

Documentation

Problem Statement:

Problem: Develop a chatbot that can understand and respond to user input in natural language. The chatbot should be capable of answering questions, providing information, and engaging in conversations on various topics.

Challenges:

1. **Natural Language Understanding:** The chatbot needs to understand and interpret user input, which can be diverse and nuanced.
2. **Response Generation:** The chatbot must generate coherent and contextually relevant responses.
3. **Training Data:** Gathering and structuring training data with intents, patterns, and responses is essential.
4. **Model Training:** Developing and training a neural network model for intent classification and response generation.
5. **User Interaction:** Creating a user-friendly interface for users to interact with the chatbot.

Design Thinking Process:

1. **Empathize:** Understand the needs and expectations of users who will interact with the chatbot. Identify potential use cases and user scenarios.
2. **Define:** Define the problem statement and the specific objectives for the chatbot. Determine the core features and capabilities the chatbot should have.
3. **Ideate:** Brainstorm ideas for chatbot functionality, including intent classification, response generation, and conversation management.
4. **Prototype:** Create a prototype of the chatbot's architecture, including the neural network model, data preprocessing, and user interface.
5. **Test:** Test the prototype with sample interactions to identify potential issues or areas for improvement.
6. **Feedback and Iterate:** Gather feedback from testing and make iterative improvements to the design and functionality.

Phases of Development:

1. Data Gathering and Preprocessing:

- Collect and structure training data (intents, patterns, responses) from "intents.json."
- Preprocess the data, tokenize sentences, and create a bag of words representation.

2. Model Development and Training:

- Define the neural network model (as seen in **model.py**) for intent classification and response generation.
- Implement custom utility functions for stemming, tokenization, and bag of words.
- Train the model using the prepared training data (**train.py**).

3. User Interface Development:

- Create a user interface where users can input text.
- Set up a Flask web application (as seen in **app.py**) to serve as the chatbot interface.

4. Chatbot Logic and Response Generation:

- Implement the chatbot's logic in **chat.py**, which includes processing user input and generating responses based on trained model predictions.

5. Testing and Debugging:

- Test the chatbot's functionality with sample interactions.
- Debug and fine-tune the model and response generation logic.

6. User Interaction and Experience:

- Improve the user interface and user experience.
- Gather user feedback and make necessary adjustments.

7. Deployment and Scaling:

- Deploy the chatbot to a server or platform where users can access it.
- Implement scalability measures to handle increased user load if necessary.

8. Maintenance and Updates:

- Regularly update the chatbot's training data and model to improve performance and accuracy.
- Address user feedback and implement new features or improvements as needed.

This development process follows a typical machine learning and chatbot development pipeline, starting from data gathering and preprocessing, model development, user interface creation, and ending with deployment and maintenance. It also incorporates the design thinking approach to ensure the chatbot meets user needs and expectations.

Libraries Used:

1. **NumPy:** NumPy is used for numerical operations and array manipulation. It's commonly used in machine learning and data processing.
2. **Random:** The **random** module is used for generating random numbers, which can be helpful for shuffling training data or making random selections, although it's not directly related to NLP.
3. **JSON:** The **json** library is used to read and parse JSON data from the "intents.json" file. It's used to load the training data for the chatbot.
4. **PyTorch:** PyTorch is a popular deep learning framework used for developing and training neural network models. In this code, PyTorch is used to define, train, and save the chatbot's neural network model.
5. **NLTK (Natural Language Toolkit):** NLTK is a library for natural language processing. It provides functions and resources for working with text data, including tokenization, stemming, and various NLP-related tasks. In this code, NLTK is used for tokenization and stemming.
6. **Flask:** Flask is a web framework for building web applications. In the provided code, Flask is used to create a web interface for the chatbot, allowing users to interact with it through a web browser.

Integration of NLP Techniques:

1. **Tokenization:** NLTK's **word_tokenize** function is used to split sentences into individual words or tokens. This is an essential NLP technique to break down text into manageable units.

2. **Stemming:** NLTK's **PorterStemmer** is used for stemming words. Stemming reduces words to their root or base form, which helps in treating similar words as the same, reducing the vocabulary size, and improving model generalization. Stemming is a common NLP technique for text preprocessing.
3. **Bag of Words (BoW):** The **bag_of_words** function is used to create a bag of words representation for text data. BoW is a technique that represents text as a vector of word counts, where each element in the vector corresponds to a unique word. This technique is often used for text classification tasks.
4. **Model Training:** The PyTorch library is used for training a neural network model for intent classification. The code defines the architecture of the neural network and uses Cross-Entropy loss for training. This integration of PyTorch and deep learning techniques is a significant part of the NLP pipeline.
5. **Intent Classification:** The code uses the trained model to classify user input into specific intents. This is a core NLP task in chatbot development, as it helps the chatbot understand the user's intention.
6. **Response Generation:** Based on the predicted intent, the chatbot generates responses. This involves selecting appropriate responses from predefined responses associated with each intent in the training data. Response selection is a key component of NLP-based chatbots.

Overall, the project code effectively integrates NLP techniques, including tokenization, stemming, and bag of words representation, along with the use of PyTorch for training a neural network model to enable the chatbot to understand user input and generate relevant responses.

Chatbot interacts with users and the web application

1. **Web Application Setup:**
 - The code sets up a web application using Flask, a Python web framework.
 - The Flask application listens for incoming HTTP requests and defines two routes: "/" and "/predict."
 - The root route ("/") serves as the landing page of the web application. It typically displays an HTML template where users can interact with the chatbot.
 - The "/predict" route is used to receive POST requests with user input in JSON format.
2. **User Interaction:**
 - When a user accesses the web application through a web browser, they see the landing page (root route).
 - The user can enter text input in a chat-like interface, typically via an input field on the web page.
 - Once the user submits a message or query, it's sent to the server as a POST request to the "/predict" route.
3. **Handling User Input:**
 - The server (Flask application) processes the POST request sent by the user.
 - It extracts the user's message from the JSON data included in the request body.

4. Chatbot Response:

- The extracted user message is then passed to the chatbot logic, specifically to the **get_response** function defined in the "chat.py" script.
- The **get_response** function analyzes the user's message, predicts the intent, and generates an appropriate response based on the trained model.
- The generated response is a text message that the chatbot intends to send back to the user.

5. Sending Response to User:

- Once the chatbot has generated a response, the server (Flask application) sends the response back to the user.
- The response is typically displayed on the web page within the chat-like interface.

6. Continuous Interaction:

- The user can continue the conversation by entering more messages or queries. Each message is sent to the server via a POST request to the "/predict" route.
- The chatbot processes each new message and generates responses in real-time.

7. Chatbot Exit:

- The chatbot interaction can be terminated by the user, typically by typing "quit" or using an exit command. This allows the user to leave the chat.

Innovative techniques or approaches used during the development

1. **Bag of Words Representation:** The code uses a Bag of Words (BoW) representation for text data. BoW is a simple and effective technique to convert text into numerical vectors. It's particularly useful for text classification tasks and allows the chatbot to understand user input by representing it as a vector of word counts.
2. **Intent Classification:** The code uses supervised learning for intent classification. It trains a neural network to predict the intent of user input. This approach allows the chatbot to understand user intentions and respond accordingly, which is a common technique in chatbot development.
3. **Stemming:** The code applies stemming to words in the training data. Stemming reduces words to their root form, which can help the model generalize better. This is a preprocessing technique used to handle variations of words.
4. **Response Confidence Threshold:** The code includes a confidence threshold for response selection. It ensures that the chatbot only provides a response if it's confident in its prediction (i.e., if the prediction probability is above a certain threshold, e.g., 0.75). If the model is uncertain, it doesn't provide a response, which can be seen as a cautious approach to improve the quality of responses.
5. **Web Application Interface:** The code integrates the chatbot with a web application using Flask. This approach provides a user-friendly interface for users to interact with the chatbot. It allows users to input messages and receive responses in a chat-like format, making it more accessible and engaging.
6. **Modular Code Structure:** The code follows a modular structure, with separate scripts for model definition (**model.py**), chatbot logic (**chat.py**), and utility functions

(`nltk_utils.py`). This modular approach enhances code maintainability and allows for easy updates and modifications.

7. **Training Data in JSON Format:** The training data (intents, patterns, and responses) is structured in a JSON file ("intents.json"). This format makes it easy to manage and update training data, which is crucial for chatbot development.
8. **Flask for Real-Time Interaction:** Flask is used to create a web interface for the chatbot. This enables real-time interaction between users and the chatbot through a web browser. Flask is a lightweight and efficient choice for web development, providing a seamless user experience.
9. **Periodic Model Saving:** The code saves the trained model's state and other relevant data to a file ("data.pth") after training. This ensures that the trained model can be easily loaded and used in the future without the need for retraining.

Dataset source and a brief description:

<https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot>

- I have take the some data from the above kaggle text file and I have trained my chatbot and succuessfully implemented it
- After analysing the data set I have understood that the above kaggle file contains the data as sets in which each question has three different formats of presentation.
- This format is present throughout the data set
- A set contains three parts: { 11 questions, 11 questions, 10 questions }

Eg: A Set contains three parts(a,b,c)

Set a:

hi, how are you doing? i'm fine. how about yourself?

i'm fine. how about yourself? i'm pretty good. thanks for asking.

i'm pretty good. thanks for asking. no problem. so how have you been?

no problem. so how have you been? i've been great. what about you?
i've been great. what about you? i've been good. i'm in school right now.

i've been good. i'm in school right now. what school do you go to?

what school do you go to? i go to pcc.

i go to pcc. do you like it there?

do you like it there? it's okay. it's a really big campus.

it's okay. it's a really big campus. good luck with school.

good luck with school. thank you very much.

Set b:

how's it going? i'm doing well. how about you?

i'm doing well. how about you? never better, thanks.

never better, thanks. so how have you been lately?

so how have you been lately? i've actually been pretty good. you?

i've actually been pretty good. you? i'm actually in school right now.

i'm actually in school right now. which school do you attend?

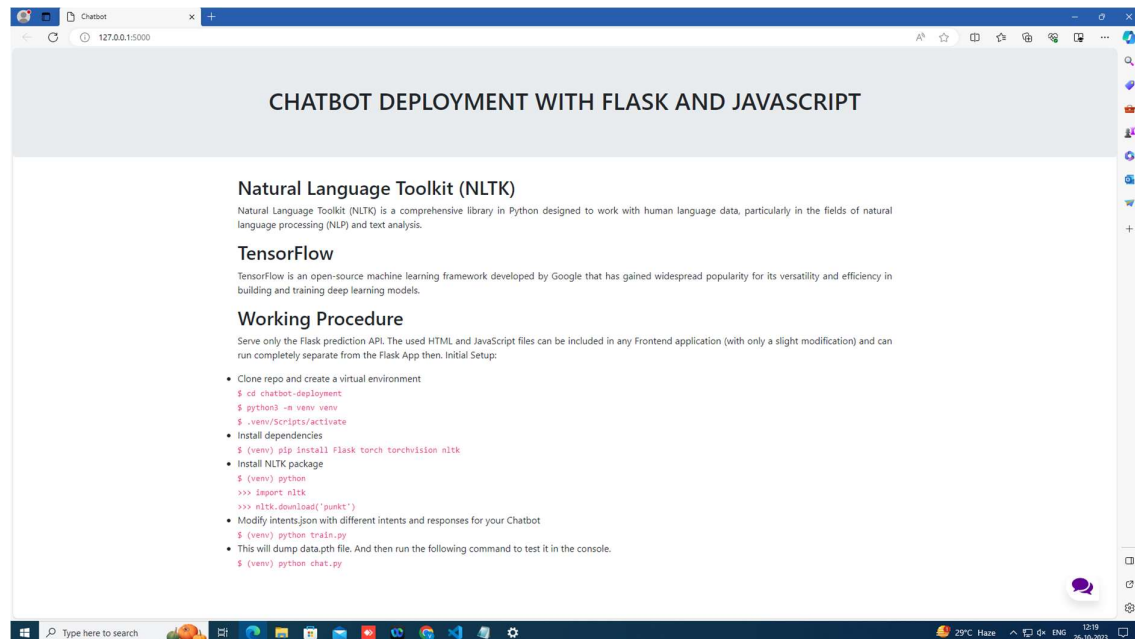
which school do you attend? i'm attending pcc right now.
i'm attending pcc right now. are you enjoying it there?
are you enjoying it there? it's not bad. there are a lot of people there.
it's not bad. there are a lot of people there. good luck with that.
good luck with that. thanks.

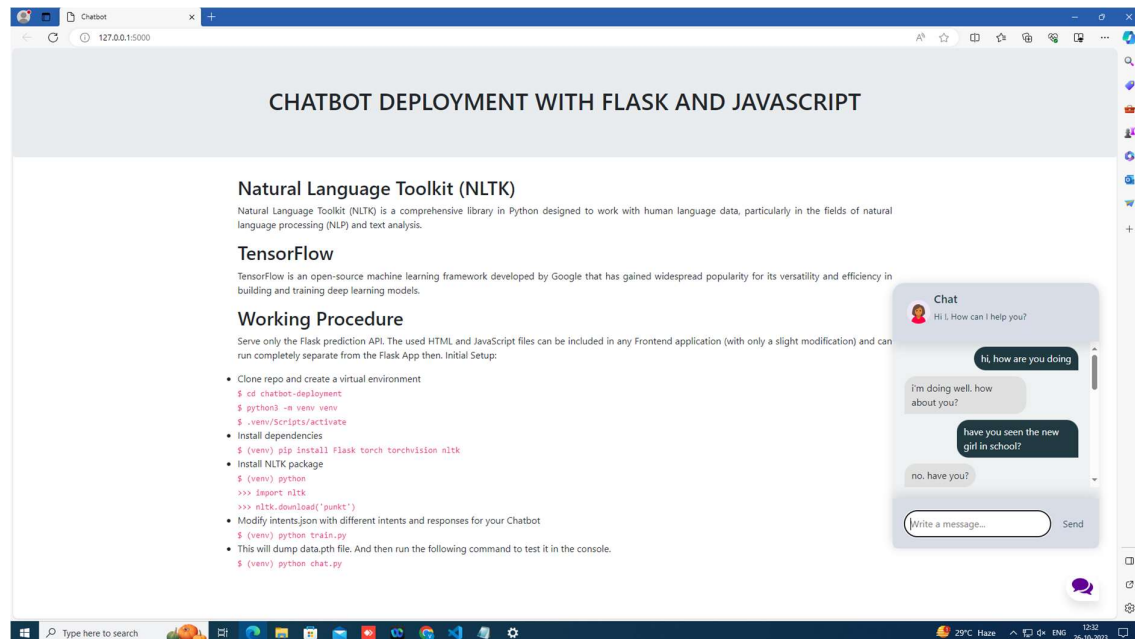
Set c:

how are you doing today? i'm doing great. what about you?
i'm doing great. what about you? i'm absolutely lovely, thank you.
i'm absolutely lovely, thank you. everything's been good with you?
everything's been good with you? i haven't been better. how about yourself?
i haven't been better. how about yourself? i started school recently.
i started school recently. where are you going to school?
where are you going to school? i'm going to pcc.
i'm going to pcc. how do you like it so far?
how do you like it so far? i like it so far. my classes are pretty good right now.
i like it so far. my classes are pretty good right now. i wish you luck.

The given dataset is further converted into json format to further process it..

Output:





The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000". The page title is "CHATBOT DEPLOYMENT WITH FLASK AND JAVASCRIPT". The content includes sections for "Natural Language Toolkit (NLTK)", "TensorFlow", and "Working Procedure". The "Working Procedure" section lists steps for cloning the repo, creating a virtual environment, installing dependencies, and running the chatbot. A chat interface is visible on the right side of the page, showing a conversation between a user and a chatbot.

CHATBOT DEPLOYMENT WITH FLASK AND JAVASCRIPT

Natural Language Toolkit (NLTK)

Natural Language Toolkit (NLTK) is a comprehensive library in Python designed to work with human language data, particularly in the fields of natural language processing (NLP) and text analysis.

TensorFlow

TensorFlow is an open-source machine learning framework developed by Google that has gained widespread popularity for its versatility and efficiency in building and training deep learning models.

Working Procedure

Serve only the Flask prediction API. The used HTML and JavaScript files can be included in any Frontend application (with only a slight modification) and can run completely separate from the Flask App then. Initial Setup:

- Clone repo and create a virtual environment

```
$ cd chatbot-deployment
$ python3 -m venv venv
$ .venv/Scripts/activate
```
- Install dependencies

```
$ (venv) pip install flask torch torchvision nltk
```
- Install NLTK package

```
$ (venv) python
>>> import nltk
>>> nltk.download('punkt')
```
- Modify intents.json with different intents and responses for your Chatbot

```
$ (venv) python train.py
```
- This will dump data.pth file. And then run the following command to test it in the console.

```
$ (venv) python chat.py
```

Chat

Hi !, How can I help you?

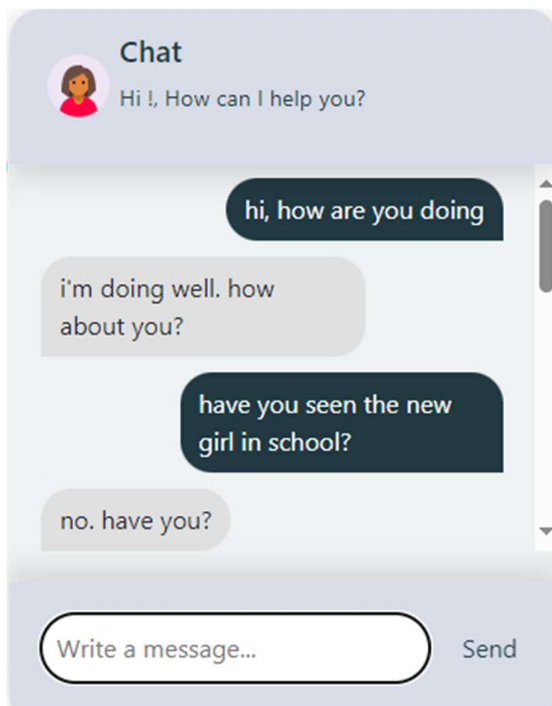
hi, how are you doing

i'm doing well. how about you?

have you seen the new girl in school?

no. have you?

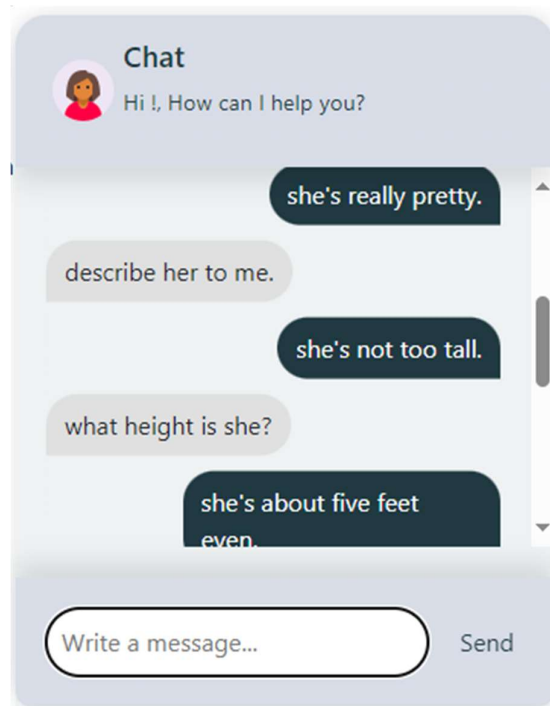
Write a message... Send



The screenshot shows a chat interface with a header "Chat" and a greeting "Hi !, How can I help you?". The chat history includes the following messages:

- User: hi, how are you doing
- Bot: i'm doing well. how about you?
- User: have you seen the new girl in school?
- Bot: no. have you?

At the bottom, there is a text input field labeled "Write a message..." and a "Send" button.



The screenshot shows a chat interface with a header "Chat" and a greeting "Hi !, How can I help you?". The chat history includes the following messages:

- User: she's really pretty.
- Bot: describe her to me.
- User: she's not too tall.
- Bot: what height is she?
- User: she's about five feet even.

At the bottom, there is a text input field labeled "Write a message..." and a "Send" button.



```

File Edit Selection View Go Run Terminal Help
chatbot-deployment

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

RIT-QALAB-13@RIT-QALAB-13 MINGW64 ~/chatbot-deployment (main)
$ source c:/Users/RIT/chatbot-deployment/.venv/Scripts/activate
(.venv)

RIT-QALAB-13@RIT-QALAB-13 MINGW64 ~/chatbot-deployment (main)
$ pip install Flask torch torchvision nltk
Requirement already satisfied: Flask in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (3.0.0)
Requirement already satisfied: torch in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (2.1.0)
Requirement already satisfied: filelock in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from torch) (3.12.4)
Requirement already satisfied: typing-extensions in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from torch) (4.8.0)
Requirement already satisfied: sympy in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from torch) (1.12)
Requirement already satisfied: networkx in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from torch) (3.2)
Requirement already satisfied: fsspec in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from torch) (2023.10.0)
Requirement already satisfied: numpy in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from torchvision) (1.26.1)
Requirement already satisfied: requests in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from torchvision) (2.31.0)
Requirement already satisfied: pillow<8.3.*,>=5.3.0 in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from torchvision) (10.1.0)
Requirement already satisfied: joblib in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from nltk) (1.3.2)
Requirement already satisfied: regex<2021.8.3 in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from nltk) (2023.10.3)
Requirement already satisfied: tqdm in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from nltk) (4.66.4)
Requirement already satisfied: colorama in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from click>=8.1.3->Flask) (0.4.6)
Requirement already satisfied: zipp>=0.5 in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from importlib-metadata>=3.6.0->Flask) (3.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from Jinja2>=3.1.2->Flask) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from requests->torchvision) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from requests->torchvision) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from requests->torchvision) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from requests->torchvision) (2023.7.22)
Requirement already satisfied: emoji>=0.19 in c:/Users/rvit/chatbot-deployment/.venv/lib/site-packages (from sympy->torch) (1.3.0)
(.venv)

RIT-QALAB-13@RIT-QALAB-13 MINGW64 ~/chatbot-deployment (main)
$ python
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import nltk
>>> nltk.download('punkt')
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\RIT\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
>>>

```

```

(.venv)
RIT-QALAB-13@RIT-QALAB-13 MINGW64 ~/chatbot-deployment (main)
$ python train.py
140 patterns
47 tags: ['about', 'activities', 'alice', 'anything', 'beach', 'better', 'bought', 'brown', 'call', 'cold/hot', 'come', 'doing', 'favorite', 'feel', 'five', 'funny', 'girl', 'good', 'like', 'luck', 'minute', 'movie', 'nice', 'ninety', 'okay', 'pretty', 'rain', 'rains', 'really', 'right', 'school', 'seen', 'short', 'smells', 'sounds', 'summer', 'tears', 'theaters', 'then', 'today', 'tomorrow', 'weather', 'weekend', 'what', 'why', 'winter', 'yesterday']
256 unique stemmed words: ['d', 'll', 'm', 're', 's', 've', 'a', 'about', 'absolut', 'activ', 'actual', 'after', 'afterward', 'ahead', 'alic', 'all', 'always', 'an', 'and', 'ani', 'answer', 'anyth', 'are', 'around', 'ask', 'at', 'attend', 'avall', 'badli', 'be', 'beach', 'beauti', 'been', 'before', 'believ', 'better', 'bismol', 'bought', 'brought', 'brown', 'but', 'ca', 'call', 'came', 'can', 'chang', 'cleaner', 'cold', 'come', 'consid', 'constantli', 'could', 'day', 'd all', 'degre', 'did', 'differe', 'do', 'doe', 'dud', 'easier', 'either', 'enjoy', 'especi', 'even', 'everi', 'eye', 'fan', 'favorit', 'feel', 'feet', 'first', 'five', 'for', 'forecast', 'found', 'fresh', 'funni', 'get', 'girl', 'go', 'good', 'go t', 'great', 'ha', 'hang', 'have', 'heard', 'hello', 'help', 'her', 'hi', 'hilari', 'hope', 'hot', 'hous', 'how', 'hundr', 'hurt', 'hyster', 'i', 'if', 'in', 'is', 'it', 'just', 'kind', 'know', 'laugh', 'lie', 'light', 'like', 'line', 'look', 'love', 'luck', 'made', 'make', 'may', 'mayb', 'me', 'mean', 'met', 'middl', 'miss', 'movi', 'muscl', 'my', 'n't', 'never', 'new', 'next', 'nice', 'nicer', 'nineti', 'no', 'not', 'notic', 'now', 'of', 'oh', 'okay', 'on', 'one', 'onli', 'or', 'our', 'out', 'outsid', 'over', 'pepto', 'perfect', 'phone', 'plan', 'pleas', 'pretti', 'prettiest', 'probabl', 'problem', 'quit', 'rain', 'rather', 'realli', 'reason', 'right', 'ruin', 'same', 'saw', 'say', 'school', 'see', 'seem', 'seen', 'she', 'short', 'should', 'sick', 'sinc', 'smell', 'so', 'someb', 'somebod', 'soon', 'sooner', 'sound', 'speak', 'start', 'stay', 'stomach', 'store', 'such', 'summer', 'suppos', 'sure', 'take', 'tall', 'tear', 'tell', 'than', 'thank', 'that', 'the', 'theater', 'there', 'thi', 'thing', 'think', 'time', 'to', 'today', 'tomorrow', 'too', 'tri', 'trip', 'true', 'ugli', 'understand', 'until', 'us', 'veri', 'wa', 'wait', 'want', 'warm', 'watch', 'way', 'we', 'weather', 'weekend', 'weird', 'well', 'went', 'were', 'what', 'when', 'where', 'whi', 'which', 'whole', 'will', 'winter', 'wish', 'with', 'would', 'wrong', 'ye', 'yeah', 'yesterday', 'yet', 'you', 'your', 'yourself']
256 47
Epoch [100/1000], Loss: 1.0307
Epoch [200/1000], Loss: 0.8130
Epoch [300/1000], Loss: 0.6035
Epoch [400/1000], Loss: 0.4005
Epoch [500/1000], Loss: 0.2116
Epoch [600/1000], Loss: 0.0801
Epoch [700/1000], Loss: 0.0000
Epoch [800/1000], Loss: 0.0000
Epoch [900/1000], Loss: 0.0000
Epoch [1000/1000], Loss: 0.0000
final loss: 0.0000
training complete. file saved to data.pth
(.venv)
RIT-QALAB-13@RIT-QALAB-13 MINGW64 ~/chatbot-deployment (main)
$ python chat.py
Let's chat! (type 'quit' to exit)
You: hi, how are doing
I'm doing great. what about you?
You: I'm pretty good. thanks for asking
everything's been good with you?
You: I've been great. what about you?
I'm actually in school right now.
You: what school do you to?
I do not understand...
You: what school do you go to?
I'm attending pcc right now.
You: do you like it there?
it's not bad. there are a lot of people there.
You: until then.
Talk to you later.
You: quit
(.venv)
RIT-QALAB-13@RIT-QALAB-13 MINGW64 ~/chatbot-deployment (main)
$

```

Project url:

<https://drive.google.com/drive/folders/1Op2qkOHuntHBXwJ6OWuKDjVrRaEDEjyq?usp=sharing>

Working video:

<https://drive.google.com/file/d/1AjiXHgfPMxDfg80r7cEI6Li0KP9CK3cU/view?usp=sharing>

Setup:

This repo currently contains the starter files.

Clone repo and create a virtual environment

```
$ cd chatbot-deployment
$ python3 -m venv venv
$ . venv/Scripts/activate
```

Install dependencies

```
$ (venv) pip install Flask torch torchvision nltk
```

Install nltk package

```
$ (venv) python
>>> import nltk
>>> nltk.download('punkt')
```

Modify intents.json with different intents and responses for your Chatbot

Run

```
$ (venv) python train.py
```

This will dump data.pth file. And then run the following command to test it in the console.

```
$ (venv) python chat.py
```