies. This project will focus exclusively on AI-based systems, utilizing machine learning algorithms to create an optimal user experience. By harnessing the power of AI, we aim to develop a chatbot capable of providing the most effective and supportive interactions for users.

# **LITERATURE REVIEW**

Constructing a chatbot is not a novel project idea, even for the problem this task is being undertaken to solve. Notwithstanding this, simulating an authentic and genuine human-like interaction is still challenging and can be approached from several angles. This project attempts to build a chatbot architecture integrated with a sentiment analysis model to improve the bot's ability to have nuanced liked responses with the target audience.

For the architecture, one path was to construct the chatbot with transformers like BERT and GPT-2 as the foundation of the build. The other path used a seq-seq model similar to the (Vinyals & Le, 2015) approach described in their work titled *A Neural Conversational Model.* Their model is based on a recurrent neural network that processes inputs one token at a time (Vinyals & Le, 2015). Despite LSTMs being a simple approach to machine language modeling, it has limitations, especially when handling long sequences.

According to (Vaswani et al., 2023), due to the sequential nature of LSTMs and their lack of parallelization in processing in training, this becomes a significant challenge as sequence lengths increase and memory limits batching across training examples. This is where the transformer-based architecture has excelled; with attention mechanisms within its arsenal, the transformer can also execute training tasks in parallel.

The execution of both a transformer and non-transformer model should demonstrate, based on literature, the increased ability of the former to generate coherent responses with less intensive architectural design.

# **METHODS**

This capstone project will explore and attempt to build an empathetic chatbot to resolve the limitations of traditional methods of counseling and therapy, particularly access to counseling, by providing a free, safe, non-discriminatory space for thoughtful discussions. The project will require engaging with various technologies such as but not limited to BERT, GPT-2, TensorFlow, Keras, NLTK, and Torch.

The framework established to guide the development process of the empathetic chatbot consisted of 12 steps: Data Collection, Data Preprocessing, Sentiment and Emotion Detection Model Development (SED), Implementation of NLP techniques, Chatbot Architecture, Chatbot Response Generation, Graphic User Interface Development, Model Integration, Testing and Evaluation, Deployment, Ethical Considerations and Documentation. The framework was designed to assist with identifying critical milestones of the project and establishing key components that could be built separately and later combined for the final result.

Three critical components of the project are the Sentiment and Emotion Detection Model, Chatbot Architecture, and Testing and Validation; the first two components are the fundamental pillars of the chatbot's ability to correctly assess the user’s input and provide an appropriate response. Ideally, This response should capture the correct tone and consider the user's emotional state according to the measure of emotion in the user input. To support the training, testing, and evaluation of these two components, two datasets were retrieved from Cornel’s University Convokit website, and the other was pulled from a GitHub repository hosted by Nia Schimnoski (aka Niaschim).

The sentiment and emotion detection model was built using the GitHub repository data, consisting of 40,000 tweet records and emotion labels for each tweet. The dataset was not cleaned and would require preprocessing, which will be discussed below. For the chatbot architecture, two approaches were experimented with; one architecture was built on BERT and GPT-2 transformers, and the other used TensorFlow and Keras. The former was trained using the dataset retrieved from the GitHub repository, while the latter was trained on the Cornell Movie-Dialogue Corpus from the Cornel’s University Convokit website. The Movie-Dialogue Corpus consisted of over 300,000 records and 8 features.

The models were evaluated using the following techniques: Sentiment and Emotion Detection Model, *Precision, Accuracy, Recall,* and *Confusion Matrix* metrics; Chatbot Architecture, evaluated using *Good Fit Learning Curves* and *Perplexity*.

## Data Collection

The initial project design considered using tweets as a reasonable data source to train the SED model. This was primarily because of the variability in tweets, the complexity of human expression in only a few characters, and the ability to express a wide range of emotions within a limited character count. However, it was reasoned that the time allotted for the execution of this project, being 12 weeks and based on the difficulty as well, was attainable but ambitious. This, coupled with the time required to pull a large number of tweets from X’s API, would have increased the project time, so a dataset of tweets of a suitable size that had not been preprocessed was identified and retrieved.

This dataset comprised 40,000 records and 3 features: tweet\_id, sentiment, and *content*. The sentiment feature captured the emotion label given to each tweet, which was stored in the feature content.

The inclusion of the Cornell Movie-Dialogue Corpus was later identified during the Chatbot architecture and training phase. This was used to help train the models to improve the responses generated from the chatbot. This dataset consisted of 304,713 records and 8 features. The features capture essential information for each dialogue, such as but not limited to conversation\_id, text(conversation), speaker (second speaker), and reply-to(first speaker).

## Data Preprocessing

A class was built to handle all the steps required for general NLP tasks. The class created was called *TextWrangler* and contained the following methods for executing text wrangling tasks: remove\_digit, removes numbers from the text; remove\_punctuations, remove punctuations from the text; clean\_text, removes HTML tags and special characters, and convert text to lowercase; word\_lengthing, contracts words to original spelling (i.e., ‘helloooo’: ‘hello’); tokenize, applies NLTk word\_tokenize and turns text into a list of work tokens; remove\_stopwords removes from text stopwords in NLTK stopword corpus.

This class was built on the following Python libraries: nltk, re, sklearn, wordcloud, string, *and typing.*

***X (formerly known as Twitter) Dataset***

Data preprocessing consisted of handling missing values and text wrangling techniques. A check for missing values was conducted using a combination of two methods, isna() and mean(), from the pandas library; this revealed that the percentage of missing values in the dataset was 0. Considering this, the next step was to clean the text by removing unwanted characters from the data.

It was determined that emojis should not be removed but remain in the text as they carry essential information that could impact the ability of the model to determine the correct emotion or sentiment. The tweets were processed and had HTML tags, special characters, digits, punctuations, and stopwords removed while words were contracted using the method word\_lengthening from the class created. This was achieved primarily using *Regular Expression Operations* supported by the *re* library in Python and filtering out stopwords downloaded from the *nltk stopword corpus*.

Despite the dataset with the tweets classified with different emotion labels, some tweets' labels were empty. It was presumed that this term was used to handle missing values. Thus, it was decided to re-classify the emotion of each tweet using the *Hugging Face transformer* library. This library facilitated emotion detection to classify the emotion in each tweet using the pipeline method, specifically the *'j-hartmann/emotion-english-distilroberta-base’* model for text classification. These technologies were incorporated in a function created and named *‘detect\_emotion.’*

***Cornell Movie-Dialogue Corpus***

Similar to the X dataset, the movie-dialogue dataset was checked for missing values using the same technique, which identified two features, reply-to and timestamp, having 27% and 100% missing values, respectively. The timestamp feature was not required as input for the chatbot architecture, so it was not included in the final transformation of the dataset. However, upon closer inspection of the missing values, it was observed that the term None was included in the reply-to, signaling the start of a new dialogue. This probe was achieved using a combination of the methods isna() and any(setting the axis parameter to 1), which returned all the records with missing values. To handle these missing values, the term ‘no comment’ was inserted using the fill () method from the pandas library.

The data was parsed and transformed into a list of tuples, with each tuple containing a question/statement and response pairing. Through the application of two user-defined functions, replace\_contractions and preprocessing, the data was further processed with the removal of punctuations, digits, emojis, memorable characters, and word contractions replaced (i.e., “who have”:” who have”). Additionally, the function preprocessing() utilized the methods nlp(), lemma\_, and *is\_alpha* from the *spacy* library to tokenize the sentences and apply lemmatization. The lemmatization technique reduces words to the root form in the context of building a chatbot architecture; this is useful as it reduces the vocabulary size the model will have to learn.

The application of lemma\_ from spacy was chosen because it requires less computational power than *WordNetLemmatizer* from the *nltk* library. This proved helpful as the size of the dataset was considerable and demanded considerable computational power to process and train the model; this will be further discussed under the chatbot architecture development.

The dataset was also assessed for outliers, more specifically as it pertains to sentence sequence length. This was a critical step in the preprocessing activities primarily because the information gained from this step significantly affected the design of the chatbot architecture via padding of sequences and dimensionality, as well as the ability for the model to train on the data. The outliers were identified using two techniques, the first by way of data visualization using histogram and boxplot (see figure 1) from the seaborn library and the second by calculating the upper and lower boundaries using the InterQuartile Range (IQR) with the method percentile from the NumPy library.

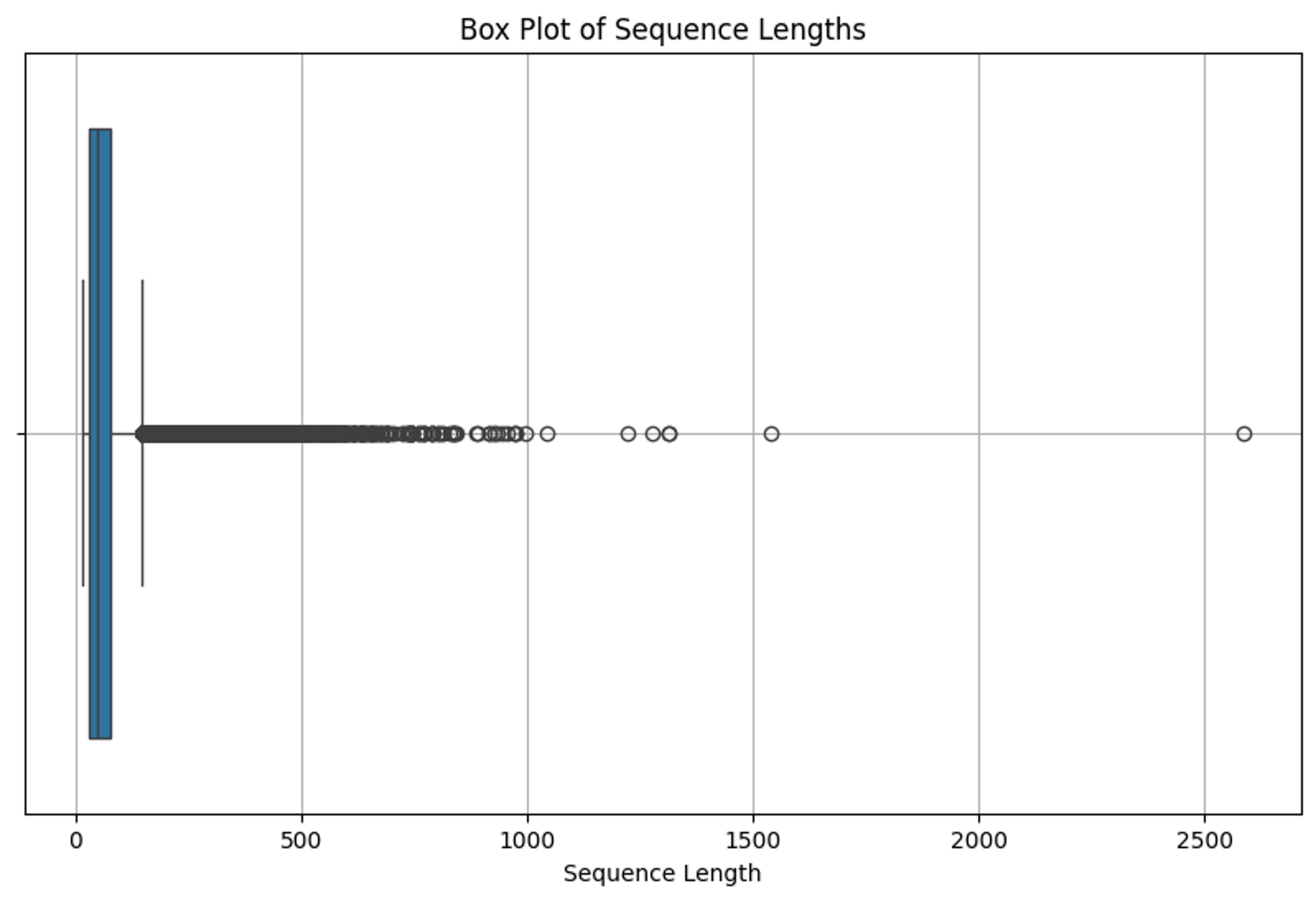


Figure 1: Data visualization of the distribution of sentence sequence length using Seaborn’s boxplot

From the boxplot observation, many sequence lengths ranged from under 200 to approximately 2700. This was corroborated based on the summary measurements calculated, where the upper boundary beyond which outliers are observed in the boxplot is, in fact, a sequence length of 147.5. Also, the blue box in Figure 1 represents the area where most of the data is distributed; this area also represents the IQR value, which is 46. Therefore, the middle 50% of sentence sequences have a length of 46 words.

To assist the model during the learning phase, a ‘start’ and ‘end’ tag was added to each sequence to distinguish the start of a new sequence from the end of one. Before commencing the chatbot architecture's construction, a validation check was performed to ensure all changes were executed as intended. Under this, the dataset was split into a training and validation set using the train\_test\_split method from the sklearn library, with an *80%* split for training and the remaining *20%* for validation.

# **SENTIMENT AND EMOTION DETECTION MODEL DEVELOPMENT (SED)**

The design concept for this component is based on extracting the sentiment from a text and correctly classifying the emotion of the text. To accomplish this, the model’s ability to determine the sentiment score of the text was built using the Valence Aware Dictionary and sentiment Reasoner (Vader or vaderSentiment) from the *nltk* library. According to (*Python et al., 2020*), *Vader* is a lexicon and rule-based sentiment analysis tool attuned explicitly to sentiments expressed in social media. This was why this method was used on the X (formerly Twitter) dataset, and special characters like ‘emojis’ were not removed during the cleaning process.

The model's ability to correctly classify emotion was built on a classification model; three algorithms were assessed using train and test data to determine which performed the best. The algorithms selected were Naive Bayes (MultinomialNB), RandomForestClassifier, and Support Vector Machine (SVM), all taken from the *sklearn* library. Before the data is passed to the classification models, they are transformed into an array of numbers that the model can interpret. This was achieved using the Term Frequency Inverse Document Frequency (TFIDFVectorizer) method from the *sklearn* library.

The performance of the models was assessed by generating the classification report and the confusion matrix, both methods of the *sklearn* library. The classification report provides the values for key metrics like precision, recall, f1-score, support, and accuracy. The confusion matrix explains how well the models classify true positives, false positives, and false negatives.

# ⁠ ⁠**CHATBOT ARCHITECTURE**

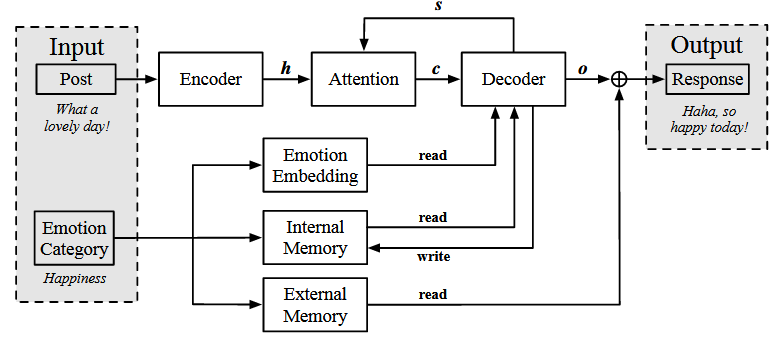


Figure : Chatbot Architecture

## TensorFlow, Keras, LSTM-Chatbot Architecture (non-transformer based)

Long Short-Term Memory (LSTM) is a network of recurrent neural networks (RNN) that are often applied as the medium through which sequence-to-sequence tasks, like those done by a chatbot or language translation tasks, are executed (Rosemary Collins, 2023). The LSTM used to build this chatbot architecture was accessed through the *TensorFlow* and *Keras* library packages for Python.

Initially, the chatbot architecture build consisted of two major components, the encoder and the decoder, which are the core elements of an LSTM build. The encoder transforms the input it receives to a higher dimensional space and then feeds that output as input for the decoder. The role of the decoder is to take this input, and in the context of this project, it sequentially, with a time step of 1, predicts the next word according to the first word in the input sequence.

Before the text data is passed to the encoder, it is tokenized by applying the *Tokenizer* method from the *Keras* library package. Tokenizer creates a list of ‘words or tokens, similar to the tokenizer from the nltk package, and assigns each word a unique number. This assignment of numbers to words creates a collection of vocabulary from the text data; coincidentally, the values are stored in a dictionary of key-value pairs. This dictionary can be accessed through the Tokenizer().word\_index method. This method, along with the len method (to find the length of a collection), was used to generate the vocabulary size, a parameter for the embedding layer in both the encoder and decoder architecture.

After the data has been tokenized, each sequence (token) of the tokenized list is padded using the method *pad\_sequences* from the Keras library. This method ensures each sequence is of equal length; for this project, the parameter padding was assigned the value post, which means each sequence will be made equal by adding zeros to the end until the required length is achieved. Another import parameter for the *pad\_sequences* method was *maxlen; this* indicated the maximum length each sequence should be padded to. Initially, this parameter was set to the length of the most extended sequence in both lists of responses; the data was prepared as statement/question and answer pairs.

However, this significantly increased the dimensional space and often resulted in *Graph execution error; in other words,* the machine did not have enough memory to process the request. A solution to this was to select a maximum length by assessing the distribution of sequence lengths and selecting the quartile, which reduces the dimensionality of the sequences but also captures the majority of sequences (excluding the outliers). Once executed, the padded sequences were transformed into a Numpy array using the *array()* method from the *Numpy* library. These now serve as the encoder and decoder inputs.

The structure of the encoder and decoder is similar. It follows the following framework: encoder input, decoder input, embedding layer, LSTM for encoder and decoder, encoder output and states, and decoder output and states. The encoder and decoder's embedding layers were constructed with input\_dim (the vocabulary size +1), output dimension (dimension of dense embedding), and *mask\_zero*. The output\_dim was initially set to 200 but was later changed through the experimentation phase. This parameter also impacted the dimension size and ability of the machine to train.

The encoder and decoder LSTMs it was constructed using the following parameters: units (dimensionality of the output space), return\_state (can either return the last state with the output or not), return\_sequences (can either return the last output of the sequence or the entire sequence) and dropout (this impacted how many units or neurons would be dropped). The first build did not include dropout but was later included to improve the model performance. These elements would have constituted the initial chatbot architecture; however, the responses were incoherent. Through further research, the architecture was adjusted to include an attention layer.

According to (Brownlee, 2022), “*The encoder-decoder architecture still achieves excellent results on a wide range of problems. Nevertheless, it suffers from the constraint that all input sequences must be encoded to a fixed length internal vector”.* This is where an attention layer is functional, as it releases the LSTM from needing to encode all input sequences into this fixed dimensional space. Theoretically, this should improve the model’s ability to interpret and predict (generate) the best coherent response.

The model is then defined to receive two inputs: the first input is the encoder\_input and decoder input (the values prior to going through the LSTM for encoder and decoder), and the decoder output after the attention layer is applied. The model is compiled with the following parameters: optimizer, *RMSprop* with learning rate defined; loss, *SparseCategoricalCrossentropy()* was the method selected; metrics and accuracy were selected. The defined model was fitted with the required inputs, and validation and call-back parameters were also defined.

Loss and accuracy curves were plotted for the training and validation runs to assess the model performance during training. These plots were executed and designed using the plot() method from the *Matplotlib* library package. The loss and accuracy values for both the training and validation runs were accessed from the model using the *history* method; the values were then passed to Matplotlib’s plot method, which created a line plot of the loss and accuracy values (see Figure 3).

A graph with blue and orange lines

Description automatically generated

Figure : Loss curve plotted using Matpotlib’s plot method

The model was further evaluated by calculating the *Perplexity* score and Bleu score post running on unseen data; this was calculated using the Perplexity and Bleu Score methods from Keras.metrics library package.

## BERT /GPT-2 Chatbot Architecture (transformer-based)

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language representation model that has achieved state-of-the-art results across various NLP tasks. In this project, BERT was fine-tuned for a multi-class sequence classification task specifically aimed at classifying emotions from Twitter data.

To prepare the data for training, categorical labels were converted into a numerical format using *LabelEncoder* from the *scikit-learn* library. This transformation is essential as it converts the string labels into integers, which are suitable for processing by the classification model. The training labels are fitted and transformed to create a consistent mapping, and the exact mapping is applied to the validation labels.

Next, the input text is tokenized using the *BertTokenizer* from the *transformers* library. Tokenization involves converting raw text into tokens that BERT can understand. The tokenizer processes the input texts by truncating or padding them to ensure a uniform length of 128 tokens. This step is crucial for maintaining consistency in the input dimensions.

A custom dataset class was created named *EmotionDataset*, which extends PyTorch's *torch.utils.data.Dataset*. This class is designed to facilitate efficient data loading and handling during training and evaluation. It encapsulates the tokenized inputs and the corresponding labels, providing a standard interface for accessing individual data points and the overall dataset length. By implementing the necessary methods, the custom dataset class ensures compatibility with PyTorch's DataLoader, which handles batching, shuffling, and loading data efficiently during training.

The training process is managed using the *Trainer* class from the *Transformers* library. A set of training arguments was defined that specify various hyperparameters and configurations, including the output directory for model checkpoints, the number of training epochs, batch sizes for training and evaluation, warmup steps, weight decay for regularization, logging configurations, and the evaluation strategy. These parameters are a must for controlling the training process and ensuring optimal performance.

The next step is the fine-tuning process involving initializing the *Trainer* with the BERT model, the defined training arguments, and the prepared datasets. The *Trainer* handles the entire training loop, including forward and backward passes, optimization, and evaluation at specified intervals. It uses the pre-trained BERT model and fine-tunes it to our specific task, achieving high accuracy in classifying emotions from text.

After training the model, the Fine-tuned BERT model and the tokenizer were saved to facilitate future use without retraining. The model and tokenizer were saved to a specified directory, ensuring they could be quickly loaded and deployed for inference or further fine-tuning. This step is crucial for preserving the trained parameters and configurations, enabling consistent and efficient model reuse.

# **CHATBOT RESPONSE GENERATION**

## TensorFlow, Keras, LSTM-Chatbot Architecture (non-transformer based)

In order to generate the chatbot response, an inference encoder/decoder architecture was created, which modeled quite closely the build of the training model. This was important as the saved model was designed to receive two inputs. The architecture for the inference model also needed to be designed to receive two inputs of similar shape and dimension.

Therefore, tokenizing and padding of user inputs should be done using the same parameters set in the training phase. The significant difference was with how the decoder output sequence is generated; the stepwise approach was taken. In other words, the model considered 1 input from the sequence at a time and used that input to predict the next word in the sequence.

When the model predicts the next “word,” this is a sequence of numbers checked against the model’s vocabulary (found in Tokenizer.word\_index); when the number is located, it returns the associated word. This process is repeated until the model predicts the sequence of numbers for the token ‘end’ or the sequence length is greater than the max sequence length (which is defined during the training phase). The model has now created a response in a tokenized form; this is then returned to human-understandable language using the *Tokenizer.sequences\_to\_text method* from the *Keras* library.

## BERT /GPT-2 Chatbot Architecture (transformer-based)

The code here initializes the GPT-2 *model* and tokenizer using pre-trained weights. GPT-2 is a language model designed for generating coherent and contextually relevant text based on a given prompt.

The *generate\_response* method takes a text prompt (user input) and encodes it into a format that GPT-2 can process. This involves converting the text into token IDs using the GPT-2 tokenizer. The encoded input is then passed to the GPT-2 model, which generates text based on the input. The *generate* method of the model is used with parameters like *max\_new\_tokens* to limit the length of the generated text and *num\_return\_sequences* to specify the number of sequences to return. The *pad\_token\_id* parameter ensures that padding is applied where necessary. The generated token IDs are decoded back into human-readable text using the GPT-2 tokenizer. The *skip\_special\_tokens=True* parameter ensures that unique tokens like padding tokens are omitted from the final output. To make the response more readable, it is truncated at the first period, ensuring it is limited to a single sentence.

Chatbot continuously prompts the user for input until a termination command (such as "exit," "quit," or "bye") is received. For each user input, the chatbot generates a response using the *generate\_response* function and returns it.

# **GRAPHICAL USER INTERFACE**

The chatbot application's graphical user interface (GUI) is crafted using Tkinter, a versatile library for creating desktop applications in Python. The main application window is designed with the title "*ChatBot*" and dimensions of 500 x700 pixels, providing a structured environment for user interaction. Within this window, a *Frame* widget displays the conversation history. This frame, which features a light gray background, is placed inside the background image label and is designed to adjust its size according to the window dimensions. A Canvas widget is used within the conversation frame to handle potentially lengthy conversations, allowing for vertical scrolling. This canvas is paired with a *Scrollbar* widget, enabling users to navigate the chat history seamlessly.

Another frame is used inside the canvas to display messages. This frame organizes and contains individual messages, ensuring a clean and structured layout. At the bottom of the window, a separate *Frame* holds an *Entry* widget for user input and a *Button* for sending messages. The entry widget allows users to type their messages, while the send button triggers the message-sending process and generates a response from the chatbot.

The application includes a method for handling user input and updating the chat display. When a user sends a message, it is added to the conversation area, and a response is generated using the chatbot's language model. The conversation area is updated to show both the user's message and the bot's reply, with messages aligned appropriately to distinguish between the user and the bot.

The integration process ensures that user inputs and bot responses are seamlessly managed. The application’s backend handles the generation of responses, while the frontend GUI updates in real-time to reflect these responses. The code handles any warnings and limits the response to one sentence. This integration provides a cohesive chat experience, combining an intuitive user interface with advanced language generation capabilities.

Overall, the GUI is designed to offer a user-friendly chat experience, integrating Tkinter widgets to create an interactive and visually appealing interface.

A screenshot of a phone

Description automatically generated

Figure : Graphical User Interface

# **DEPLOYMENT**

The chatbot post-completion could be deployed using Microsoft Azure cloud services. According to (Microsoft, 2024), the following steps must be executed to deploy the model as an online endpoint successfully. First, ensure that all the prerequisites are in place and accessible; these include but are not limited to Azure machine learning workspace (establishing the kernel and creating a handle to the ml workspace), compute resources, and adequate quotas. The second step involves registering the trained model and confirming registration.

Once that is completed, the next step is to create and define endpoints (HTTPS path) where the inference model will be hosted. After completing this, the model will be deployed and tested with data to assess its operation. The model can later be scaled based on traffic needs, which in this project’s scenario would be based on user uptake.

# **ETHICAL CONSIDERATIONS**

Often overlooked in technology and engineering projects, ethics is a critical component of every build. Ethics, as defined by (Singer, 2024), is “ the discipline concerned with what is morally good and bad and morally right and wrong. The term is also applied to any system or theory of moral values or principles”. Therefore, in considering the moral implications of this chatbot, the bot's design must consider the user's safety.

Despite these measures not being implemented in the build, the application's ability to safely recognize, remove, and encrypt personal details entered into the conversation is critical. This also becomes even more important when cloud integration is being considered for deployment. The problem this chatbot is trying to solve falls within the realm of the medical field. Therefore, ethical considerations must account for best practices regarding patient care and regulatory requirements for handling patient information.

This bot is not designed to provide a medical diagnosis but merely provide a safe environment where those suffering from various mental health challenges and needing someone to speak with could engage as a first layer. In order to provide a safe digital environment, attempts must be made to model the security of a doctor's or therapist's office. This will account for how the conversations are stored and managed. A rudimentary but fundamentally important concept is selling user details to marketing companies for profit.

This is an example within the medical space where this would more often than not be considered unethical.

# **RESULTS**

## Sentiment and Emotion Detection

The sentiment analysis is done with the help of the VADER module from NLTK, and Emotion is identified from the Twitter responses datasets using the J-Hartmann emotion detection model.

After the emotion is identified, the emotion is taken as the target variable, and the Twitter responses are fed to the models to predict the emotion of an out-of-train set and unseen text.

Three classification models, Naive Bayes Classifier, Random Forest Classifier, and Support Vector Machine, were implemented to classify the text response into emotion labels. The classification report and confusion matrix were used as metrics to assess the performance of the models.

### Naïve Bayes

The Naive Bayes Classifies model has performed very well in classifying anger and fear emotions, which can be observed from the classification report. The confusion matrix of the model also shows high precision and better actual positive cases for joy, neutrality, and sadness. However, failing to classify the other emotions. This may be attributed to the fact that the key phrases in joy, neutral, and sadness are more repetitive and easily identified, and the text frequency of these emotions in the dataset is more considerable than that of other emotions. So, the model had picked up these effectively but failed to classify other emotions, leading to an accuracy of 60%.

A screenshot of a computer

Description automatically generated

Figure : Classification report of Naive Bayes

### Random Forest Classifier

The random forest classifier also continued the trend of the Naive Bayes model and predicted better for joyful, neutral, and sad emotions but was not consistent in classifying the other emotions. The accuracy is 64%, an improvement from the previous one. The improvement can be explained by the cluster of decision trees that Random Forest utilizes to identify more relations between the text data than by the simplistic approach of Naive Bayes.

A screenshot of a graph

Description automatically generated

Figure : Classification report of Random Forest Classifier

### Support Vector Machine

Support Vector Machine performed better out of the three models, with an accuracy score of 67%. The improvement is because of the SVM’s ability to find the optimum hyperplane that separates all the emotion classes. SVM is adept at making complex boundary decisions, which was utilized effectively in the text data to classify emotions.

**A screenshot of a graph

Description automatically generated**

Figure : Classification report of Support Vector Machine

## Chatbot (non-transformer based)

Once the architecture was complete and all configuration challenges were addressed, the model was trained and validated using an 80:20 train and test ratio of the data. Due to computational limitations, the model was not trained on all 304713 records but 40% of that amount; thus, the dataset size for model training was 121885 records.

To assist with the training process, the CUDA 11.2 and cuDNN 8.1.0 packages were installed to enable Tensorflow to access the machine's GPU for improved training processing time. A virtual environment was created to ensure no interference with the required libraries and packages to execute the training and any local libraries or dependencies on the machine used.

Despite these efforts, the first training run (see Table 1) was not executed due to a ResourceExhaustedError citing Graph execution error; in other words, there was insufficient graphical memory to execute the command. This pointed to the significantly sizeable dimensional space created by the chatbot architecture. Through some analysis of the parameters used in the architecture build and their contribution to the dimensionality of the inputs and outputs, *maxlen* was identified as one whose value needed revision.

The *TensorFlow* documentation describes *maxlen* as an integer and the maximum length of all sequences. Initially, this was how the maximum length for this architecture was found, and the maximum length of all sequences turned out to be 482. This was significant as the *maxlen* determines how long all sequences should be padded. The size or length of these sequences ultimately will impact the computer's resources when learning and predicting. The mean length of all the sequence lengths was used in the second run to address this.

Table : A matrix showing model performance during training

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Run** | **hidden\_dim** | **output\_dim** | **maxlen** | **Epoch** | **Batch size** | **Learning rate** | **Patience** | **Result** |
| **1** | 150 | 150 | 482 | 15 | 64 | 0.001 | 3 | **did not run** |
| **2** | 150 | 150 | 12 | 100 | 50 | 0.001 | 3 | **underfit** |
| **3** | 150 | 150 | 12 | 100 | 50 | 0.001 | 3 | **underfit** |
| **4** | 150 | 150 | 12 | 100 | 50 | 0.001 | 9 | **underfit** |
| **5** | 150 | 150 | 46 | 15 | 256 | 0.001 | 3 | **underfit** |
| **6** | 200 | 200 | 46 | 25 | 128 | 0.001 | 3 | **underfit** |
| **7** | 128 | 64 | 46 | 25 | 256 | 0.001 | 3 | **underfit** |
| **8** | 128 | 64 | 46 | 100 | 256 | 0.001 | 3 | **underfit** |
| **9** | 128 | 64 | 46 | 100 | 32 | 0.001 | 3 | **underfit** |
| **10** | 128 | 64 | 77 | 25 | 64 | 0.0005 | 9 | **overfit** |
| **11** | 128 | 64 | 77 | 25 | 64 | 0.0003 | 6 | **overfit** |
| **12** | 128 | 64 | 77 | 25 | 64 | 0.0002 | 5 | **slight overfit** |
| **13** | 128 | 64 | 77 | 25 | 64 | 0.0002 | 4 | **underfit** |
| **14** | 128 | 64 | 77 | 25 | 64 | .0001 | 5 | **good fit** |
| **15** | 128 | 64 | 77 | 50 | 64 | .0001 | 5 | **underfit** |
| **16** | 128 | 64 | 77 | 50 | 64 | .00007 | 5 | **min overfit** |
| **17** | 128 | 64 | 77 | 50 | 64 | .00005 | 5 | **Good fit** |

In the second run, the model was greatly underfitting, and it would complete its learning before the number of epochs set. For the second run, 100 epochs were set, but the model completed its run in 10 epochs, and while it was able to learn well on the training data, it did not learn or generalize well enough to achieve similar performance on the test data. This is demonstrated by the loss of the training data gradually decreasing over epochs; however, for the validation data, the loss was significantly higher than the training, decreasing minimally over several epochs (see Figure 8).

After the third attempt, the model was examined to identify what parameters would impact the model’s ability to learn for the entire epoch set. The low-hanging fruit was determined to be the patience parameter in the EarlyStopping method from the Keras library package. According to the official documentation, patience dictates how many epochs the training will continue, with no improvement. Considering how abruptly the model concluded the training, it was presumed this was due to the “aggressive” patience value of 3.

A graph of a model loss

Description automatically generated

Figure : Loss curve for training session run 2

However, the patience was set to 9 for the fourth run, and no change in model performance was observed. The model exhibited underfitting similar to the runs before and concluded the training at 8 epochs, less than the previous trials (see Figure 9). It seems the increase in patience resulted in a higher loss for the validation curve than in previous runs and a practically horizontal line for the training. Suggesting the model was not able to make any generalizations.

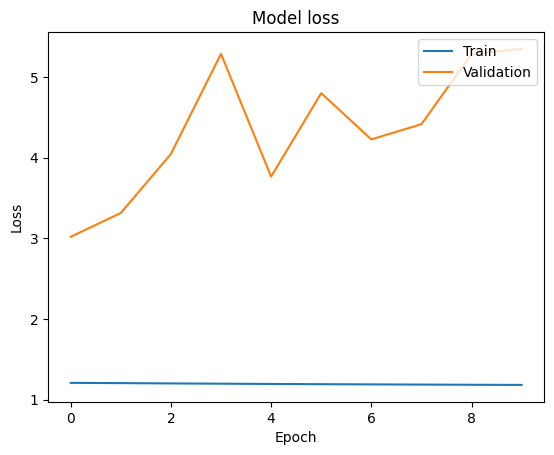


Figure : Loss curve for training session run 4

The following nine experiments (run 5 - 9) resulted in an underfit model. Throughout these trials, the maxlen was changed to the second quartile value for the sequence lengths, 46. This approach was taken after considering the distribution of the sequence lengths and recognizing that the outliers were significant in the count and size (see Figure 1). A closer examination of the boxplot distribution of the sequence lengths confirmed that the sequence lengths were skewed, specifically right-skewed, which is why the median value is higher than the mean. This value also represented 50% of the sequence lengths, so it was deemed an appropriate adjustment for training.

Throughout the trials, this type of manual hyperparameter tuning saw batch-size cycles through 64,50,256,128, and 32 before eventually settling on 64. Unlike batch\_size, *hidden\_dim* and *output\_dim* parameters were only changed three times, with the initial experiments using the same value for 150 and later 200. After switching *hidden\_dim* and *output\_dim* to 128 and 64, respectively, these integers remained for the remaining experiments.

The decision to use a hidden\_dim of 128 for the LSTM layer and an output\_dim of 64 for the embedding layer was mainly grounded in balancing computational resources and potential overfitting. Considering the size of my dataset and how much time was needed to run a training session, further tuning of the model was done by considering adjustments to the *maxlen*, learning rate, and patience. Up to experiment 9, the model could not train through the total number of epoch sets. Considering this, the *maxlen* was adjusted to reflect the third quartile, which is 77, for sequence length.

In addition to this, the batch\_size was changed back to 64, and the epoch was reduced to 25. This marked a turning point in the trials as it was observed that the *batch\_size* and *maxlen* both improved the model's performance significantly. Despite overfitting towards the end, both the training and validation curves had a point of convergence, which had not been achieved prior (see Figure 10).



Figure : Loss curve for training session run 10

At this point, the decision was made to keep the parameter values that achieved the loss curve in run 10 (see Figure 10), except the *learning\_rate* parameter of *Keras* *Optimizers*, particularly *RMSprop.* According to (Brownlee, 2020), deep learning neural networks are trained on a stochastic gradient descent algorithm, which estimates the error gradient for the current value in the model and updates the model weights using back-propagation. “The amount that the weights are updated during training is referred to as the step size or the ‘learning rate.’” reported (Brownlee, 2020). Simplistically, manipulating the learning rate can impact the model's ability to converge by dictating when the weights are updated.

The aim was to allow the model to learn sufficiently in training without increasing the epoch, as previously increasing the epoch did not yield improvement. So when Table 1 is observed, the learning rate was adjusted, which decreased incrementally from 0.001 to .0001, and patience, starting at 9, was reduced to as low as 4. From experiment runs 10-12 at 25 epochs, it was observed that the modeling was overfitting less with each adjustment. However, for run 15, when patience was set to 4, the model loss curve exhibited underfitting. At experiment run 14, an excellent fit curve was achieved (see Figure 11) with a patience of 5 and a learning rate of 0.0001.

Recognizing this, it was established that 5 was the ideal patience value, and the proceeding experiments sought to improve learning at a higher epoch (50). These experiments demonstrated how vital and powerful the learning rate is in the model’s ability to converge, and so it was the focus of further tuning at 50 epochs. As expected on first increasing the epoch to 50, the model loss curve exhibited underfitting.

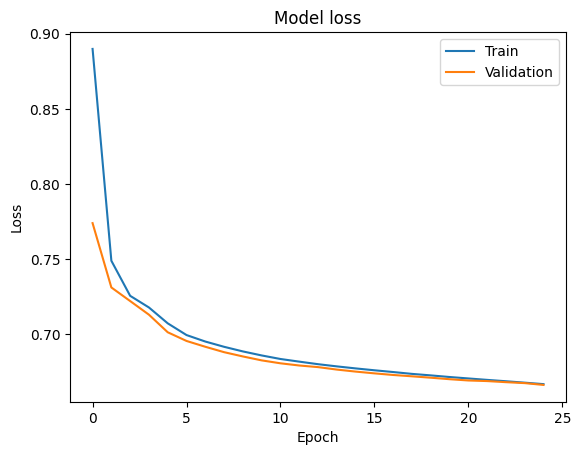


Figure : Loss curve for training session run 14

The subsequent two runs (16 and 17) were run with a learning rate of 0.0007 and 0.0005, which return a loss curve with minimal overfit and a best-fit curve, respectively (see Figure 12).

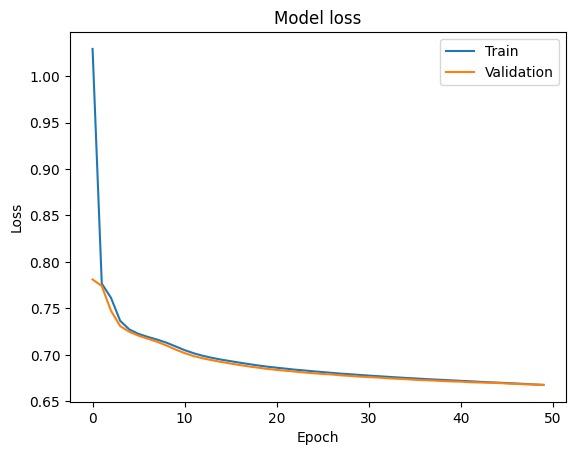


Figure : Loss curve for training session run 17

The model was saved at this point, and an attempt was made to evaluate it on new data to assess how well it predicts the most appropriate human-like response. A metric called perplexity was used to do this assessment, and it was calculated by finding the exponential loss. Ultimately, the chatbot was assessed to see what responses it would give when a user interacted with it. To simulate this, the inference model was created on top of the saved model in the training phase. Unfortunately, the chatbot response was incoherent even with simple inputs like “Hello” or “Hi, how are you?”.

## GPT-2 Chatbot (transformer based)

User input is tokenized and passed to the GPT-2 model, which generates text based on the input and passes it to GUI through integration.

***A screenshot of a chat

Description automatically generatedA screenshot of a chat

Description automatically generated***

Figure : Interaction with Chatbot

# **CONCLUSIONS AND FUTURE WORK**

The development of the chatbot presented many challenges, mainly surrounding the management of computational resources and hyperparameter tuning. Despite efforts to thoroughly train and methodically tune the hyperparameters, the model, in most instances, was either overfitting or underfitting. However, in experiment 17, an optimal state was achieved, confirmed by the loss curve exhibiting a good fit (see Figure 12). Despite this achievement, the model was challenged to return human-like responses for accurate word tests. This is indicative of the need for more refinement of the architecture.

To achieve the goal of a fully functional empathetic chatbot, there should be consideration for upgrading the architecture to include attention mechanisms or a hybrid model incorporating transformers like BERT and GPT-2/3. To enhance the scalability and reliability of our chatbot, a crucial next step is to deploy the system on a cloud platform, Microsoft Azure. Deploying on a cloud platform simplifies maintenance and updates. Continuous integration and continuous deployment (CI/CD) pipelines can be set up to automatically test and deploy new features or improvements, ensuring the chatbot is always up to date with the latest advancements.

# **REFERENCES**

Hillyer, M. (2020, November 18). *How has technology changed - and changed us - in the past 20 years?* Retrieved August 3, 2024, from <https://www.weforum.org/agenda/2020/11/heres-how-technology-has-changed-and-changed-us-over-the-past-20-years/>

Statistics Canada (2024, July 5). *Labour Force Survey, June 2024*. Retrieved August 3, 2024, from <https://www150.statcan.gc.ca/n1/daily-quotidien/240705/dq240705a-eng.htm>

Barghiel , N. (2024, June 6). *Why studies show burnout is getting worse: ‘Employees are exhausted’*. Retrieved August 3, 2024, from <https://globalnews.ca/news/10550099/canadian-workplace-burnout-survey/>

Stephenson, E. (2023, September 22). *Insights on Canadian Society Mental disorders and access to mental health care*. Retrieved August 3, 2024, from <https://www150.statcan.gc.ca/n1/pub/75-006-x/2023001/article/00011-eng.htm>

American Psychological Association (2018, April 19). *Counseling*. APA Dictionary of Psychology. Retrieved August 3, 2024, from <https://dictionary.apa.org/counseling>

American Psychological Association, S. (2024, February 15). *Percentage of Canadians who perceived their mental health as fair or poor from 2003 to 2022*. Retrieved August 3, 2024, from <https://www.statista.com/statistics/434065/share-of-canadians-perceiving-their-mental-health-as-fair-or-poor/>

Chang, J. P., Chiam, C., Fu, L., Wang, A., Zhang, J., & Danescu-Niculescu-Mizil, C. (2020). ConvoKit: A Toolkit for the Analysis of Conversations. Convokit. Retrieved July 17, 2024, from <https://convokit.cornell.edu/>

Python Software Foundation (2020, May 22). *VaderSentiment 3.3.2*. Python Package Index. Retrieved June 18, 2024, from <https://pypi.org/project/vaderSentiment/>

Rosemary Collins, E. (2023, July 4). *Transformer vs LSTM: A Helpful Illustrated Guide*. Retrieved August 5, 2024, from <https://blog.finxter.com/transformer-vs-lstm/>

Brownlee, J. (2022, September 27). *Attention in Long Short-Term Memory Recurrent Neural Networks*. Retrieved August 5, 2024, from <https://machinelearningmastery.com/attention-long-short-term-memory-recurrent-neural-networks/>

Singer, P. (2024, July 22). *Ethics philosophy*. Retrieved August 5, 2024, from <https://www.britannica.com/topic/ethics-philosophy>

Brownlee, J. (2020, September 12). *Understand the Impact of Learning Rate on Neural Network Performance*. Retrieved August 6, 2024, from <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>