

Chatbot Project 2

Introduction

This NLP project leverages traditional Machine Learning models alongside modern Natural Language Processing techniques to create a sophisticated chatbot. Our approach integrates models such as Naive Bayes (NB) and Support Vector Machines (SVM) with advanced structures, including encoders, decoders, and attention mechanisms. The goal is to provide an interactive and intelligent system capable of understanding and responding to user inquiries about Python programming.

Objective:

The primary objective of our project is to develop a robust Q&A system that can pre-emptively resolve coding queries, thus streamlining the development process and increasing productivity. This system is designed to assist novice and experienced programmers by providing immediate solutions to their coding issues and paving the way for future tools that enhance software development efficiency.

Dataset:

Glaive Code Assistant dataset: this dataset contains ~140k code problems and solutions designed to create intelligent Python code assistants. Structured in a Q&A format, this dataset contains real-world user questions worded for coding issues from the basics of data types to more complex object-oriented programming problems and features – approximately 60% being Python. By using this dataset, developers can create automated systems that can accurately respond to the queries posed by users in any given environment. Creating an intelligent Q&A system could lead the way for new tools that solve user coding problems before they even arise, improving efficiency in development while simplifying their workflow. Whether you're a beginner or an advanced coder, this dataset has something of interest for all experience levels! This Glaive Code Assistant dataset contains ~140k code problems and solutions designed to create intelligent Python code assistants. Structured in a Q&A format, this dataset contains real-world user questions worded for coding issues from the basics of data types to more complex object-oriented programming problems and features –approximately 60% being Python. By using this dataset, developers can create automated systems that can accurately respond to the queries posed by users in any given environment. Creating an intelligent Q&A system could lead the way for new tools that solve user coding problems before they even arise, improving efficiency in development while simplifying their workflow.

Example of the Data:

	answer	question
0	Yes, you can format the output text in Bash to...	How can I output bold text in Bash? I have a B...
1	To install Python 3 on an AWS EC2 instance, yo...	How can I install Python 3 on an AWS EC2 insta...
2	You can achieve the desired time format using ...	How can I format the elapsed time from seconds...
3	Your current implementation is actually quite ...	I am trying to create a matrix of random numbe...
4	The use of 'self' in Python is quite different...	I am learning Python and have noticed extensiv...
...
9995	Implementing a "Did you mean?" feature without...	How can I implement a "Did you mean?" feature,...
9996	Yes, you can open a website via a proxy in Pyt...	In Python, I am trying to open a website via a...
9997	To extract a substring from a string after a s...	How can I extract a substring from a given str...
9998	In Python, creating an 'empty if statement' is...	How can I create an 'empty if statement' in Py...
9999	The two methods you've mentioned are used to u...	What is the difference between the two methods...

System Description:

1. Encoder-Decoder Model (with Attention Mechanism)

Purpose: This model is designed to generate contextually appropriate responses based on user inputs, using sequence-to-sequence learning typically employed in machine translation and chatbot applications. Due to resource constraints, I have taken only the first 10000 rows to train the model Due to Constraints of resources.

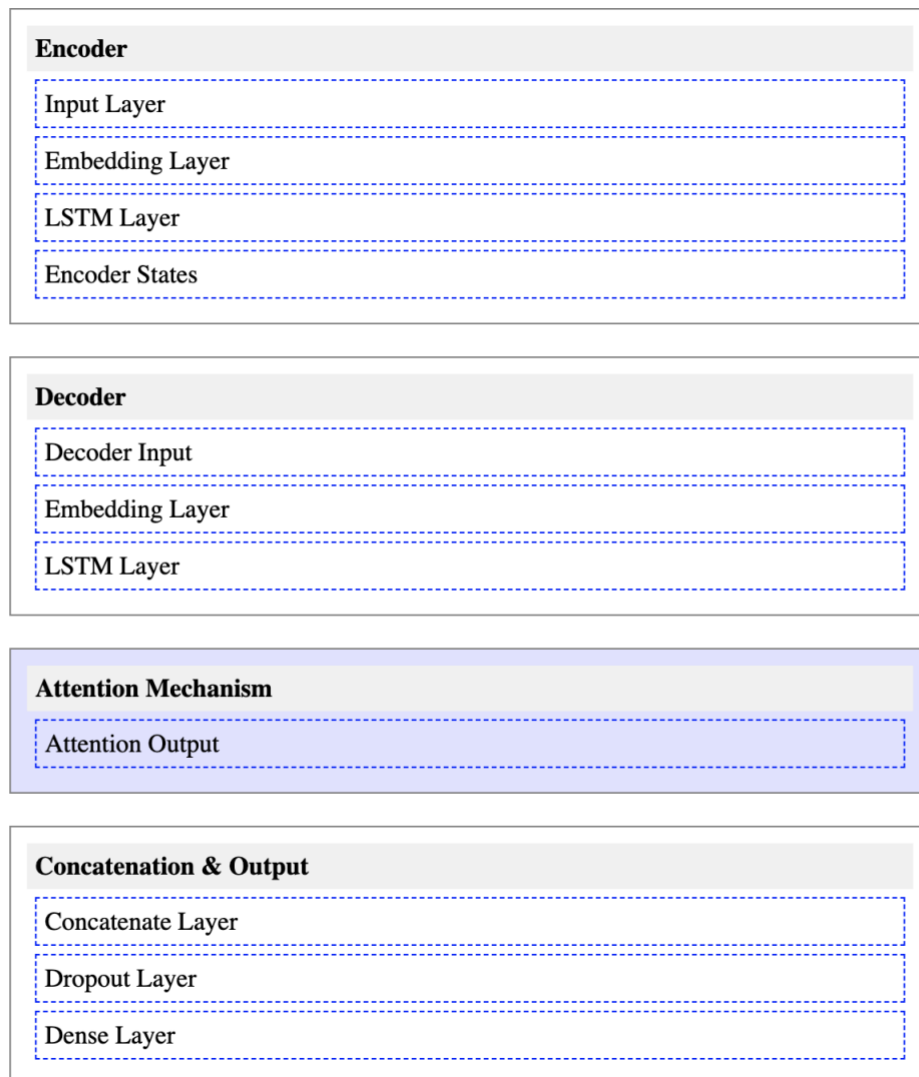
Model: "functional_3"

Layer (type)	Output Shape	Param #	Connected to
input_layer_3 (InputLayer)	(None, 60)	0	-
input_layer_4 (InputLayer)	(None, 59)	0	-
embedding_2 (Embedding)	(None, 60, 256)	11,663,872	input_layer_3[0]...
embedding_3 (Embedding)	(None, 59, 256)	11,663,872	input_layer_4[0]...
lstm_2 (LSTM)	[(None, 60, 256), (None, 256), (None, 256)]	525,312	embedding_2[0][0]
lstm_3 (LSTM)	[(None, 59, 256), (None, 256), (None, 256)]	525,312	embedding_3[0][0]... lstm_2[0][1], lstm_2[0][2]
attention_1 (Attention)	(None, 59, 256)	0	lstm_3[0][0], lstm_2[0][0]
concatenate_1 (Concatenate)	(None, 59, 512)	0	lstm_3[0][0], attention_1[0][0]
dropout_1 (Dropout)	(None, 59, 512)	0	concatenate_1[0]...
dense_1 (Dense)	(None, 59, 45562)	23,373,306	dropout_1[0][0]

Total params: 47,751,674 (182.16 MB)

Trainable params: 47,751,674 (182.16 MB)

Non-trainable params: 0 (0.00 B)



Key Components:

Encoder: Processes the input text and converts it into a context vector.

Decoder: Uses the context vector to generate output text step by step.

Attention Mechanism: Enhances the model's ability to focus on relevant parts of the input during the decoding process, improving the relevance and specificity of responses.

Integration in Chatbot:

Role: Acts as the primary response generation engine. When a user query is received, this model processes the text to generate a coherent and contextually relevant response.

Data Flow: User inputs are pre-processed, tokenized, and fed into the encoder. The decoder then constructs a response, guided by the attention mechanism, which is returned to the user.

2. Naive Bayes Classifier:

Purpose: Utilized to classify user queries into predefined categories, which can help direct the query to the most suitable response mechanism within the chatbot.

Key Components:

- **TF-IDF Vectorization:** Converts text data into a format that can be efficiently processed by the Naive Bayes algorithm.
- **Naive Bayes Algorithm:** Predicts the category of the input query based on statistical inference.

Integration in Chatbot:

- **Role:** Acts as a preliminary filter to categorize user queries and decide if they can be answered directly with stored responses or if they need to be passed to the Encoder-Decoder model for generating a custom response.
- **Data Flow:** Incoming queries are vectorized and classified. Depending on the classification result, the query is either immediately answered or further processed by the Encoder-Decoder model.

Support Vector Machine (SVM):

Purpose: Utilized for classifying user queries into predefined categories, which can help direct the query to the most suitable response mechanism within the chatbot.

Key Components:

- **TF-IDF Vectorization:** Converts text data into a format that can be efficiently processed by the Naive Bayes algorithm.
- **Naive Bayes Algorithm:** Predicts the category of the input query based on statistical inference.

Integration in Chatbot:

- **Role:** Acts as a preliminary filter to categorize user queries and decide if they can be answered directly with stored responses or if they need to be passed to the Encoder-Decoder model for generating a custom response.
- **Data Flow:** Incoming queries are vectorized and classified. Depending on the classification result, the query is either immediately answered or further processed by the Encoder-Decoder model.

Specific NLP and ML techniques used:

1 . Encoder-Decoder Model (with Attention Mechanism)

Overview: This segment implements an Encoder-Decoder architecture enhanced with an Attention mechanism, suitable for sequence-to-sequence learning, often used in machine translation and chatbot applications.

Key Components:

- **Encoder:** Uses LSTM (Long Short-Term Memory) units to process the input sequences and capture temporal dependencies. The encoder outputs a context vector representing the input sequence.

- **Decoder:** Also powered by LSTM units, it generates the output sequence step by step using the context vector and previous outputs.
- **Attention Mechanism:** Improves the model by allowing the decoder to focus on different parts of the encoder's output for each step in the output sequence, thereby capturing nuances in longer sequences.

Techniques:

- **Text Pre-processing:** Includes converting characters from Unicode to ASCII, removing non-alphabetic characters, and handling contractions to clean and standardize the text.
- **Tokenization and Padding:** Converts text to sequences of integers and ensures that sequences are padded to a consistent length for modeling.
- **Embedding:** Transforms tokenized text into dense vectors that capture semantic meanings.
- **LSTM with Dropout:** Enhances the model's generalization by randomly dropping units (dropout) during training to prevent overfitting.

2. Traditional ML Algo:

Naive Bayes Classifier:

Overview: Utilizes a Naive Bayes model for classifying text, a popular statistical technique for NLP tasks like spam detection and sentiment analysis, due to its simplicity and effectiveness.

Techniques:

- **TF-IDF Vectorization:** Transforms text into a meaningful vector of numbers based on the term frequency-inverse document frequency, which reflects the importance of words relative to the document and corpus.
- **Model Training and Prediction:** Employs the Naive Bayes algorithm, which assumes feature independence and calculates the probability of each category based on the Bayes theorem.

Support Vector Machine (SVM):

Overview: Implements an SVM for text classification. SVM is a powerful, supervised machine learning model that is effective in high-dimensional spaces and ideal for binary classification tasks.

Techniques:

- **Text Cleaning and Lemmatization:** This involves removing special characters and converting words to their base or dictionary form, reducing complexity and variability in the text.
- **SpaCy for Tokenization and Lemmatization:** Leverages the SpaCy library to process text for tokenization and lemmatization, helping to refine the text for further analysis.
- **TF-IDF Vectorization:** Like the Naive Bayes section, it prepares text data by converting it into TF-IDF vectors.
- **SVM Training:** Trains an SVM classifier with a linear kernel to distinguish between different classes based on the decision boundaries defined in the high-dimensional feature space.

3. ChatBot:

Overview: This part integrates user interaction and personalization into the chatbot, enhancing user experience by tailoring responses and maintaining a context of interaction.

Techniques:

- **Personal Data Handling:** Manages and utilizes user-specific data to personalize interactions, improving engagement.
- **Pattern Matching:** Uses regular expressions to identify user preferences (likes and dislikes) and other personal details from the conversation, which can influence the chatbot's responses.
- **Used NER** to extract the information from the User Input
- **Dynamic Response Generation:** The chatbot generates and outputs personalized responses based on the processed input and personal data.

Data Cleaning for Different Methods:

1. Unicode Normalization

Function Used: `unicode_to_ascii(s)`

- **Purpose:** Converts Unicode characters to ASCII by removing diacritics from characters. This is important for standardizing text data and avoiding issues with character encoding.
- **Method:** Utilizes Python's '`unicodedata.normalize`' function to decompose characters into their base characters and diacritics, then filters out diacritic marks using a list comprehension.

2. Lowercasing and Stripping Whitespace

Applied In: `preprocess_text(text)`

- **Purpose:** Converts all text to lowercase to reduce complexity and variability, making the text uniform. Stripping removes any leading or trailing whitespace, cleaning up the input.
- **Method:** Text strings are modified with the `.lower()` method for case normalization and `.strip()` method to remove extra spaces.

3. Removing Non-Alphabetic Characters

Applied In: `preprocess_text(text)`

- **Purpose:** Cleans the text of punctuation, special characters, and other non-essential elements that might confuse the models.
- **Method:** Uses regular expressions (`re.sub`) to replace non-word characters with spaces.

4. Exclusion of Words Containing Numbers

Applied In: `preprocess_text(text)`

- **Purpose:** Removes words that contain digits, which are generally irrelevant for natural language understanding tasks.

- **Method:** Another application of **re.sub** filters out any token that includes digits and ensures that only purely textual data is processed.

5. Handling Contractions

Applied In: `preprocess_text(text)`

- **Purpose:** Expands contractions (e.g., changing “can’t” to “cannot”) to standardize text and improve the matching process during tokenization.
- **Method:** A dictionary of contractions is used, and each word in the text is replaced with its expanded form if it exists in the dictionary.

6. Removing Stopwords

In SVM and NB Sections

- **Purpose:** Stopwords (common words like 'the', 'is', 'at') are typically removed because they offer little value in understanding the intent of a question or response.
- **Method:** Utilizes NLTK’s list of English stopwords and filters out these words from the processed text.

7. Lemmatization

Applied In: `lemmatize_text(text)` in SVM section

- **Purpose:** Reduces words to their base or dictionary form (lemma) to treat different word forms as the same, reducing the complexity of the model’s input space.
- **Method:** Utilizes SpaCy’s linguistic annotations to convert each token to its lemma.

8. Text Vectorization (TF-IDF)

In SVM and NB Sections

- **Purpose:** Converts text into a numerical format that machine learning algorithms can process by weighing the terms based on their frequency and inverse document frequency across the corpus.
- **Method:** **TfidfVectorizer** from scikit-learn transforms text into a sparse matrix of TF-IDF features.

9. Named Entity Recognition (NER) Usage

Purpose: Named Entity Recognition (NER) is used to identify and classify named entities mentioned in text into pre-defined categories such as names of people, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. In the context of your chatbot, NER enhances user interaction by recognizing personal details, which can then be used to tailor the chatbot’s responses.

Implementation Details:

- Library Used: SpaCy
- Model: "en_core_web_sm"
- Function: `extract_and_update_user_data(text, user_data)`

Operational Steps:

1. Text Processing:
 - The text input from the user is processed through SpaCy's NLP model. This model has been pre-trained on various text corpora and can perform several NLP tasks, including NER.

2. Entity Extraction:

- As the text is processed, SpaCy identifies entities and classifies them into categories like PERSON (names), ORG (organizations), LOC (locations), and DATE (dates).
- Each identified entity is tagged with its corresponding category, which the chatbot can use to understand more about the user's context or intent.

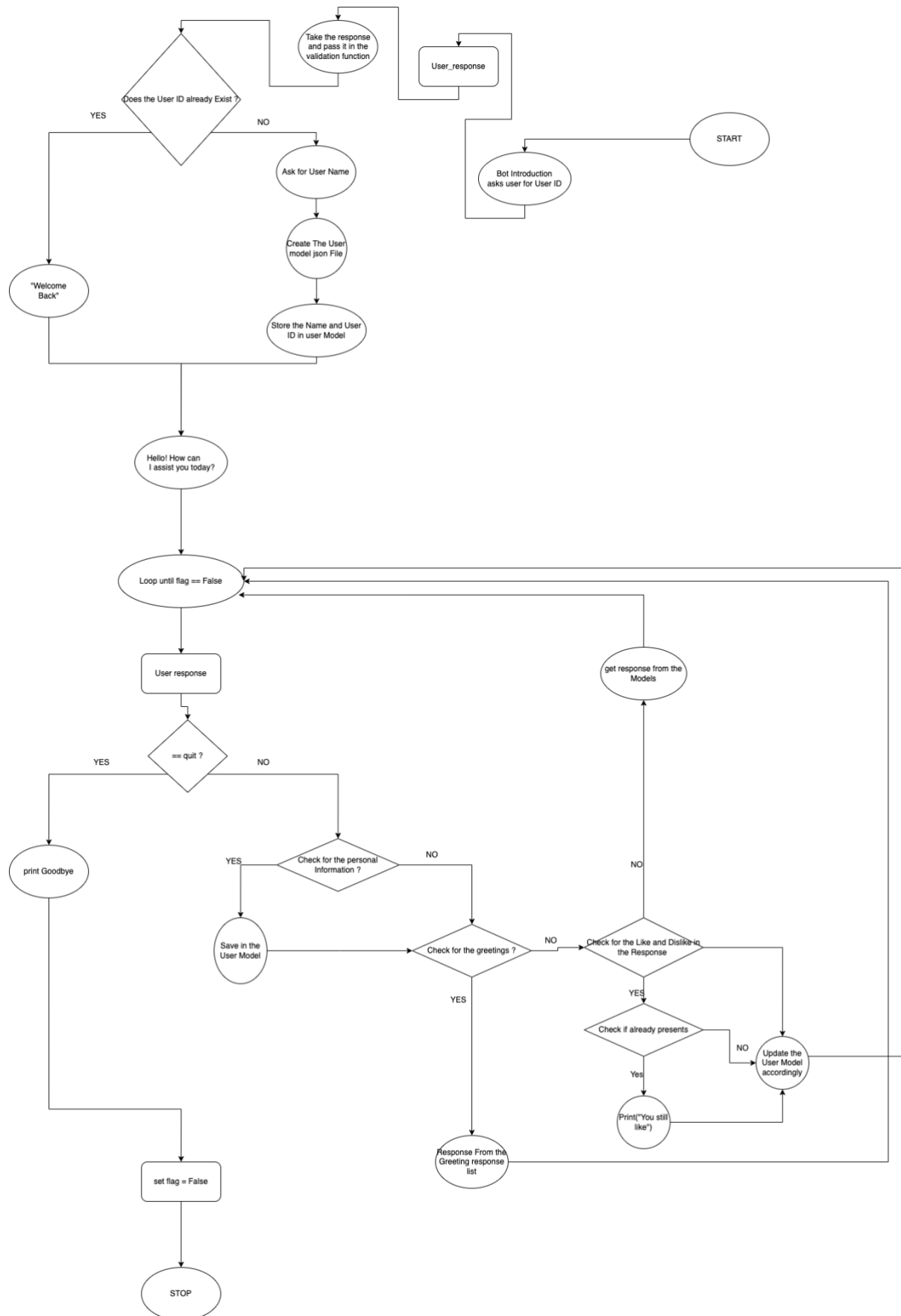
3. Utilizing Extracted Entities:

- The extracted entities are used within the chatbot to personalize the conversation. For example, if a user mentions their name or a place, the chatbot can acknowledge this in subsequent responses, creating a more engaging and personalized interaction.
- Personal details extracted via NER are stored in a user-specific data structure (user_data). This information may include the user's name, associated locations, important dates, etc.

4. Response Personalization:

- The chatbot uses the stored personal information to tailor its responses. For instance, if a user mentions their name, the chatbot might use this name in future interactions to create a more personalized and conversational experience.

A diagram of logic:



Sample Dialog of Interaction:

For Naïve Bayes:

Model loaded successfully and it can predict.
Hi, I'm your Friendly ChatBot. What's your name?
Enter your name: Denish
Welcome back, Denish!
Denish: hello
Bot: Hello! How can I assist you today?

Denish: i like coffee
Bot: You still like coffee, good to know!

Denish: I am doing my master's at University of Texas at Dallas
Bot: Interesting to hear you are involved with University of Texas.

Denish: how to use timestamp
Bot: in python, you can convert each individual timestamp in a pandas series into a string using the 'astype()' function. 'astype()' function is used to cast a pandas object to a specified datatype, here, it's used to convert timestamps to strings.

here is a step-by-step guide on how you can achieve that:

```
1, import the necessary libraries:  
``python  
import pandas as pd
```

```
2, let's assume you have a dataframe called 'df' with a column 'timestamp' as follows:  
``python  
print(df)
```

```
output:  
````  
 timestamp
0 2021-01-01 00:00:00.000000
1 2021-01-01 00:01:00.000000
2 2021-01-01 00:02:00.000000
````
```

```
3, to convert the 'timestamp' column to string, you can apply 'astype(str)' to the 'timestamp' column:  
``python  
df['timestamp'] = df['timestamp'].astype(str)
```

```
4, now, if you print your dataframe, you'll see that the 'timestamp' column has been converted to string:  
``python  
print(df)
```

```
output:  
````  
 timestamp
0 2021-01-01 00:00:00.000000
1 2021-01-01 00:01:00.000000
2 2021-01-01 00:02:00.000000
````
```

you can confirm this by checking the data type of the 'timestamp' column:

```
``python  
print(df['timestamp'].dtype)
```

```
output:  
````  
dtype('o')
```

the output 'o' stands for 'object', which is equivalent to a string in pandas. so, now your 'timestamp' column is successfully converted to string.

Denish: us of lstm keras model

Bot: this error is likely due to a version mismatch between keras and tensorflow. 'get\_default\_graph' is a method that was available in tensorflow 1.x, but has been removed in tensorflow 2.x. if you're using tensorflow 2.x, you might be using a version of keras that isn't compatible with it.

the best way to solve this problem is to use the keras api that comes bundled with tensorflow 2.x, instead of using standalone keras. you can import the necessary modules from 'tensorflow.keras' instead of 'keras'. here is how you can modify your code:

```
``python
from tensorflow.keras.models import sequential
```

## For SVM:

model loaded successfully and it can predict.  
Hi, I'm your Friendly ChatBot. What's your name?  
Enter your name: Denish  
Nice to meet you, Denish !  
Denish : hello  
Bot: Hello! How can I assist you today?

Denish : i like coffee  
Bot: It's great that you like coffee.

Denish : What is bilinear scaling ?  
Bot: it is likely that the pillow and pytorch libraries implement bilinear interpolation differently leading to the difference in results you are seeing to get the same results with both libraries you will need to use the same scaling method during both training and inference

here is a stepbystep guide on how you can compare the difference

start by importing the required libraries and defining the transformation from pil to torch and the reshape size

```
python
import numpy as np
from pil import image
import torch
import torch.nn.functional as f
from torchvision import transforms
import matplotlib.pyplot as plt
```

```
piltotorch = transforms.Lambda(lambda img: f.to_tensor(img.convert('RGB')).unsqueeze(0))
resshape = (1, 3, 224, 224)
```

open the image and convert it to a torch tensor

```
python
piling = image.open('img.png')
torchimg = piltotorch(piling)
```

scale the image using both pil and torch and then convert the pil image to a torch tensor

```
python
pilimagescaled = piling.resize((224, 224)).convert('RGB')
torchimgscaled = f.interpolate(torchimg.unsqueeze(0), size=(224, 224), mode='bilinear', align_corners=False)
```

compute the relative difference between the two scaled images

```
python
pilimagescaledontorch = piltotorch(pilimagescaled)
relativediff = torch.abs(pilimagescaledontorch - torchimgscaled).mean().item()
print('relative pixel diff: ', relativediff)
```

convert both images to numpy arrays for visualization and compute the mean absolute difference

```
python
pilimagescalednumpy = pilimagescaledontorch.cpu().numpy().transpose(0, 2, 3, 1)
torchimgscalednumpy = torchimgscaled.cpu().numpy().transpose(0, 2, 3, 1)
plt.imshow(pilimagescalednumpy)
plt.imshow(torchimgscalednumpy)
plt.savefig('diff.png')
plt.close()
```

remember that for your model to produce consistent results its important to use the same scaling method during both training and inference if you trained your model with images scaled using pillow you should also scale your images with pillow during inference.  
Denish : Describe python's opencv  
Bot: the different python interfaces for opencv cater to different needs and have evolved over time

opencv this is the original opencv library for python developed by the opencv team itself it provides a comprehensive interface for most of the functionality provided by the opencv library you can find its documentation [here](http://opencv.willowgarage.com/documentation/python/introduction.html)

cv2 this is an older version of the opencv library for python it was also created by the opencv team and it provides a similar interface to opencv but it may not contain some of the newer features or in some cases it may be deprecated

# Encoder-Decoder Model (with Attention Mechanism)

```
Denish: python library to visualize the data
1/1 ----- 0s 53ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 51ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 49ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 49ms/step
1/1 ----- 0s 49ms/step
1/1 ----- 0s 51ms/step
1/1 ----- 0s 48ms/step
1/1 ----- 0s 51ms/step
1/1 ----- 0s 56ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 55ms/step
1/1 ----- 0s 51ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 50ms/step
1/1 ----- 0s 51ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 55ms/step
1/1 ----- 0s 57ms/step
1/1 ----- 0s 54ms/step
1/1 ----- 0s 55ms/step
1/1 ----- 0s 54ms/step
1/1 ----- 0s 55ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 51ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 54ms/step
1/1 ----- 0s 54ms/step
1/1 ----- 0s 55ms/step
1/1 ----- 0s 58ms/step
1/1 ----- 0s 58ms/step
1/1 ----- 0s 58ms/step
1/1 ----- 0s 57ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 56ms/step
1/1 ----- 0s 56ms/step
1/1 ----- 0s 55ms/step
1/1 ----- 0s 54ms/step
1/1 ----- 0s 55ms/step
1/1 ----- 0s 56ms/step
1/1 ----- 0s 54ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 52ms/step
1/1 ----- 0s 53ms/step
1/1 ----- 0s 52ms/step

Bot: <so> as of the original data instead of using the reshape function to create a new plot and then create a new plot with the x and y dimensions of the data you can do this using the grayscale conversion you re using the grayscale conversion from pil import matplotlib pyplot as plt import numpy as np from matplotlib
Denish: models used for text classifications
1/1 ----- 0s 53ms/step
```

## Appendix for User-Model:

The JSON structure represents a user profile for a chatbot, detailing the user's preferences and personal information, which the chatbot can use to personalize interactions:

- **name:** A string representing the user's name is used to personalize the interaction and file naming for saving user data.
- **likes:** An array of strings representing the things that the user likes. The chatbot can use this information to relate to the user and bring up topics that the user is interested in.
- **dislikes:** An array of strings indicating what the user dislikes. This information helps the chatbot avoid topics or things that might displease the user.
- **personal\_info:** An object that stores structured personal information about the user:
  - **org** (organization): A string indicating the name of an organization with which the user is associated, in this case, "University of Texas". The chatbot can reference this in conversation for a more personalized experience.
  - **gpe** (geopolitical entity): A string denoting a geopolitical location, "Dallas" in this case. This information can be used to discuss location-specific topics or tailor the conversation to the user's locale.

The chatbot can reference and update this structure during the conversation to reflect and react to new information provided by the user.

Example :

```
{
 "name": "Denish",
 "likes": [
 "coffee",
 "tea"
],
 "dislikes": [
 "tea"
],
 "personal_info": {
 "org": "University of Texas",
 "gpe": "Dallas"
 }
}
```

- **"name": "Denish"**
  - This key-value pair holds the user's name, which the chatbot can use to address the user directly, making interactions feel more personal and engaging.
- **"likes": ["coffee", "tea"]**
  - This is an array of strings under the key "likes". It represents the things that the user, Denish, enjoys. In this example, Denish has expressed liking "coffee" and "tea". The chatbot can use this information to make conversation about these beverages or suggest related topics.

- **"dislikes": ["tea"]**
  - Under the "dislikes" key is an array that lists things the user does not enjoy. Interestingly, "tea" appears here and in the "likes" array. This inconsistency could be an input error or suggest that Denish has a complicated relationship with tea, such as liking certain types but disliking others. This would require further clarification with the user to understand their preferences better for a chatbot.
- **"personal\_info": { "org": "University of Texas", "gpe": "Dallas" }**
  - The "personal\_info" object contains structured data about the user's affiliations and location.
    - **"org": "University of Texas"**: Indicates that Denish is associated with the University of Texas, which could be an educational institution or a workplace. The chatbot can use this to discuss academic or work-related topics.
    - **"gpe": "Dallas"**: Stands for Geopolitical Entity, specifying that Denish is located in or associated with Dallas. This information might be used to customize the chat experience with local news, weather, or events.

#### Usage in Chatbot Interaction:

- **Customization**: The chatbot can use these preferences to tailor dialogues, such as discussing programming topics related to Java or referencing events at the University of Texas.
- **Recommendations**: Based on his likes, the chatbot could recommend coffee brands, Java tutorials, or local events in Dallas.
- **Engagement Strategies**: To engage Denish effectively, the chatbot might initiate conversations by mentioning updates or news related to Java programming or inquiring about his experiences at the University of Texas.

#### Implications for System Development:

- **Enhanced Personalization**: This model exemplifies how detailed user profiles can facilitate deeper personalization, enhancing user experience by making interactions more relevant and engaging.

## **Evaluation and Analysis:**

### Strengths:

- **Personalization:** The ability to learn user preferences and incorporate them into conversations using Likes and dislikes from the conversation.
- **Flexibility:** Utilizes various NLP and Advanced Machine Learning techniques for robust understanding and response generation using Heavy Models.
- **Scalability:** Designed to incorporate additional data sources and refine NLP capabilities easily.
- **Speed :** the speed of the response generation is pretty good.

### Weaknesses:

- **Context Understanding:** Limited ability to maintain context over long conversations and The generated responses are sometimes exactly not related to the topics asked.
- **Entity Recognition Limitations:** May not always accurately capture or utilize personal information from the User.
- **Dependence on Quality:** The relevance of responses heavily relies on the Training data. The more training data the more it gets better. The resources to train the model are not available and the requirements are quite higher so I have to train on the very limited amount of the Data.

Performance and accuracy can be improved if a chance is given to train on more amount of Data.

Also, integrating Complex structure in addition to current model can leverage ChatBot's performance to very good level.

### User Feedback (Survey Results of 10 people on an average basis):

1. How easily can you interact with a chatbot?

**Ease of Interaction:** 4.5 / 5

2. How relevant information about the Topic can you get from the chatbot?

**Relevance of Responses:** 4.5 / 5

3. How effective is the personalization feature of this chatbot?

**Personalization Effectiveness:** 4 / 5

User Feedback: "I find the Chatbot idea useful. We often search on Google for relevant information and spend an enormous amount of time reviewing several links. The chatbot eases the hunt by providing the most relevant information on the specified key input. Additionally, the GUI option of the Chatbot gives a modern feel to the end user and allows us to stay interested in using the application more times. One feature I would like to see is the ability to save our chat conversations for later access, thus allowing us to avoid going through the same conversations again with the Chatbot. Overall, I would rate this product 4/5, which excels in features and ease of use but could improve by introducing a conversation save feature."

Feedback indicates a positive reception to the chatbot's personalization and ease of use, pointing out areas for improvement in understanding context and response relevance.

## **Conclusion**

This chatbot demonstrates the potential of NLP and Advanced Machine Learning technologies in creating interactive, personalized user experiences.

Future work will improve context management, more training on Data and refine entity recognition to enhance conversation quality and user satisfaction.

## **References:**

- Dataset : <https://www.kaggle.com/datasets/thedevastator/glaive-python-code-qa-dataset>
- Encoder-Decoder :
  - <https://aclanthology.org/2022.acl-long.531/>
  - <https://aclanthology.org/D19-5627/>
  - <https://www.kaggle.com/code/tusharkhoche/seq2seq-lstm-encoder-decoder-implementation>