PHASE 3

VISUALIZATION CHATBOT

Date	18.10.23
Team ID	Proj_212175_Team-3
Project Name	Create a Chatbot in Python
Maximum Marks	

ABSTRACT :

Chatbot visualization is the practice of representing chatbot interactions in a visual format, allowing users and developers to gain insights into the conversations and underlying processes. Visualizations can take the form of flowcharts, timelines, or graphical user interfaces that display conversational structures and data.

VISUALIZATION CODE :

```
df['question tokens']=df['question'].apply(lambda x:len
(x.split()))
df['answer tokens']=df['answer'].apply(lambda x:len(x.s
plit()))
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
sns.set_palette('Set2')
sns.histplot(x=df['question tokens'],data=df,kde=True,a
x=ax[0])
sns.histplot(x=df['answer tokens'],data=df,kde=True,ax=
ax[1])
sns.jointplot(x='question tokens',y='answer tokens',dat
a=df,kind='kde',fill=True,cmap='YlGnBu')
plt.show()
```

```
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
ax[0].plot(history.history['loss'],label='loss',c='red'
)
```

```
ax[0].plot(history.history['val_loss'],label='val_loss'
,c = 'blue')
ax[0].set_xlabel('Epochs')
ax[1].set_xlabel('Epochs')
ax[0].set_ylabel('Loss')
ax[1].set_ylabel('Accuracy')
ax[0].set_title('Loss Metrics')
ax[1].set_title('Accuracy Metrics')
ax[1].plot(history.history['accuracy'],label='accuracy'
)
ax[1].plot(history.history['val_accuracy'],label='val_a
ccuracy')
ax[0].legend()
ax[1].legend()
plt.show()
```

EXPLANATION :

This code is written in Python and uses the Matplotlib library and Pandas library to create a figure with two subplots (side-by-side) for visualizing the training history of a machine learning model, typically a neural network. Here's a step-by-step explanation:

- df['question_tokens'] = df['question'].apply(lambda x: len(x.split()))

  This line creates a new column called 'question tokens' in the DataFrame df. It uses the apply function to apply a lambda function to each element in the 'question' column of the DataFrame. The lambda function calculates the number of tokens (words) in each 'question' by splitting the text using whitespace and then finding the length of the resulting list of words.

- df['answer tokens'] = df['answer'].apply(lambda x: len(x.split()))

Similar to the previous line, this one creates a new column called 'answer tokens' in the DataFrame df.It calculates the number of tokens (words) in each 'answer' using the same approach as for the questions.

- plt.style.use('fivethirtyeight')

Sets the style of the Matplotlib plot to the 'fivethirtyeight' style, which is a predefined style for the appearance of the plots.

- fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 5))

This line creates a Matplotlib figure with one row and two columns, resulting in a layout for two subplots side by side. The figsize parameter sets the dimensions of the figure to 20 units in width and 5 units in height.

- sns.set_palette('Set2')

Sets the color palette used by Seaborn to 'Set2'. This will affect the colors used in the subsequent plots.

- sns.histplot(x=df['question tokens'], data=df, kde=True, ax=ax[0])

This line creates a histogram plot of the 'question tokens' column from the DataFrame df. It includes a kernel density estimate (KDE) overlaid on the histogram. The resulting plot is placed in the first subplot specified by ax[0].

- sns.histplot(x=df['answer tokens'], data=df, kde=True, ax=ax[1])

Similar to the previous line, this one creates a histogram plot for the 'answer tokens' column in the second subplot specified by ax[1].

- `sns.jointplot(x='question tokens', y='answer tokens', data=df, kind='kde', fill=True, cmap='YlGnBu')`

This line creates a joint plot that displays the relationship between 'question tokens' and 'answer tokens' using a 2D kernel density estimate (KDE) plot. The `fill=True` parameter fills the contours in the KDE plot. The `cmap='YlGnBu'` parameter sets the color map for the KDE plot.

- `fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 5)`

This line initializes a figure (`fig`) and two subplots (`ax`) organized in a single row (1 row) and two columns (2 columns). The `figsize` parameter specifies the dimensions of the figure, with a width of 20 units and a height of 5 units.

- `ax[0].plot(history.history['loss'], label='loss', c='red')`

This line plots the training loss from a history object (likely obtained during training a machine learning model). It assigns a red color to the line and sets a label as "loss" for the legend.

- `ax[0].plot(history.history['val_loss'], label='val_loss', c='blue')`

This line plots the validation loss on the same subplot as the training loss but in blue. It also provides a label for the legend.

- `ax[0].set_xlabel('Epochs')`

Sets the label for the x-axis (horizontal) of the first subplot to "Epochs."

- `ax[0].set_ylabel('Loss')`

Sets the label for the y-axis (vertical) of the first subplot to "Loss."

- `ax[1].set_xlabel('Epochs')`

Sets the label for the x-axis of the second subplot to "Epochs."

- `ax[1].set_ylabel('Accuracy')`

PHASE 3

Sets the label for the y-axis of the second subplot to "Accuracy."

- `ax[1].set_title('Accuracy Metrics')`

Sets a title for the second subplot as "Accuracy Metrics."

- `ax[1].plot(history.history['accuracy'], label='accuracy')`

This line plots the training accuracy on the second subplot and assigns a label as "accuracy."

- `ax[1].plot(history.history['val_accuracy'], label='val_accuracy')`

This line plots the validation accuracy on the second subplot and provides a label for the legend.

- `ax[0].legend()`

Adds a legend to the first subplot to differentiate between "loss" and "val_loss."

- `ax[1].legend()`

Adds a legend to the second subplot to differentiate between "accuracy" and "val_accuracy."

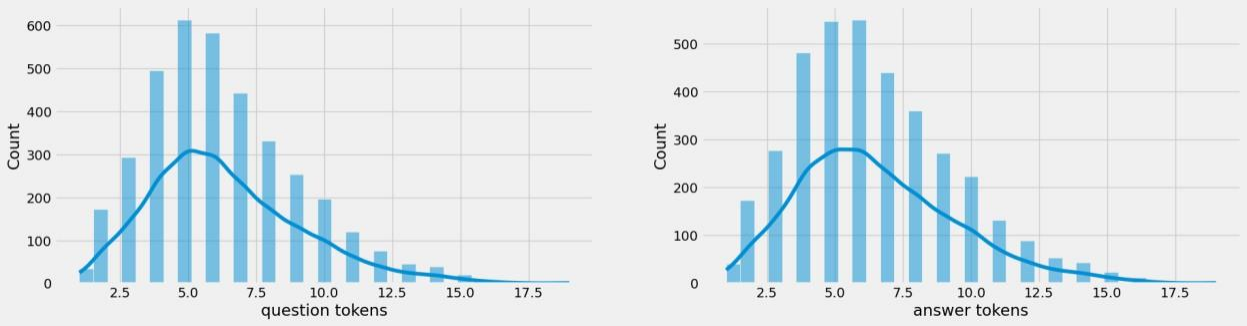
- `plt.show()`

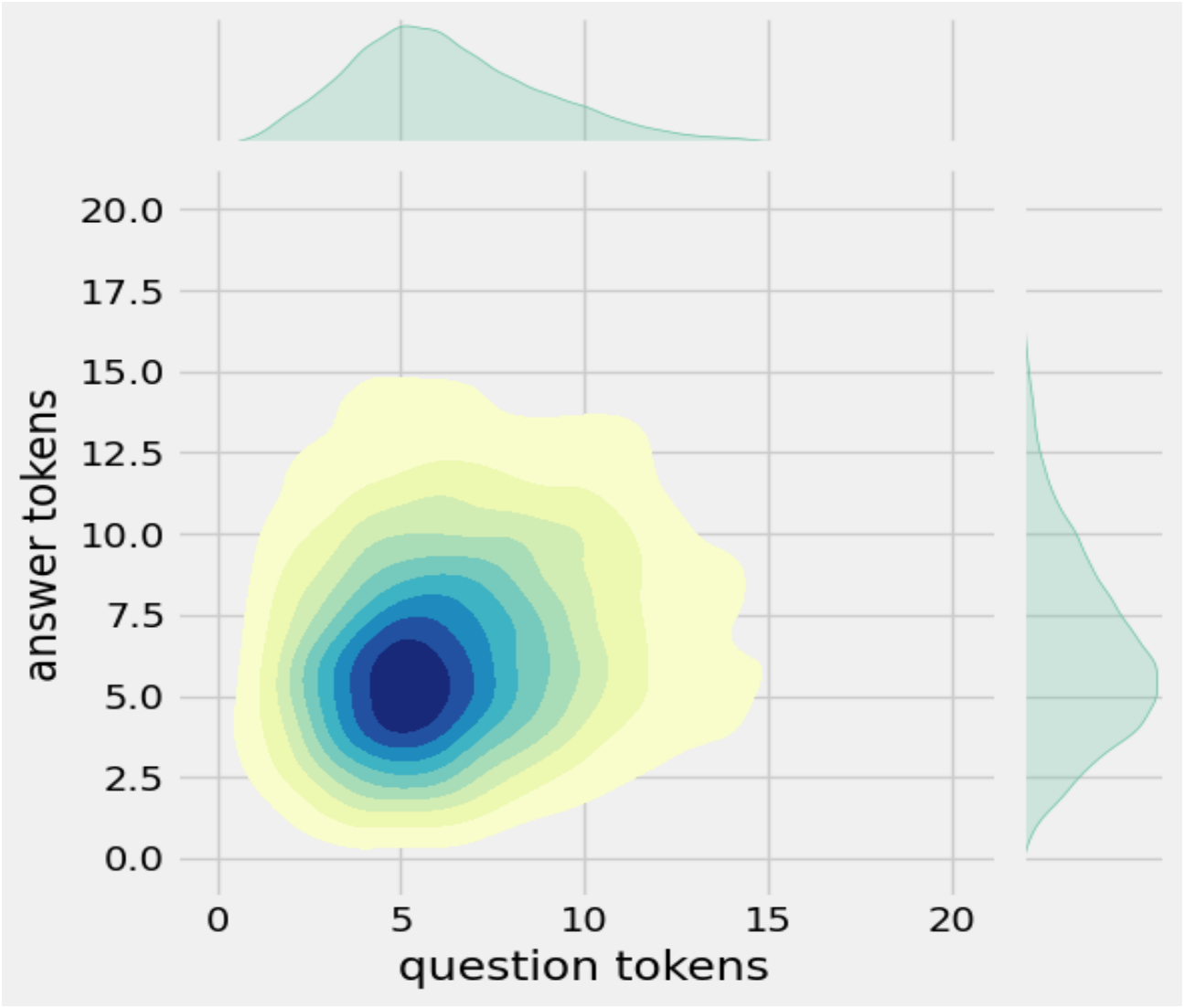
Finally, this line displays the entire figure with both subplots and their respective plots, labels, and legends.

In summary, this code is used to visualize the training and validation loss on one subplot and the training and validation accuracy on another subplot for a machine learning model over a series of epochs. The use of different colors, labels, and legends helps in understanding the training progress and model performance.

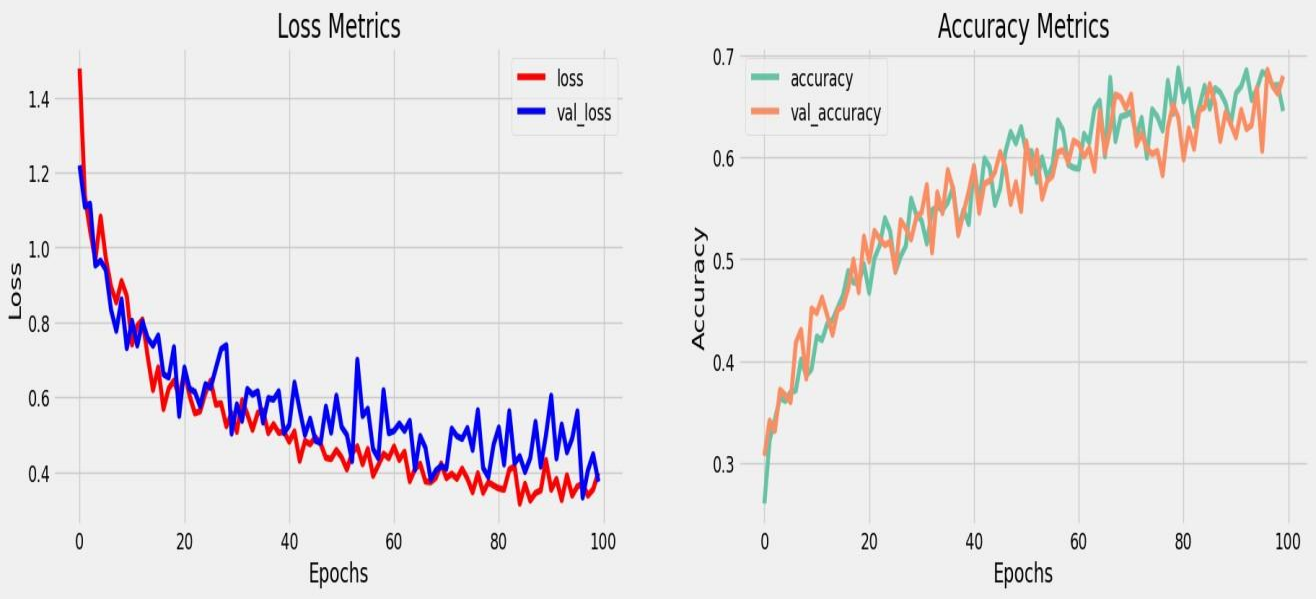
PHASE 3

OUTPUT :





PHASE 3



CONCLUSION :

In conclusion, the provided Python code demonstrates how to create avisual representation of a machine learning model's training history using Matplotlib. It generates a figure with two subplots, where onesubplot displays loss metrics (training and validation loss), and the other subplot displays accuracy metrics (training and validation accuracy) over a series of epochs.

This type of visualization is essential for monitoring and understanding the performance and training progress of machine learning models. By examining these graphs, developers and data scientists can make informed decisions about model training, spot issues like overfitting or underfitting, and make improvements to enhance the model's accuracy and reliability.

In practical applications, this visualization helps in assessing the model's convergence and identifying when it might be suitable to stoptraining, fine-tune hyperparameters, or make architectural changes. It provides valuable insights into the model's behavior, making.