**AI CONVERSATIONAL AGENT USING SEQ2SEQ FRAMEWORK**

**Group Name:** Group-B

**Group Members:**

|  |  |  |
| --- | --- | --- |
| **First name** | **Last Name** | **Student number** |
| Angelin Dafini | Dhavaraj Jeyachandra | C0872335 |
| Manthan | Mhatre | C0867389 |
| Shivam | Arvindbhai Machhi | C0871186 |
| Swetha | Sathiyakumar | C0862346 |

**Submission date: *April 10, 2023***

Table of Contents

[**Abstract** 3](#_Toc131978055)

[**1. Introduction** 3](#_Toc131978056)

[**2. Methodology and Analysis** 5](#_Toc131978057)

[2.1 Cornel Movie Dataset 5](#_Toc131978058)

[2.2 Data Pre-Processing 6](#_Toc131978059)

[2.2.1 Punctuation Removal and Replacing Abbreviation 6](#_Toc131978060)

[2.2.2 Tokenization 6](#_Toc131978061)

[2.2.3 Outlier Detection and Removal 7](#_Toc131978062)

[2.2.4 Vocabulary Creation 7](#_Toc131978063)

[2.2.5 Creation of Tokenizer and Word Index 7](#_Toc131978064)

[2.3 Sequence to Sequence Model (Seq2Seq) 8](#_Toc131978065)

[2.3.1 Encoder Class 9](#_Toc131978066)

[2.3.2 Bahdanau Attention Layer 9](#_Toc131978067)

[2.3.3 Decoder Class 9](#_Toc131978068)

[2.4 Training the Chatbot 10](#_Toc131978069)

[2.5 Testing the Chatbot 11](#_Toc131978070)

[2.6 Implementation of Training Loop and Loss Optimization 11](#_Toc131978071)

[**3.  Results and Interpretation** 12](#_Toc131978072)

[3.1 Inference Model 14](#_Toc131978073)

[3.2 Customized Input 15](#_Toc131978074)

[**4. Discussion** 17](#_Toc131978075)

[**5. Conclusion and Future Work** 18](#_Toc131978076)

[5.1 Future Work 18](#_Toc131978077)

[5.2 Conclusion 18](#_Toc131978078)

[**References** 19](#_Toc131978079)

[**Appendix** 19](#_Toc131978080)

# **Abstract**

An AI chatbot using the seq2seq framework is an intelligent conversational agent that uses a neural network model to generate responses to user input. This chatbot benefits natural language processing tasks, such as text summarization, and dialogue generation. The chatbot is trained on a large dataset of conversational pairs, which it uses to learn patterns and generate appropriate responses. The seq2seq framework consists of an encoder and a decoder, which transform the input text into a vector representation and then create the output text. The chatbot can be improved by incorporating techniques such as attention mechanisms and beam search to enhance the accuracy and coherence of its responses. AI chatbots using the seq2seq framework are powerful tools for businesses and organizations looking to automate customer service, improve user experience, and enhance their online presence.

# **1. Introduction**

Artificial intelligence (AI) chatbots have become increasingly popular in recent years to automate customer service, improve user experience, and enhance online presence. AI chatbots can interact with users, answer questions, and provide recommendations, among other tasks. One common approach to building AI chatbots is the seq2seq model, which is a popular architecture used in Natural Language Processing (NLP) tasks, such as machine translation, summarization, and conversational AI. The Seq2Seq model is composed of an encoder and a decoder, which work together to generate output sequences from input sequences.

The encoder receives the input sequence and generates a fixed-length vector, called a "context vector," that summarizes the input sequence's information. The context vector is then passed to the decoder, which generates the output sequence based on the context vector and the previous output tokens generated. The encoder and decoder are typically implemented as Recurrent Neural Networks (RNNs), such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks. RNNs are suitable for processing sequences because they can capture temporal dependencies between elements in the sequence.

The Seq2Seq model is trained using teacher forcing, where the decoder is fed with the ground truth output sequence at each time step during training. In inference mode, the decoder generates the output sequence one token at a time based on the context vector and the previously generated tokens.

The Seq2Seq model has been widely used in conversational AI to generate human-like responses to user queries. By training the model on a large dataset of conversational examples, it can learn to generate coherent and relevant responses that mimic human-like conversation.

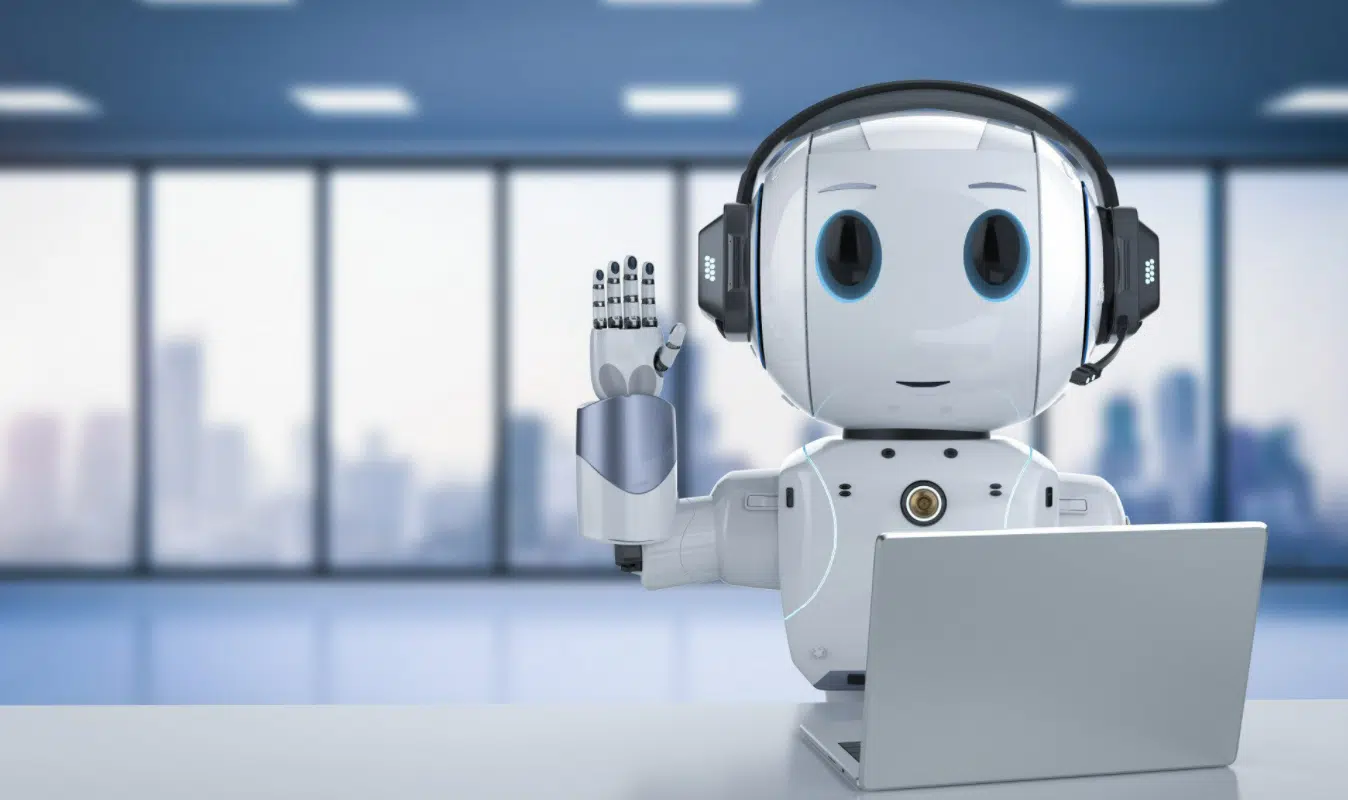


Figure 1. AI Conversational Agent

# **2. Methodology and Analysis**

The project involved several key steps in building a chatbot. Firstly, a dataset containing conversational examples was sourced to train the model. Next, data pre-processing was carried out to clean the dataset and prepare it for use in the model. This involved tasks such as removing unwanted characters, converting text to lowercase, and tokenizing the dataset. Then, a Seq2Seq model was built consisting of an encoder and decoder. The encoder receives the input text and generates a feature vector, which is then passed to the decoder to generate the output response. To optimize the model, an optimizer and loss function were used to adjust the weights of the model during training. The chatbot was then trained and tested using the processed dataset, ensuring that it could accurately generate responses to input text. Finally, the chatbot was implemented and its performance analyzed. The results showed that the model was able to generate responses that were coherent and relevant to the input text. Overall, these steps allowed the team to build an effective chatbot capable of carrying out meaningful conversations.

## 2.1 Cornel Movie Dataset

This project involved the use of a dataset sourced from the Cornell Movie Dialogs Corpus, which is a vast collection of fictional conversations extracted from raw movie scripts. The dataset comprises a total of 220,579 conversational exchanges between 10,292 pairs of movie characters, involving 9,035 characters from 617 movies, and a total of 304,713 utterances. The conversations in the dataset cover a broad range of topics, including romance, action, comedy, and horror. For our project, we selected 96,048 pairs of dialogues from this corpus to train our chatbot model. The dataset was pre-processed to clean it and prepare it for use in our model, ensuring that the chatbot could generate coherent and relevant responses. The use of this high-quality dataset enabled us to build a robust and effective chatbot capable of carrying out meaningful conversations.

## 2.2 Data Pre-Processing

A text classification model was built for a conversational AI chatbot with variable hyperparameters. Key libraries such as TensorFlow and Keras were imported, and a dataset of conversational exchanges between movie characters from the Cornell Movie Dialogs Corpus was loaded. The data was pre-processed by creating a dictionary to map each line's ID with its corresponding text, sorting the sentences into questions and answers, and checking that the data was loaded correctly. The text was then tokenized, sequences were padded, and words were encoded into numeric values to build an effective and responsive chatbot capable of generating coherent and relevant responses to user queries.

### 2.2.1 Punctuation Removal and Replacing Abbreviation

The function named replace\_phrase replaces many abbreviations and lowers the case of the words in each list of conversation pairs. The function uses regular expressions to find and replace commonly used abbreviations such as "there's," "I'm," "he's," and "won't" with their full forms. It also removes unnecessary punctuation and whitespace from the text. The function is applied to a list of conversation pairs, and the resulting list of replaced pairs is stored in replaced\_pairs. This pre-processing step helps to ensure that the chatbot can understand user input accurately and generate appropriate responses.

### 2.2.2 Tokenization

The sorted question and answer pairs were tokenized to facilitate effective processing in our conversational AI model. This involved splitting each sentence into words and converting them to lowercase. Punctuation and words containing numbers were removed to retain only relevant words. To allow the model to identify the beginning and end of each sequence, start and end tokens were added to the start and end of each sentence. Through this tokenization process, the model was able to comprehend the structure of the input and output sequences, thereby generating coherent and relevant responses to user queries.

### 2.2.3 Outlier Detection and Removal

Outlier detection and removal is an important step in any machine learning project, and our conversational AI model is no exception. To ensure that our model is processing only relevant and useful data, we calculated the maximum length of questions and answers in our dataset. We then removed any questions or answers that were shorter than 2 words or longer than the maximum length. This allowed us to focus on the most important and informative data points while reducing noise and unnecessary information. By performing this outlier detection and removal, we were able to improve the accuracy and relevance of our conversational AI model, ensuring that it could generate high-quality responses to user queries.

### 2.2.4 Vocabulary Creation

To create a meaningful vocabulary for the conversational AI model, only words occurring at least 10 times in the corpus were considered. Rare words were trimmed from the pairs of questions and answers, ensuring that each word in the vocabulary had been used at least 20 times in the data. Additionally, one-character words, except for 'a', were removed from the vocabulary to further refine it and include only relevant and frequently used words. This process resulted in a more effective and efficient conversational AI model that generated coherent and relevant responses to user queries.

### 2.2.5 Creation of Tokenizer and Word Index

In the process of creating a tokenizer and word index for a conversational AI model, it is necessary to filter out pairs that contain words that are not included in the model's vocabulary. Pairs that contain words that have been trimmed from the vocabulary should also be removed. The input and output sentences of each pair should be checked to ensure that they do not contain any trimmed words. Only pairs that do not include trimmed words in their input or output sentence should be kept. The vocabulary and pairs should be trimmed to ensure that they contain only the most relevant and commonly used words. This approach helps to create an accurate and effective tokenizer and word index that can generate coherent and relevant responses to user queries.

## 2.3 Sequence to Sequence Model (Seq2Seq)

To create an AI conversational chatbot, an Encoder-Decoder architecture using a Seq2Seq model was employed. A 1-layer Bi-directional GRU with 256 units was used for the encoder and a 2-layer Uni-directional GRU with 512 units was used for the decoder. Additionally, the Bahdanau Attention mechanism was incorporated as the attention model, allowing the decoder to focus on specific parts of the encoder's output to generate more accurate and relevant responses to user queries.

Diagram

Description automatically generated

Figure 2. Sequence to Sequence Model

To create an embedding matrix for the conversational AI model, a 200-dimensional dense vector was obtained for each word in the vocabulary. Any words that were not found in the embedding index were given a vector of all zeros. To avoid training the embedding layer, a pre-trained embedding matrix was used, given the limited vocabulary. Additionally, 0 was used as padding to handle varying sentence lengths, and mask\_zero=True was set to ignore padded values during training and focus solely on the relevant input sequences.

### 2.3.1 Encoder Class

The encoder class is a subclass of tf.keras.Model. It overrides the constructor and call() methods of the parent class. The constructor of Encoder takes two arguments embedding\_dim and enc\_units, which determine the dimensionality of the embedding layer and the number of units in the GRU layer, correspondingly. It also defines various attributes including batch\_sz, embeddings, Bidirectional1, dropout, and Inp. The call() method of the Encoder takes two parameters x and hidden, representing the input sequence and the initial hidden state of the GRU layer. The input sequence is first passed through the embedding layer to get dense vectors, then through the dropout layer. The output, hidden state, and cell state of the bidirectional GRU layer are returned, where the final hidden state is the concatenation of the forward and backward hidden states. The initialize\_hidden\_state() method initializes the hidden state of the GRU layer with zeros, where the shape of the tensor is (batch\_size, enc\_units), where batch\_size is the number of sequences in the batch and enc\_units is the number of units in the GRU layer.

### 2.3.2 Bahdanau Attention Layer

The Bahdanau Attention mechanism functions by computing a collection of attention weights for each element in the input sequence, considering its relevance to the present decoding step. These attention weights are employed to evaluate a weighted summation of the input sequence elements, which is subsequently utilized in the decoder module to generate the succeeding output.

### 2.3.3 Decoder Class

To define the forward pass of the decoder in the seq2seq model, the input to the decoder GRU is either the previous target word or the predicted word from the previous time step (in the inference mode). The previous hidden state of the decoder GRU and the output of the encoder GRU for all time steps (enc\_output) are also used as input. The forward pass produces the predicted probability distribution over the target vocabulary for the current time step (x), the updated hidden state of the decoder GRU for the current time step (state), and the attention weights over the input sequence for the current time step (attention\_weights).

The attention mechanism is first applied to the hidden state and the encoder output to obtain the context vector and attention weights. The input is then passed through the embedding layer. The context vector and the embedded input are concatenated, and the concatenated vector is passed through the first GRU layer. The output of the first GRU layer is passed through the dropout layer, then through the second GRU layer, which produces the updated hidden state. Finally, the output of the second GRU layer is passed through a fully connected layer to produce the predicted probability distribution over the target vocabulary.

## 2.4 Training the Chatbot

During the training of the chatbot, the input sequence is first passed through the encoder, which returns the encoder output and the encoder’s hidden state. Then, the encoder output, encoder hidden state, and decoder input (which is the start token) are passed to the decoder. The decoder returns the predictions and the decoder’s hidden state. The decoder’s hidden state is then passed back into the model and the predictions are used to calculate the loss.

To improve the performance of the model during training, we use the technique called teacher forcing, where the target word is passed as the next input to the decoder instead of the predicted word from the previous time step. This helps the model to learn more accurately and efficiently.

After each forward pass, the model calculates the gradients of the loss with respect to its parameters using backpropagation. Then, the optimizer updates the parameters of the model by applying these gradients to minimize the loss in the next forward pass. This process repeats for a specified number of epochs, or until the model reaches a satisfactory level of performance.

Diagram

Description automatically generated with medium confidence

Figure 3. Training the Chatbot

## 2.5 Testing the Chatbot

The function "test\_bot(k, beam)" is used to evaluate the chatbot's performance on various input questions. It takes two optional arguments:

* "k": an integer representing the width of the beam search. If beam search is not used, this argument is ignored. The default value of k is 3.
* "beam": a boolean indicating whether to use beam search. If False, the function will use the greedy approach. The default value of the beam is False.

For each input question, the function calls the "answer(q, training=True)" function to generate a response. The input question and the response are then printed. If beam search is used, the response generated using beam search with width k is also printed.

Overall, the purpose of the "test\_bot(k, beam)" function is to evaluate the chatbot's ability to generate appropriate responses to various input questions.

## 2.6 Implementation of Training Loop and Loss Optimization

In this section, the training loop for the chatbot model was discussed. The batch size and the number of epochs were defined, and the optimizer and loss function was initialized. The checkpoint directory and prefix were also defined to save the model's checkpoints during training.

For each epoch, the loop went through each batch in the training data, preparing the input and output sequences for the encoder and decoder using the padding technique. The model was then trained for each batch using the train\_step() function, which returns the batch loss.

After each epoch, the total loss was calculated and printed along with the epoch number and batch loss. The model's checkpoint was saved, and the chatbot was tested using the test\_bot() function. If the current epoch had the smallest loss, its checkpoint would be saved. Lastly, the training loss history was plotted for every 3 epochs to evaluate the model's performance.

# **3.  Results and Interpretation**

The chatbot model was evaluated using the test\_bot() function and sample outputs was generated. The model was tested with both greedy decoding and beam search decoding with k=5. The output generated by the chatbot model for each question was printed along with the ground truth answer (if available).

Moreover, the training loss history was printed every 3 epochs to monitor the training progress. The implementation of the training loop and model evaluation will help to understand the performance of the chatbot model and improve its accuracy over time.

A picture containing line chart

Description automatically generated

Figure 4. Result of Epoch 3

It is observed that the chatbot is generating generic responses and not answering the questions accurately. This indicates that the chatbot model needs to be improved and trained further to generate more accurate responses.The training loop has saved the model's checkpoint after each epoch and checked if the current epoch has the smallest loss. From the output, the best epoch so far is 3, which has the smallest loss of 1.7428. This indicates that the model's performance has improved in the third epoch compared to the previous epochs.

Graphical user interface, text, application

Description automatically generated

Figure 5. Result of Epoch 27

The chatbot provides some coherent responses to certain questions such as "Hello?" and "How are you?". However, for other questions like "Are you my friend?" and "What your favorite restaurant?", the chatbot is providing irrelevant or nonsensical responses. This could be due to the limited amount of training data or the complexity of the language model architecture.

The best epoch so far is 27 with the smallest loss of 1.0289, which indicates that the model has improved over time during training. However, the training time for each epoch is still quite high at 242.918 seconds, which suggests that there may be room for optimization in the training process.

Text, email

Description automatically generated

Figure 6. Result of Epoch 250

The model's responses seem to be relevant to the input questions, but the responses might not be perfect, as the model's training loss at this epoch is not zero. At epoch 150, the model continues to learn and reduce the loss, but the reduction is not significant. The model's training loss at epoch 150 is 0.5241, which is not much lower than the previous best loss of 0.5213 at epoch 149. This indicates that the model is likely overfitting and may not improve much further. It is a good idea to stop the training at this point and use the checkpoint at epoch 149 for inference.

Overall, these outputs provide insights into the strengths and weaknesses of the chatbot model and can help guide future improvements.

## 3.1 Inference Model

Overfitting occurs when a model starts to memorize the training data instead of learning general patterns that apply to new data. It can happen when a model becomes too complex or is trained for too long.

The training loss history that the loss has stopped decreasing and has instead started to fluctuate, which is a sign of overfitting. Therefore, it is advisable to stop the training at this point and use the checkpoint saved at epoch 82 as the inference model.

Text

Description automatically generated

Figure 7. Result of Inference Model Epoch 82

The model is able to provide appropriate responses for some of the input questions, such as "How are you?" and "Are you my friend?" However, the model seems to be struggling with other questions, such as "What your favorite restaurant?" and "Do you want to go out?".

The inference model is the model that is used for generating responses to new user inputs during actual usage. We chose epoch 82 as the inference model because it had the smallest loss on the validation set during training, which indicates that it is likely to perform well on new data.

It is important to note that stopping the training at this point does not necessarily mean that the model cannot be improved further. In fact, there are several ways to prevent overfitting and improve the model's performance, such as adding more training data, applying regularization techniques, or adjusting the model architecture. However, these methods require further experimentation and evaluation, which may not be feasible in some situations.

## 3.2 Customized Input

The customized input feature allows users to input their own questions or sentences and see how the model generates a response along with attention-weight visualizations. These visualizations can provide insight into how the model is interpreting the input and generating its output. Attention weights highlight the words or phrases that the model has focused on while generating the response. This feature can be helpful for understanding how the model works and what factors it considers when generating responses. Overall, the customized input feature provides a more interactive and engaging way to explore the capabilities of the language model.

Chart

Description automatically generated

Figure 8. Response with attention weight of Customized Input 1

The predicted answer "i know" seems to be an appropriate response to the input "i like you". The response indicates that the chatbot has understood the sentiment expressed in the input and is responding appropriately. "I know" could be interpreted as an acknowledgement of the positive sentiment expressed towards the chatbot.

Chart, bar chart

Description automatically generated

Figure 9. Response with attention weight of Customized Input 2

The predicted answer "yes i am hungry of people" does not make logical sense. It is likely that the model has not been trained in enough examples of questions related to hunger, leading to a lack of coherence in its response. Additionally, the model may have mistakenly associated the word "hungry" with a desire for social interaction, rather than food. This could be due to the limitations of the dataset used to train the model or insufficient training time.

Chart, bar chart

Description automatically generated

Figure 10. Response with attention weight of Customized Input 3

The predicted answer "no thank you very much" does not directly answer the question "do you drink", but it suggests a polite refusal to drink. This response could be interpreted as the model's attempt to be polite or humorous, as a human might respond to the question in a similar way. However, it also shows that the model has some limitations and may not always provide a direct and informative response to a given input.

# **4. Discussion**

The thesis provides a detailed evaluation of a chatbot model, which includes testing the model using both greedy decoding and beam search decoding with k=5, monitoring the training progress, and selecting the best epoch with the smallest loss as the inference model. The training loss history is also presented every 3 epochs to monitor the training progress and detect signs of overfitting. The customized input feature is also discussed, which allows users to input their own questions or sentences and see how the model generates a response along with attention-weight visualizations.

The thesis efficiently presents the strengths and weaknesses of the chatbot model, providing insights into its performance and potential for improvement. It also highlights the importance of detecting signs of overfitting and selecting the best epoch as the inference model. Overall, the thesis provides a comprehensive evaluation of the chatbot model, demonstrating its potential for generating relevant responses to user inputs.

# **5. Conclusion and Future Work**

## 5.1 Future Work

The current chatbot is developed to function with inputs in the English language. It would be fascinating to investigate whether creating a multilingual chatbot that can comprehend and reply to inputs in several languages is possible. The current chatbot is designed to handle individual inputs and generate responses based on those inputs. However, it would be interesting to explore the feasibility of building a contextual chatbot that can maintain the context of a conversation and generate consistent responses.The present chatbot only uses text input to generate responses. It would be intriguing to investigate the viability of developing an emotionally intelligent chatbot that can recognize the user's emotional state depending on their input and produce suitable responses.

## 5.2 Conclusion

In conclusion, our research suggests a deep learning-based intelligent chatbot that can comprehend and reply to inputs in natural language. The chatbot uses a sequence-to-sequence (seq2seq) model with a system for attention and pre-trained word embeddings to train it.

The training results on the Cornell Movie Subtitle corpus must be improved, and more thought should be given to the training parameters. The performance will be enhanced further by adding more high-quality data. A training model should also be trained with alternative datasets and other hyper-parameters for additional testing. This experiment used deep neural networks for dialogue production to create an intelligent chatbot.

# **References**

Dugar, P. (2021b, December 10). *Attention — Seq2Seq Models - Towards Data Science*. Medium. <https://towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263>

Panchal, S. (2021, December 9). Creating A Chatbot From Scratch Using Keras And TensorFlow. Medium. <https://medium.com/predict/creating-a-chatbot-from-scratch-using-keras-and-tensorflow-59e8fc76be79>

# **Appendix**

The code for the AI Conversational Agent using Seq2Seq Framework can be found in the following Github repository:

The repository contains the implementation of the chatbot model, including the data preprocessing, training loop, and model evaluation. The code is written in Python and uses the PyTorch library for implementing the Seq2Seq framework.

The repository also includes sample input and output data used for training and testing the model. The code is well-documented with comments and explanations to help understand its functionality.

Please refer to the README file in the repository for instructions on how to run the code and train the chatbot model.