Contents

[LSTM Chatbot Project Documentation 1](#_Toc198279192)

[1. Introduction 1](#_Toc198279193)

[1.1. Project Objectives 1](#_Toc198279194)

[1.2. Team Members 1](#_Toc198279195)

[2. Project Architecture 1](#_Toc198279196)

[2.1. Components 2](#_Toc198279197)

[3. Implementation Details 2](#_Toc198279198)

[3.1. Data Preprocessing 2](#_Toc198279199)

[3.2. LSTM Model Architecture 2](#_Toc198279200)

[3.3. Model Training 3](#_Toc198279201)

[3.4. Chatbot Application (Streamlit) 3](#_Toc198279202)

[4. How to Run the Project 4](#_Toc198279203)

[4.1. Prerequisites 4](#_Toc198279204)

[4.2. Setup and Installation 5](#_Toc198279205)

[4.3. Running the Chatbot Application 5](#_Toc198279206)

[5. Project Files Overview 5](#_Toc198279207)

[6. Potential Improvements and Future Work 6](#_Toc198279208)

[7. Conclusion 6](#_Toc198279209)

# LSTM Chatbot Project Documentation

## 1. Introduction

This document provides a detailed overview of the LSTM (Long Short-Term Memory) Chatbot project. The primary goal of this project is to develop an intelligent conversational agent capable of understanding user queries and providing relevant responses based on a predefined dataset. The chatbot utilizes an LSTM neural network, a type of recurrent neural network (RNN) well-suited for sequence modeling tasks like natural language processing.

### 1.1. Project Objectives

* To build a functional chatbot using LSTM architecture.
* To preprocess and prepare textual data for training the LSTM model.
* To train the LSTM model on a custom dataset to learn patterns and relationships between user inputs and corresponding answers.
* To create an interactive application (using Streamlit) that allows users to interact with the trained chatbot.
* To provide a clear and understandable documentation of the project, including its architecture, implementation details, and usage instructions.

### 1.2. Team Members

* Abdallah Samir
* Youssef Samy
* Shaaban Mosaad
* Nada Amr
* Mostafa Ahmed Elesely
* Mohammed Ahmed Mohammed Badrawy

## 2. Project Architecture

The LSTM chatbot project follows a typical machine learning pipeline:

1. **Data Collection and Preprocessing**: Gathering conversational data and cleaning it for model consumption.
2. **Model Building**: Designing and implementing the LSTM neural network.
3. **Model Training**: Training the LSTM model on the preprocessed data.
4. **Model Evaluation**: Assessing the performance of the trained model (implicitly done through observing chat interactions).
5. **Deployment**: Creating a user interface for interacting with the chatbot.

### 2.1. Components

* **Dataset**: A collection of intents, patterns (user questions), and responses. The data\_merged.csv file likely contains this data.
* **Tokenizer**: Converts text data into sequences of integers (tokenizer.pickle).
* **Label Encoder**: Converts categorical intent labels into numerical format (label\_encoder.pickle).
* **LSTM Model**: The core neural network model responsible for understanding input and predicting responses (simple\_chatbot\_train\_model.h5).
* **Streamlit Application**: The user interface for the chatbot (lstm\_app\_v2.py).

## 3. Implementation Details

This section delves into the specifics of how the project was implemented, referencing key files from the GitHub repository (https://github.com/YosefSamy019/DEPI\_graduation\_project/tree/main/simple\_chatbot\_code).

### 3.1. Data Preprocessing

Found primarily in the chatbot.ipynb notebook, the data preprocessing steps include:

* **Loading Data**: Reading the conversational data from data\_merged.csv.
* **Cleaning Text**: Removing punctuation, converting to lowercase, and potentially lemmatization or stemming (using libraries like NLTK or spaCy, though spaCy (en\_core\_web\_sm) is explicitly loaded in lstm\_app\_v2.py).
* **Tokenization**: Breaking down sentences into individual words (tokens).
* **Padding Sequences**: Ensuring all input sequences have the same length by padding shorter sequences and truncating longer ones. The MAX\_LEN.pickle file likely stores the maximum sequence length used.
* **Encoding Labels**: Converting intent labels into a numerical format suitable for the model.

**Key Files:** \* chatbot.ipynb: Contains the data loading, preprocessing, model building, and training code. \* data\_merged.csv: The dataset file. \* tokenizer.pickle: Stores the trained Keras Tokenizer. \* label\_encoder.pickle: Stores the trained scikit-learn LabelEncoder. \* MAX\_LEN.pickle: Stores the maximum sequence length for padding. \* tags\_answers.pickle: Likely stores a mapping between intent tags and their possible answers.

### 3.2. LSTM Model Architecture

The LSTM model architecture, defined in chatbot.ipynb, typically consists of:

* **Embedding Layer**: Converts integer-encoded words into dense vector representations. The input dimension would be the vocabulary size, and the output dimension is the embedding size.
* **LSTM Layer(s)**: One or more LSTM layers to process the sequence of word embeddings and capture contextual information.
* **Dense Layer(s)**: Fully connected layers, often with activation functions like ReLU, to further process the LSTM output.
* **Output Layer**: A Dense layer with a softmax activation function to output a probability distribution over the possible intents. The number of units in this layer corresponds to the number of unique intents.

**Example (Conceptual Code Snippet from chatbot.ipynb):**

# Conceptual model structure (actual code in the notebook)  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Embedding, LSTM, Dense  
  
model = Sequential()  
model.add(Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=max\_sequence\_len))  
model.add(LSTM(units=128, return\_sequences=True)) # Example units  
model.add(LSTM(units=128))  
model.add(Dense(units=num\_classes, activation="softmax"))  
  
model.compile(optimizer="adam", loss="categorical\_crossentropy", metrics=["accuracy"])

### 3.3. Model Training

The model is trained using the preprocessed data in chatbot.ipynb.

* **Splitting Data**: The data might be split into training and validation sets (though simple\_chatbot\_val\_model.h5 suggests a validation model was saved, its direct use in training isn’t immediately obvious from file names alone).
* **Training Process**: The model.fit() method is used to train the model for a certain number of epochs with a specified batch size.
* **Saving the Model**: The trained model weights are saved to simple\_chatbot\_train\_model.h5.

### 3.4. Chatbot Application (Streamlit)

The lstm\_app\_v2.py file contains the code for the Streamlit web application.

**Key Functionalities:**

* **Loading Resources**: Loads the trained model (simple\_chatbot\_train\_model.h5), tokenizer (tokenizer.pickle), label encoder (label\_encoder.pickle), max sequence length (MAX\_LEN.pickle), and intent-answer mappings (tags\_answers.pickle). It also loads the spaCy English model (en\_core\_web\_sm) for text processing.
* **User Input**: Provides a text input field for the user to type their message.
* **Input Processing**: The user’s input text is preprocessed in the same way as the training data (cleaning, tokenization, padding).
* **Prediction**: The preprocessed input is fed to the trained LSTM model to predict the intent.
* **Response Generation**: Based on the predicted intent, a suitable response is selected from the tags\_answers.pickle mapping.
* **Displaying Conversation**: The chat history (user messages and bot responses) is displayed in the Streamlit interface.

**Conceptual Code Snippet from lstm\_app\_v2.py for prediction:**

# Conceptual prediction flow (actual code in the file)  
import numpy as np  
from tensorflow.keras.preprocessing.sequence import pad\_sequences  
  
# ... (load model, tokenizer, label\_encoder, max\_len, tags\_answers)  
  
def get\_response(user\_input\_text):  
 # Preprocess user\_input\_text (clean, tokenize)  
 # Convert to sequence using tokenizer  
 sequence = tokenizer.texts\_to\_sequences([processed\_input\_text])  
 # Pad sequence  
 padded\_sequence = pad\_sequences(sequence, maxlen=max\_len, padding="post")  
   
 # Get prediction from model  
 prediction = model.predict(padded\_sequence)  
 predicted\_class\_index = np.argmax(prediction)  
   
 # Decode intent tag  
 predicted\_tag = label\_encoder.inverse\_transform([predicted\_class\_index])[0]  
   
 # Select a random response for the predicted tag  
 response = np.random.choice(tags\_answers[predicted\_tag])  
 return response

## 4. How to Run the Project

### 4.1. Prerequisites

* Python 3.x
* pip (Python package installer)
* The necessary Python libraries as listed in simple\_chatbot\_code/requirements.txt. Key libraries include:
  + streamlit
  + tensorflow
  + scikit-learn
  + pandas
  + numpy
  + spacy (and the en\_core\_web\_sm model)
  + nltk (potentially, based on common NLP practices, though not explicitly in lstm\_app\_v2.py imports beyond spaCy for cleaning)

### 4.2. Setup and Installation

1. **Clone the Repository**: bash git clone https://github.com/YosefSamy019/DEPI\_graduation\_project.git cd DEPI\_graduation\_project/simple\_chatbot\_code
2. **Install Dependencies**: It is recommended to create a virtual environment first. bash python -m venv venv source venv/bin/activate # On Windows: venv\Scripts\activate pip install -r requirements.txt You might also need to download the spaCy English model if not already present (the application attempts to download it if missing): python # If spacy.cli.download fails within the app, run this manually # import spacy.cli # spacy.cli.download("en\_core\_web\_sm")

### 4.3. Running the Chatbot Application

Once the setup is complete, you can run the Streamlit application:

streamlit run lstm\_app\_v2.py

This will typically open the chatbot interface in your default web browser.

## 5. Project Files Overview

Located in DEPI\_graduation\_project/simple\_chatbot\_code/:

* chatbot.ipynb: Jupyter Notebook containing the code for data preprocessing, LSTM model creation, training, and saving.
* lstm\_app\_v2.py: Python script for the Streamlit web application that serves the chatbot.
* requirements.txt: Lists the Python dependencies required for the project.
* data\_merged.csv: The dataset containing intents, patterns, and responses.
* simple\_chatbot\_train\_model.h5: The saved trained Keras LSTM model.
* simple\_chatbot\_val\_model.h5: Potentially a model saved after validation (its exact role needs clarification from the notebook).
* tokenizer.pickle: Saved Keras Tokenizer object.
* label\_encoder.pickle: Saved scikit-learn LabelEncoder object.
* MAX\_LEN.pickle: Saved maximum sequence length value.
* tags\_answers.pickle: Saved dictionary mapping intent tags to lists of possible answers.
* dataset/ (folder): Likely contains the original or intermediate dataset files before merging into data\_merged.csv.

## 6. Potential Improvements and Future Work

* **Larger and More Diverse Dataset**: Training on a more extensive and varied dataset can significantly improve the chatbot’s understanding and response quality.
* **Advanced NLP Techniques**: Incorporating techniques like attention mechanisms, transformer models (though this project specifically uses LSTM), or more sophisticated text cleaning and feature engineering.
* **Context Management**: Enhancing the chatbot to maintain context over multiple turns of a conversation.
* **Error Handling and Fallback Responses**: Implementing more robust error handling for unrecognized inputs and providing graceful fallback responses.
* **Integration with External APIs**: Connecting the chatbot to external services or APIs to provide real-time information or perform actions.
* **More Rigorous Evaluation**: Implementing quantitative evaluation metrics (e.g., F1-score, perplexity) on a held-out test set.

## 7. Conclusion

The LSTM Chatbot project successfully demonstrates the development of a conversational AI using Long Short-Term Memory networks. It covers the essential steps from data preparation to model training and deployment via a Streamlit application, providing a foundational example of building an intelligent chatbot.