

**AI Chatbot: A Neural Network-Based Conversational Agent**

Prepared by

**Nour Yaakoub**

**n.yaakoub@pristini-international.tn**

**July 1st - July 31st, 2024**

Contents

[*I.* *Executive Summary* 3](#_Toc174952774)

[*II.* *Introduction* 3](#_Toc174952775)

[*III.* *Company Profile* 3](#_Toc174952776)

[Company Overview 3](#_Toc174952777)

[Department Details 3](#_Toc174952778)

[*IV.* *Internship Role and Responsibilities* 3](#_Toc174952779)

[Job Description 3](#_Toc174952780)

[Daily Tasks 4](#_Toc174952781)

[Projects 4](#_Toc174952782)

[*V.* *Skills and Knowledge Gained* 4](#_Toc174952783)

[Technical Skills 4](#_Toc174952784)

[Soft Skills 4](#_Toc174952785)

[Tools and Technologies 4](#_Toc174952786)

[*VI.* *Challenges and How You Overcame Them* 5](#_Toc174952787)

[Challenges 5](#_Toc174952788)

[Solutions 5](#_Toc174952789)

[*VII.* *Explanation* 5](#_Toc174952790)

[Neural Network Model: 5](#_Toc174952791)

[Chat Logic and Model Integration: 7](#_Toc174952792)

[Training the Model: 8](#_Toc174952793)

[Backend Code (Flask Server): 11](#_Toc174952794)

[Frontend Code (JavaScript): 13](#_Toc174952795)

[*VIII.* *Output* 16](#_Toc174952796)

[*IX.* *Conclusion* 17](#_Toc174952797)

# Executive Summary

During my one-month internship at Belmont Web, I worked on developing a neural network-based AI chatbot using PyTorch. This report outlines the tasks performed, skills gained, challenges faced, and the contributions made during this period.

# Introduction

In today's digital age, the demand for intelligent and interactive systems has significantly increased, leading to the rise of chatbots. These systems are designed to simulate human conversation and provide users with instant responses to their queries. The project, titled "AI Chatbot: A Neural Network-Based Conversational Agent," aims to develop a chatbot capable of understanding and responding to natural language inputs from users.

# Company Profile

## Company Overview

Belmont Web, established in 2018, is a prominent digital agency based in Tunisia, specializing in creating high-quality, custom websites. The company offers a wide range of services, including web design, e-commerce development, SEO, and digital marketing. Belmont Web is committed to delivering solutions that meet and exceed clients' expectations, fostering growth and digital transformation.

## Department Details

I worked in the IT department, which is responsible for developing and maintaining web applications, implementing digital marketing strategies, and ensuring the technical excellence of all projects.

# Internship Role and Responsibilities

## Job Description

My role involved developing an AI chatbot using PyTorch, integrating it with a web-based frontend interface, and ensuring smooth interaction between the backend server and the neural network model.

## Daily Tasks

-Developing and testing the neural network model using PyTorch.

-Building a web-based chat interface using HTML, CSS, and JavaScript.

-Integrating the frontend with the backend server using Flask.

-Testing and debugging the chatbot for various user queries.

## Projects

The primary project was the development of the AI chatbot, which involved:

Frontend Interface: Creating a web-based chat interface for user interactions.

Backend Server: Setting up a Flask server to handle message processing and communication with the neural network.

Neural Network Model: Training a PyTorch-based model to classify user intents and generate responses.

# Skills and Knowledge Gained

## Technical Skills

-Proficiency in PyTorch for neural network development.

-Experience with web development using HTML, CSS, and JavaScript.

-Knowledge of backend development with Flask.

-Understanding of natural language processing techniques.

## Soft Skills

-Enhanced problem-solving abilities.

-Improved communication and teamwork skills.

-Experience in project management and time management.

## Tools and Technologies

-PyTorch for deep learning.

-Flask for backend development.

-HTML, CSS, and JavaScript for frontend development.

# Challenges and How You Overcame Them

## Challenges

-Handling complex or ambiguous user queries.

-Ensuring a seamless integration between the frontend and backend.

-Managing the resource-intensive training process.

## Solutions

-Implementing additional preprocessing steps to handle complex queries.

-Thorough testing and debugging of the integration process.

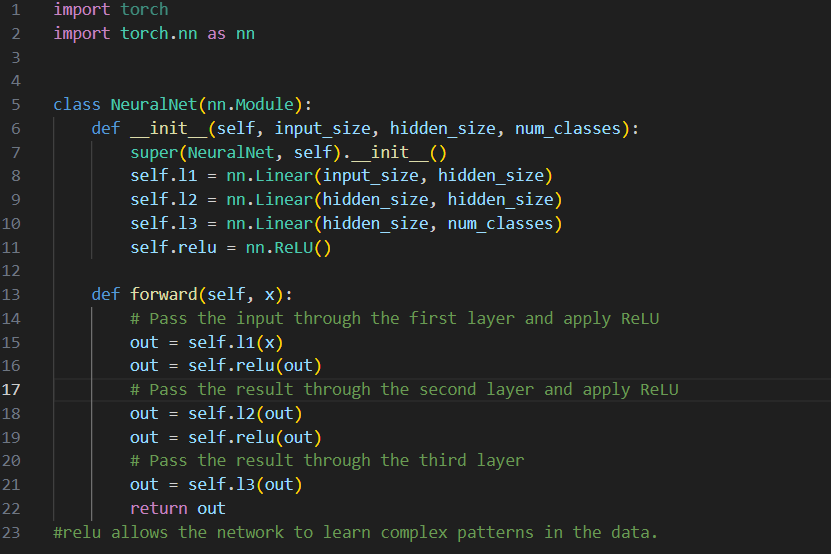
-Optimizing the training process and utilizing efficient resources.

# Explanation

Neural Network Model:

* The neural network model is responsible for processing input messages and classifying them into intents.

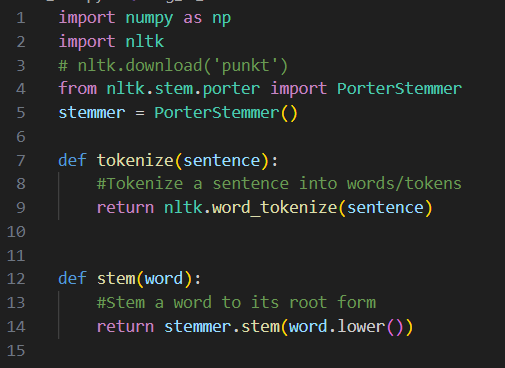
‘model.py’ - Part 1: Model Definition



Neural Network Architecture

* The ‘NeuralNet’ class extends ‘nn.Module’, representing a fully connected feedforward neural network with three layers.
* Input Layer (‘l1’): Takes input features of size ‘input\_size’ and maps them to ‘hidden\_size’ neurons.
* Hidden Layer (‘l2’): Maps ‘hidden\_size’ neurons to another ‘hidden\_size’ neurons.
* Output Layer (‘l3’): Maps ‘hidden\_size’ neurons to ‘num\_classes’ neurons, each representing an intent class.
* ReLU Activation: Applies the ReLU activation function after each linear layer to introduce non-linearity.

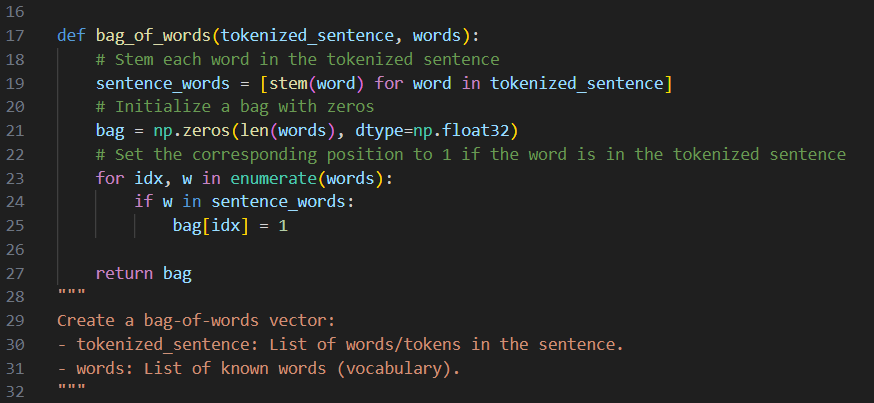
‘nltk\_utils.py’ - Part 1: Tokenization and Stemming



Text Preprocessing

* Tokenization (‘tokenize’): Splits a sentence into individual words (tokens) using NLTK's ‘word\_tokenize’.
* Stemming (‘stem’): Reduces words to their root form using PorterStemmer, which helps in standardizing words with similar meanings but different forms.

‘nltk\_utils.py’ - Part 2: Bag of Words

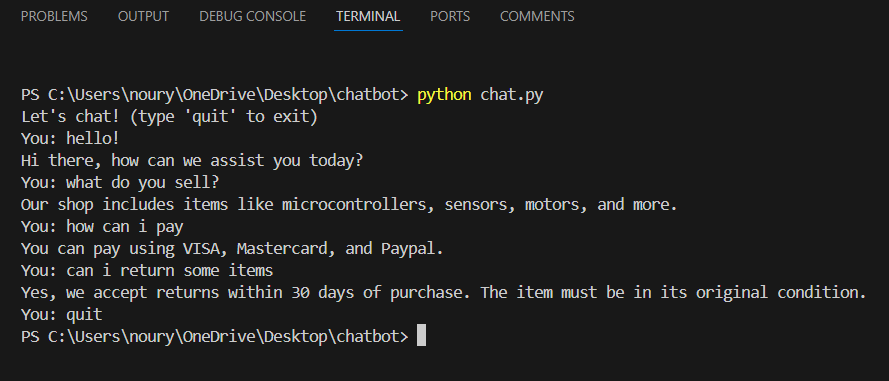


Bag of Words (‘bag\_of\_words’)

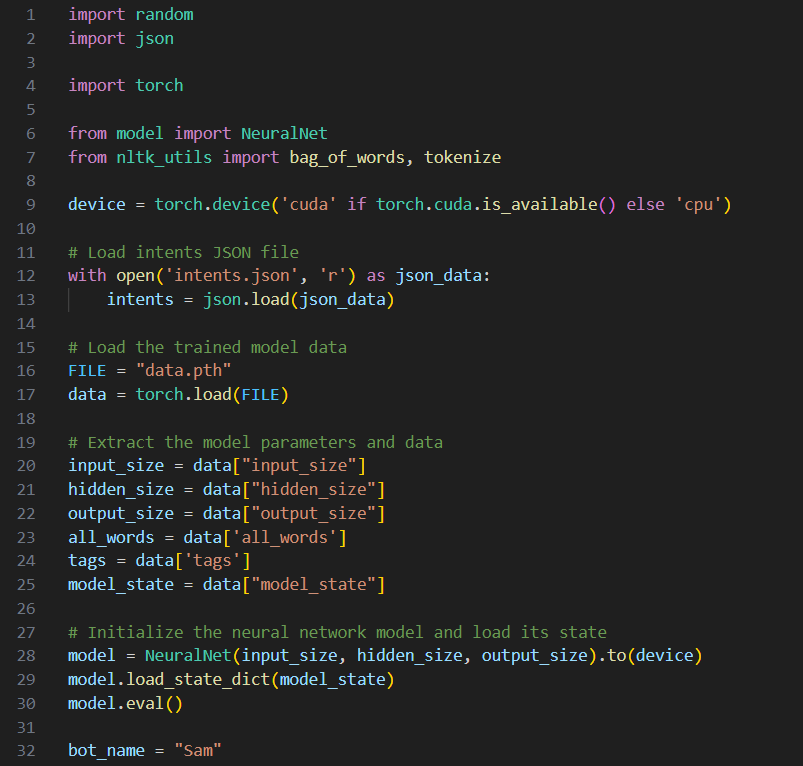
* Converts a tokenized sentence into a bag-of-words representation, where each word is mapped to a binary vector based on its presence in a predefined list (‘words’). This vector is used as input to the neural network.

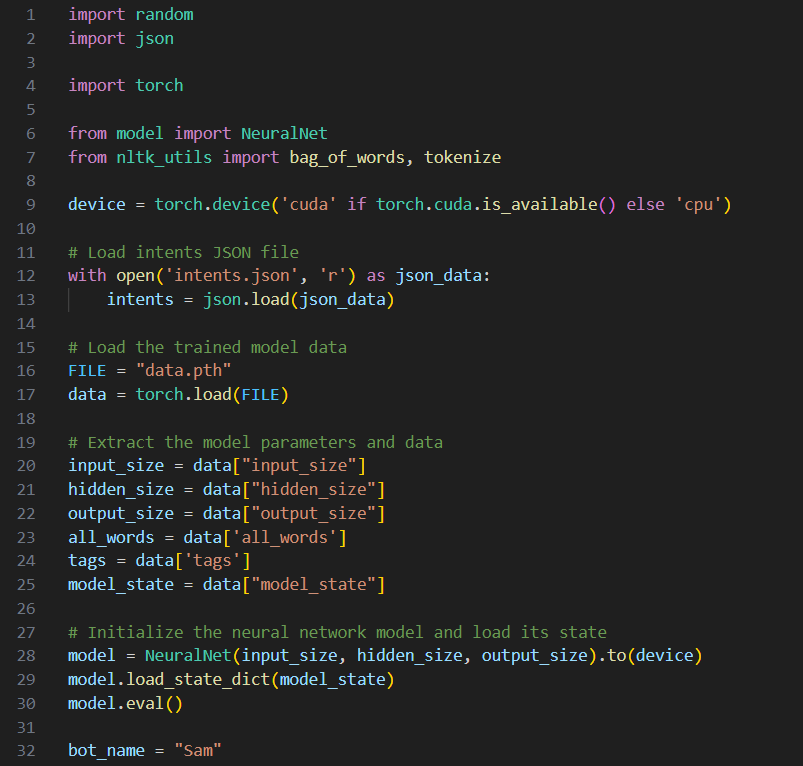
Chat Logic and Model Integration:

This part of the project integrates the trained model with the chatbot logic. (after running the chat.py in the terminal, the user can interact with the chatbot from the terminal)



‘chat.py’ - Part 1: Model Loading and Initialization

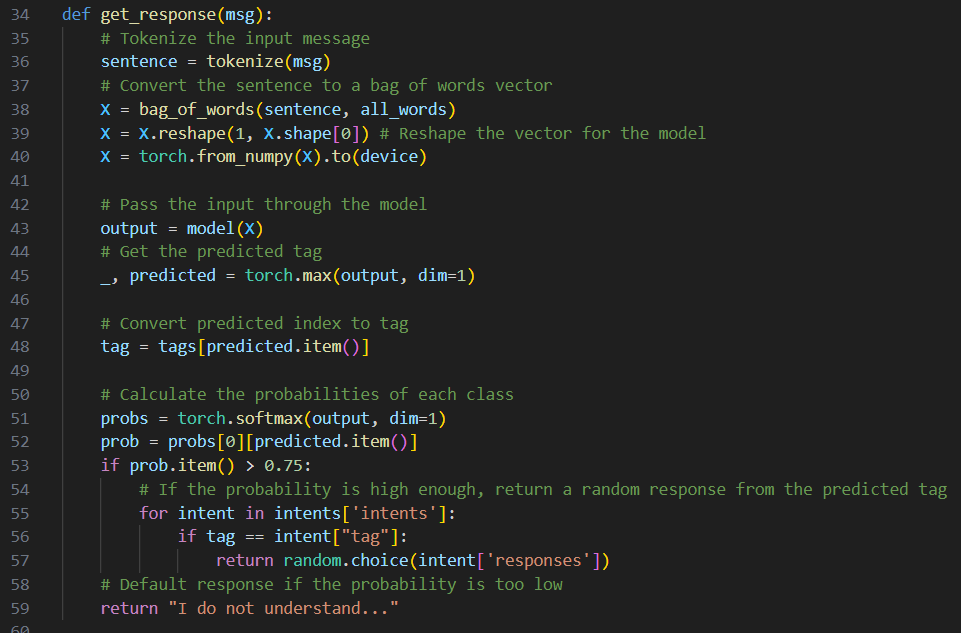




Model Initialization

* Loads the saved model parameters and training data (such as input size, hidden size, output size, words, and tags) from ‘data.pth’.
* Initializes the ‘NeuralNet’ model using the loaded parameters.
* Loads the model state dictionary (saved weights) and sets the model to evaluation mode (‘model.eval()’).

‘chat.py’ - Part 2: Generating Responses

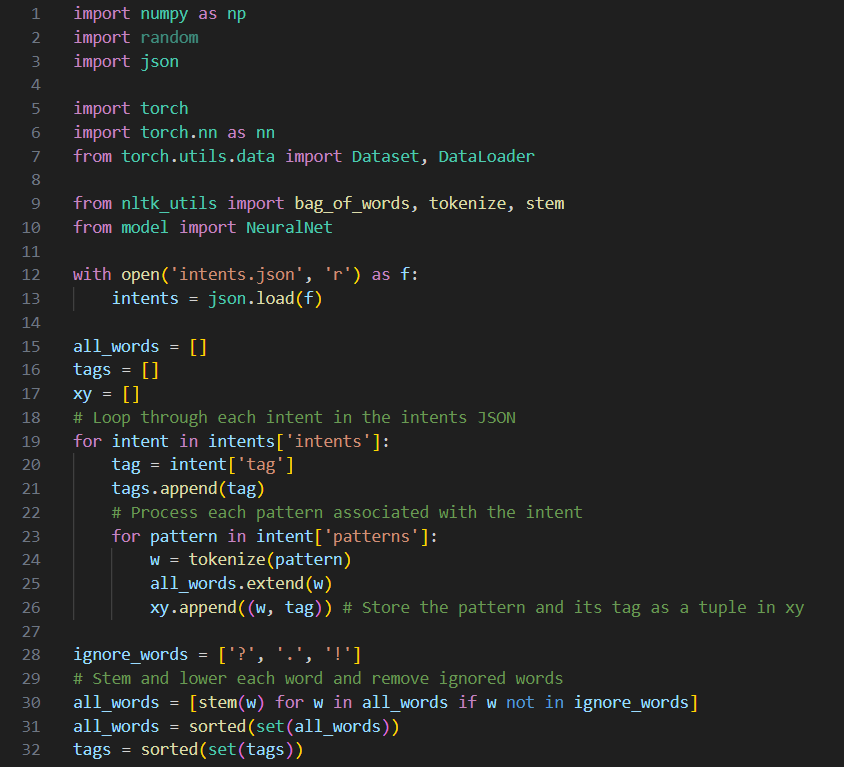


* Response Generation (‘get\_response’)
* Tokenization and Bag of Words: Converts the user message into a bag-of-words vector.
* Model Inference: Passes the vector through the neural network to predict the intent class.
* Response Selection: If the confidence score (probability) is above 75%, selects a random response from the corresponding intent; otherwise, returns a default message.

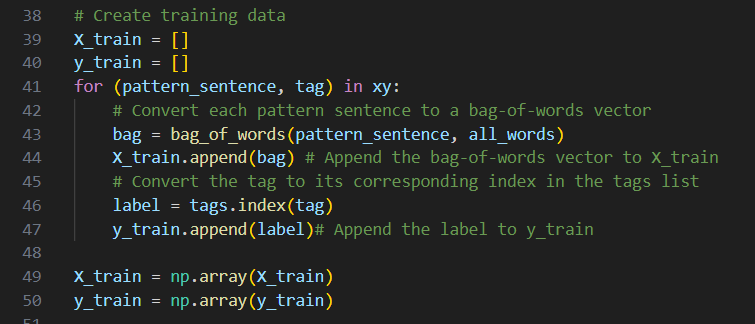
Training the Model:

The training script handles data preprocessing, model training, and saving the trained model.

‘train.py’ - Part 1: Data Preparation



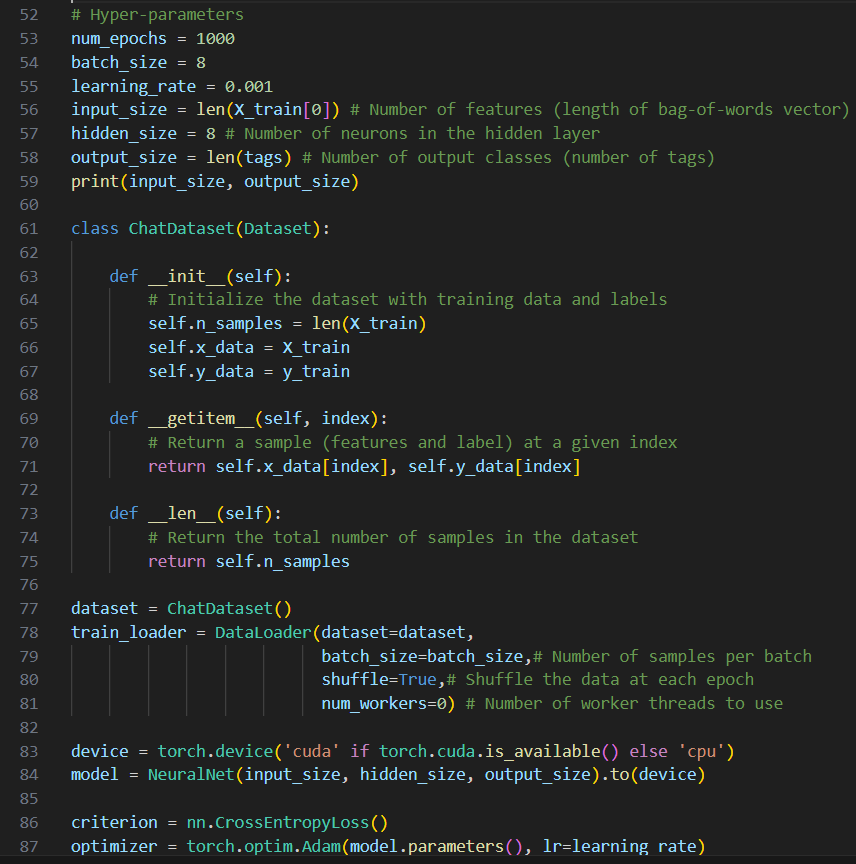
Data Preparation

* Loads the intents from ‘intents.json’.
* Extracts patterns and their corresponding tags, tokenizes, and stems the words.
* Creates a dataset of word patterns (‘all\_words’) and corresponding tags.
* ‘train.py’ - Part 2: Feature Extraction and Labeling

Feature Extraction

* Converts each tokenized pattern sentence into a bag-of-words vector.
* Maps each tag to its corresponding label index.
* Stores the features (‘X\_train’) and labels (‘Y\_train’) as numpy arrays for efficient processing.

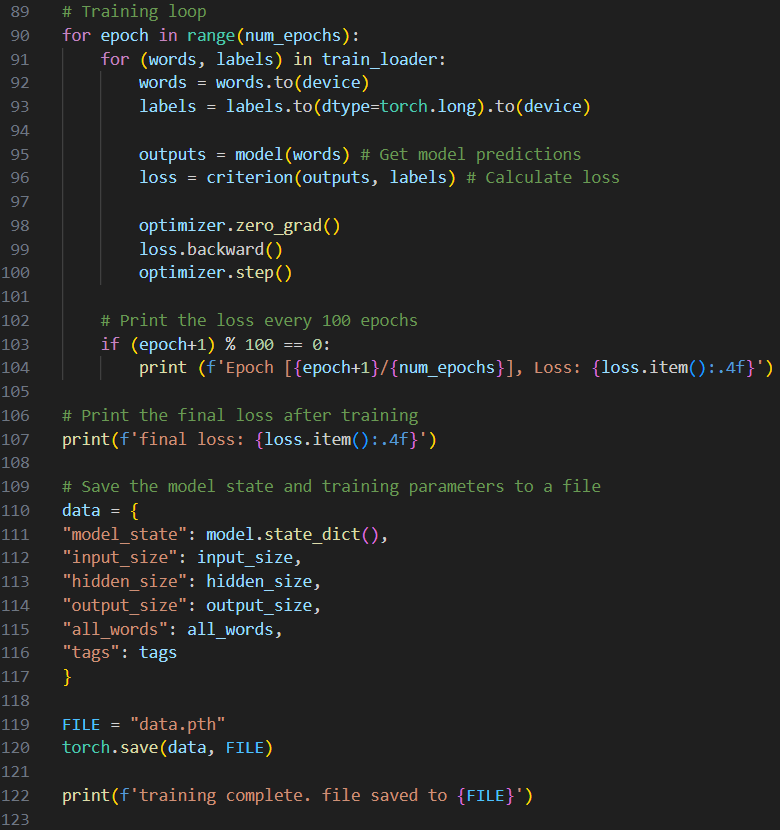
‘train.py’ - Part 3: Model Training

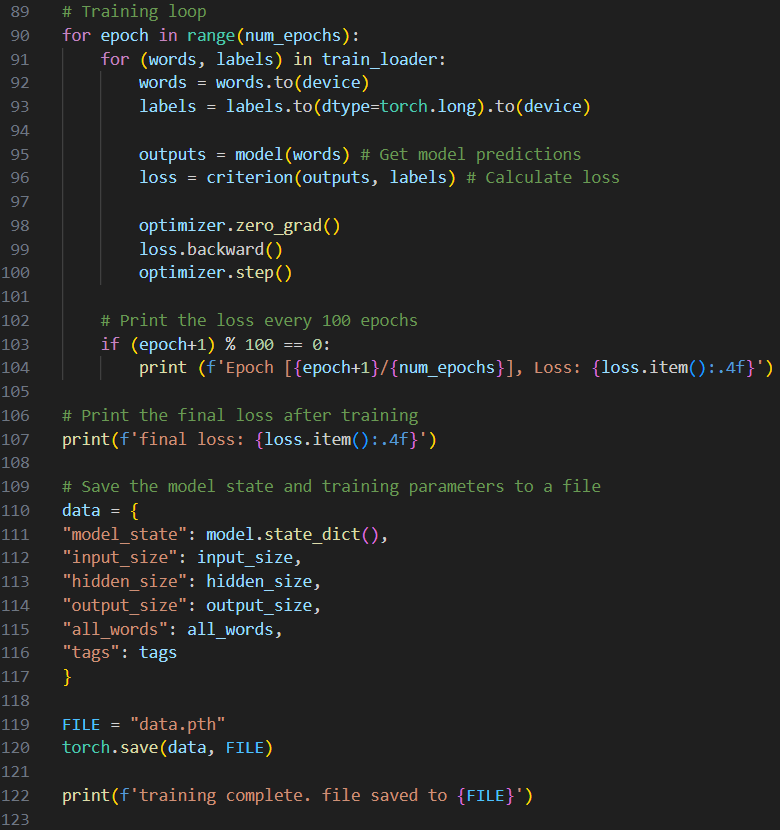


Hyperparameters and Model Setup

* Defines important hyperparameters like the number of epochs, batch size, and learning rate.
* Initializes the model, loss function (‘CrossEntropyLoss’), and optimizer (‘Adam’).
* Prepares the data for training using a custom ‘ChatDataset’ and a ‘DataLoader’ for efficient batching and shuffling.

‘train.py’ - Part 4: Training Loop and Model Saving





Training Loop

* Iterates over the dataset for a predefined number of epochs.
* Performs forward and backward passes to calculate gradients and update model weights using the optimizer.
* Prints the loss every 100 epochs to monitor training progress.
* Model Saving:
* Saves the trained model state along with important parameters (input size, hidden size, output size, words, and tags) to ‘data.pth’.

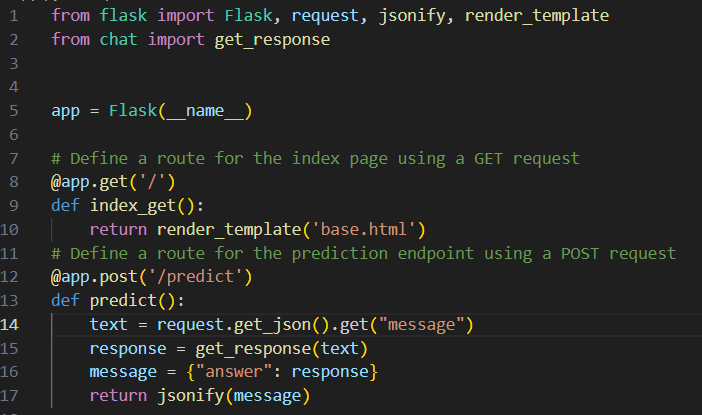
‘train.py’ – Output



Backend Code (Flask Server):

The Flask backend receives requests from the frontend, processes them using the chatbot model, and returns responses.

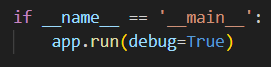
‘app.py’ - Part 1: Flask Initialization and Routes



Flask Setup (app.py)

* Initializes a Flask web application with the ‘Flask’ class.
* Sets up a ‘GET’ route for the homepage (‘index\_get’) that renders an HTML template (‘base.html’).
* Defines a POST route (‘predict’) to handle incoming messages from the frontend. The route extracts the message, processes it using the ‘get\_response’ function (from ‘chat.py’), and returns the response as JSON.

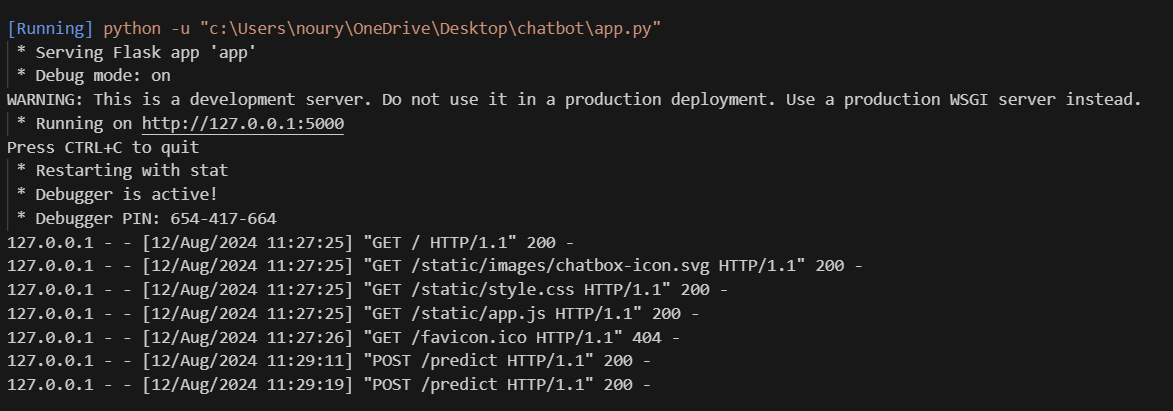
‘app.py’ - Part 2: Running the Server



Server Execution:

* Runs the Flask application in debug mode, which provides detailed error messages and auto-reloads on code changes. The server starts listening for incoming HTTP requests on the default port (5000).

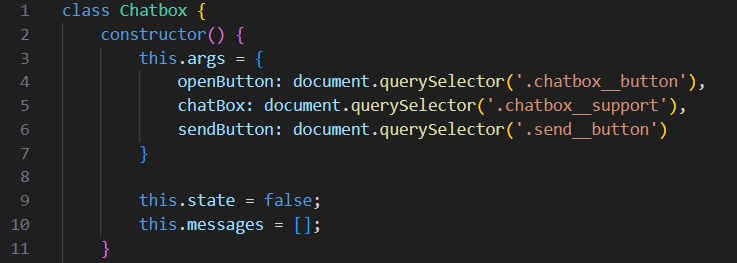
‘app.py’ - Output

******

Frontend Code (JavaScript):

This code manages the user interface and interactions with the chatbot.

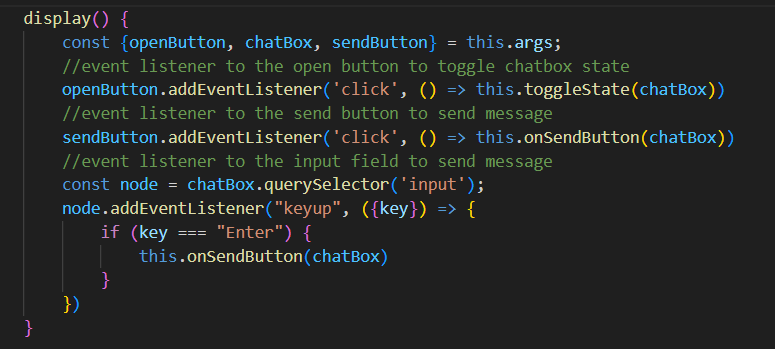
‘app.js’ - Part 1: Initialization



Class Initialization (‘Chatbox’)

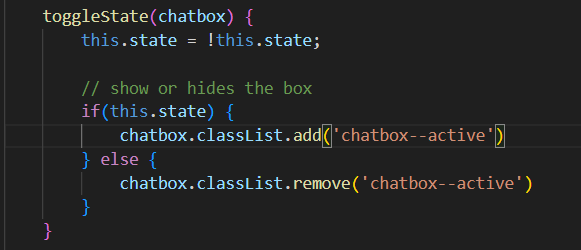
* The constructor initializes the chatbox by selecting elements from the HTML DOM using ‘document.querySelector’.
* ‘openButton’ refers to the button that toggles the chatbox visibility.
* ‘chatBox’ is the main chat interface.
* ‘sendButton’ is the button used to send messages.
* ‘state’ is a boolean to track whether the chatbox is open or closed.
* ‘messages’ is an array that stores the history of chat messages.

‘app.js’ - Part 2: Display and Toggle Functionality



‘display’ Method

* Attaches event listeners to handle clicks on the ‘openButton’ and ‘sendButton’.
* Also listens for the "Enter" key press to trigger the message send functionality.



‘toggleState’ Method

* Toggles the chatbox visibility by adding or removing the ‘chatbox—active’ CSS class based on the ‘state’ boolean.

‘app.js’ - Part 3: Sending and Displaying Messages





‘onSendButton’ Method

* Retrieves the user input and creates a message object (‘msg1’).
* Sends the user message to the backend Flask server via a ‘fetch’ POST request.
* Receives the bot's response and updates the message history.
* Calls ‘updateChatText’ to refresh the chat display.

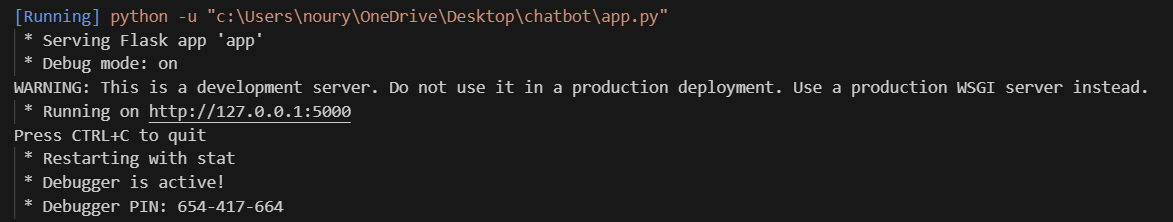


‘updateChatText’ Method

* Iterates over the ‘messages’ array in reverse to display the latest messages.
* Updates the chatbox with HTML content, differentiating between user and bot messages.

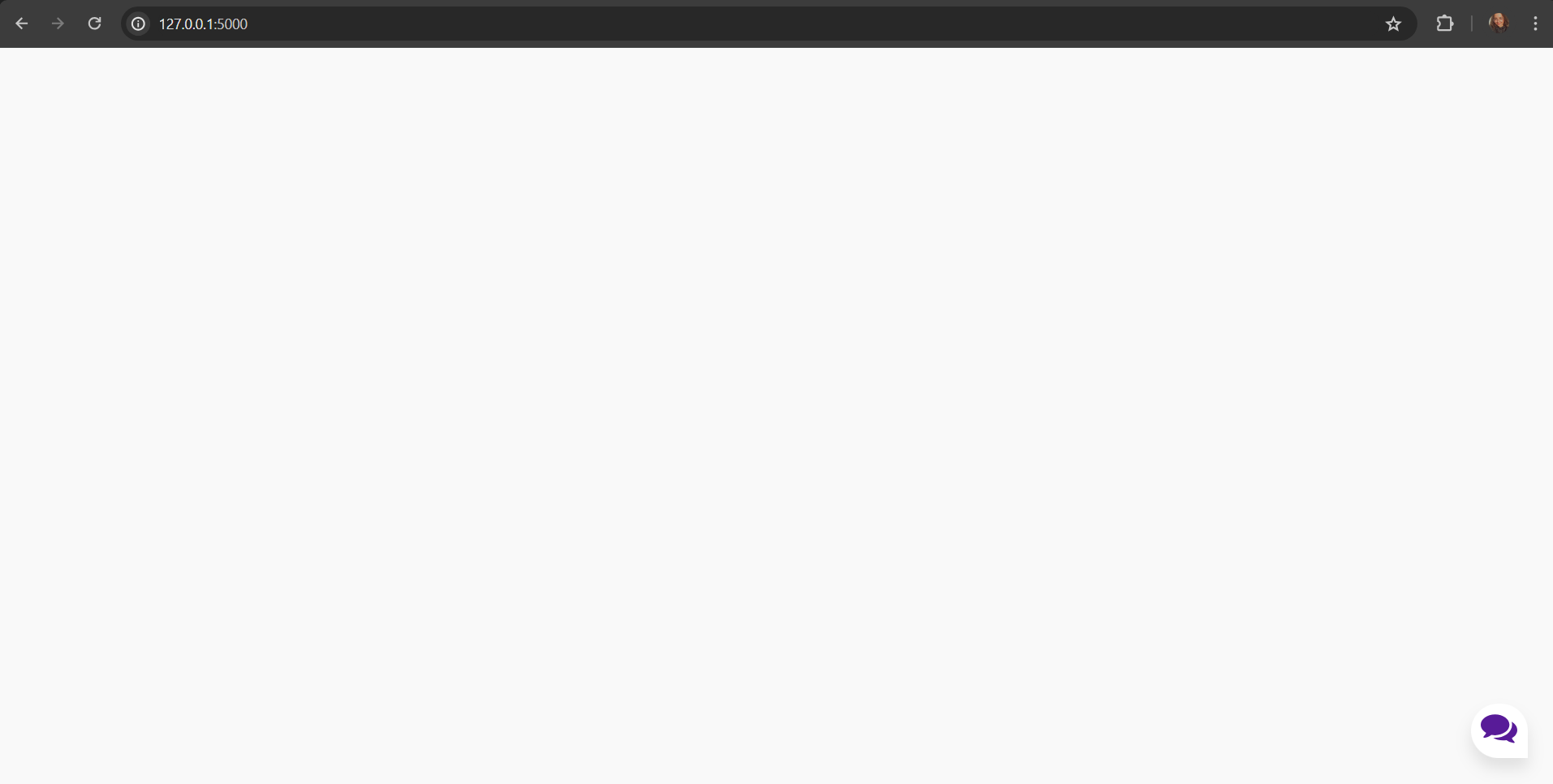
# Output

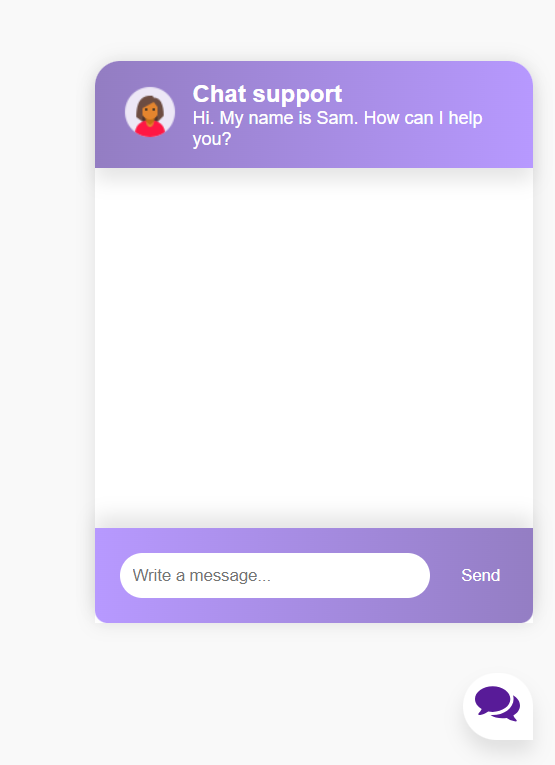
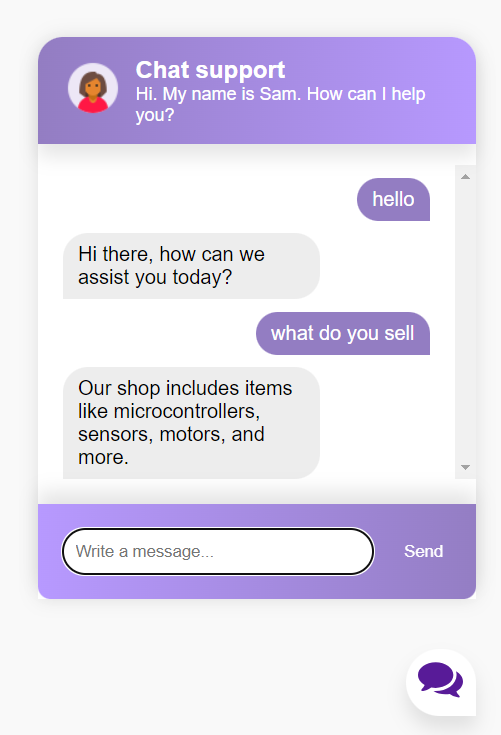
After running ‘app.py’ on VS code, this is the output:



Clicking on the link will refer the user to the front-end interface.

The following is the running front-end interface:



# Conclusion

This project combines various components—data preprocessing, model training, Flask backend, and JavaScript frontend—to create a functional AI chatbot. Each component is designed to work seamlessly with the others, allowing the chatbot to process user inputs, predict intents, and return relevant responses effectively.

The project was a significant learning experience that developed my skills in AI development, natural language processing, and web development. This internship has reinforced my career aspirations and provided a strong foundation for future projects.