Integrating a chatbot into a web app using Flask is a great way to provide a user-friendly interface for your chatbot. In this phase, I'll provide a high-level overview of the steps involved in building a Flask-based web app with chatbot integration. This assumes that you've already created the chatbot functionality in a previous phase.

1. Setup Flask Project: Start by setting up a new Flask project or using an existing one. If you're starting a new project, make sure to install Flask using pip.

bashCopy code

pip install Flask

1. Project Structure: Organize your project into folders and files. A common structure might look like this:

arduinoCopy code

/your\_project

├── app.py

├── templates/

│ ├── index.html

└── static/

├── style.css

1. Create Routes: Define routes in your app.py file to handle various interactions. At the very least, you'll need routes for rendering the chat interface and for processing user inputs.

pythonCopy code

from flask import Flask, render\_template, request, jsonify

app = Flask(\_\_name)

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/chat', methods=['POST'])

def chat():

user\_message = request.form['user\_message']

# Call your chatbot function here

bot\_response = your\_chatbot\_function(user\_message)

return jsonify({'bot\_response': bot\_response})

1. HTML Templates: Create an HTML template for your chat interface (e.g., index.html) in the templates folder. This template should include the chatbox, input field, and a mechanism for displaying responses from the chatbot.

htmlCopy code

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css') }}">

</head>

<body>

<div class="chat-container">

<div id="chat-box">

<!-- Chat messages will be displayed here -->

</div>

<div id="user-input">

<input type="text" id="user-message" placeholder="Type your message..." />

<button id="send-button">Send</button>

</div>

</div>

<script src="your\_js\_file.js"></script>

</body>

</html>

1. CSS Styling: Customize the appearance of your chat interface by creating CSS styles in the style.css file located in the static folder.
2. JavaScript: Implement JavaScript to handle user interactions, such as sending messages to the server and displaying responses. You can use JavaScript to make AJAX requests to the /chat route and update the chatbox.
3. Chatbot Integration: In the /chat route, call your chatbot function, passing the user's message as input, and obtain the chatbot's response.
4. User Session Management (Optional): If your chatbot needs to maintain state across user interactions, you may need to implement session management using Flask's session or an external database.
5. Deployment: Deploy your Flask app to a web server or a platform of your choice. Popular choices include Heroku, AWS, and PythonAnywhere.
6. Testing: Thoroughly test your web app to ensure that the chatbot functions as expected in the web interface. Handle edge cases and errors gracefully.
7. Documentation and User Guide: Provide clear documentation and a user guide for your web app, explaining how to interact with the chatbot and any specific commands or features it supports.
8. Scaling and Maintenance: As your project grows, consider scaling and maintaining the web app and chatbot, incorporating user feedback, and making necessary updates and improvements.

import pandas as pd

import numpy as np

import string

from string import digits

import matplotlib.pyplot as plt

import seaborn as sns

import re

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

from keras.layers import Input, LSTM, Embedding, Dense, Bidirectional, Concatenate, Dot, Activation, TimeDistributed

from keras.models import Model

from keras.utils import plot\_model

warnings.warn(f"A NumPy version >={np\_minversion} and <{np\_maxversion}"

data\_path = "/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt"

with open(data\_path, 'r', encoding='utf-8') as f:

lines = f.read().split('\n')

inputs = []

targets = []

num\_samples = 10000 # Number of samples to train on.

for line in lines[: min(num\_samples, len(lines) - 1)]:

input, target = line.split('\t')

inputs.append(input)

targets.append(target)

lines = pd.DataFrame({'input':inputs, 'target':targets})

lines.shape

Out[5]:

(3724, 2)

lines.head()

def cleanup(lines):

# Since we work on word level, if we normalize the text to lower case, this will reduce the vocabulary. It's easy to recover the case later.

lines.input=lines.input.apply(lambda x: x.lower())

lines.target=lines.target.apply(lambda x: x.lower())

# To help the model capture the word separations, mark the comma with special token:

lines.input=lines.input.apply(lambda x: re.sub("'", '', x)).apply(lambda x: re.sub(",", ' COMMA', x))

lines.target=lines.target.apply(lambda x: re.sub("'", '', x)).apply(lambda x: re.sub(",", ' COMMA', x))

# Clean up punctuations and digits. Such special chars are common to both domains, and can just be copied with no error.

exclude = set(string.punctuation)

lines.input=lines.input.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))

lines.target=lines.target.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))

remove\_digits = str.maketrans('', '', digits)

lines.input=lines.input.apply(lambda x: x.translate(remove\_digits))

lines.target=lines.target.apply(lambda x: x.translate(remove\_digits))

st\_tok = 'START\_'

end\_tok = '\_END'

def data\_prep(lines):

cleanup(lines)

lines.target = lines.target.apply(lambda x : st\_tok + ' ' + x + ' ' + end\_tok)

data\_prep(lines)

lines.head()

def tok\_split\_word2word(data):

def data\_stats(lines, input\_tok\_split\_fn, target\_tok\_split\_fn):

input\_tokens=set()

for line in lines.input:

for tok in input\_tok\_split\_fn(line):

if tok not in input\_tokens:

input\_tokens.add(tok)

target\_tokens=set()

for line in lines.target:

for tok in target\_tok\_split\_fn(line):

if tok not in target\_tokens:

target\_tokens.add(tok)

input\_tokens = sorted(list(input\_tokens))

target\_tokens = sorted(list(target\_tokens))

num\_encoder\_tokens = len(input\_tokens)

num\_decoder\_tokens = len(target\_tokens)

max\_encoder\_seq\_length = np.max([len(input\_tok\_split\_fn(l)) for l in lines.input])

max\_decoder\_seq\_length = np.max([len(target\_tok\_split\_fn(l)) for l in lines.target])

return input\_tokens, target\_tokens, num\_encoder\_tokens, num\_decoder\_tokens, max\_encoder\_seq\_length, max\_decoder\_seq\_length

input\_tokens, target\_tokens, num\_encoder\_tokens, num\_decoder\_tokens, max\_encoder\_seq\_length, max\_decoder\_seq\_length = data\_stats(lines, input\_tok\_split\_fn=tok\_split\_fn, target\_tok\_split\_fn=tok\_split\_fn)

print('Number of samples:', len(lines))

print('Number of unique input tokens:', num\_encoder\_tokens)

print('Number of unique output tokens:', num\_decoder\_tokens)

print('Max sequence length for inputs:', max\_encoder\_seq\_length)

print('Max sequence length for outputs:', max\_decoder\_seq\_length)

Number of samples: 3724

Number of unique input tokens: 2338

Number of unique output tokens: 2400

Max sequence length for inputs: 20

Max sequence length for outputs: 22

pad\_tok = 'PAD'

sep\_tok = ' '

special\_tokens = [pad\_tok, sep\_tok, st\_tok, end\_tok]

num\_encoder\_tokens += len(special\_tokens)

num\_decoder\_tokens += len(special\_tokens)

def vocab(input\_tokens, target\_tokens):

input\_token\_index = {}

target\_token\_index = {}

for i,tok in enumerate(special\_tokens):

input\_token\_index[tok] = i

target\_token\_index[tok] = i

offset = len(special\_tokens)

for i, tok in enumerate(input\_tokens):

input\_token\_index[tok] = i+offset

for i, tok in enumerate(target\_tokens):

target\_token\_index[tok] = i+offset

# Reverse-lookup token index to decode sequences back to something readable.

reverse\_input\_tok\_index = dict(

(i, tok) for tok, i in input\_token\_index.items())

reverse\_target\_tok\_index = dict(

(i, tok) for tok, i in target\_token\_index.items())

return input\_token\_index, target\_token\_index, reverse\_input\_tok\_index, reverse\_target\_tok\_index

input\_token\_index, target\_token\_index, reverse\_input\_tok\_index, reverse\_target\_tok\_index = vocab(input\_tokens, target\_tokens)

max\_encoder\_seq\_length = 30

max\_decoder\_seq\_length = 30

def init\_model\_inputs(lines, max\_encoder\_seq\_length, max\_decoder\_seq\_length, num\_decoder\_tokens):

encoder\_input\_data = np.zeros(

(len(lines.input), max\_encoder\_seq\_length),

dtype='float32')

decoder\_input\_data = np.zeros(

(len(lines.target), max\_decoder\_seq\_length),

dtype='float32')

decoder\_target\_data = np.zeros(

(len(lines.target), max\_decoder\_seq\_length, num\_decoder\_tokens),

dtype='float32')

return encoder\_input\_data, decoder\_input\_data, decoder\_target\_data

def vectorize(lines, max\_encoder\_seq\_length, max\_decoder\_seq\_length, num\_decoder\_tokens, input\_tok\_split\_fn, target\_tok\_split\_fn):

encoder\_input\_data, decoder\_input\_data, decoder\_target\_data = init\_model\_inputs(lines, max\_encoder\_seq\_length, max\_decoder\_seq\_length, num\_decoder\_tokens)

for i, (input\_text, target\_text) in enumerate(zip(lines.input, lines.target)):

for t, tok in enumerate(input\_tok\_split\_fn(input\_text)):

encoder\_input\_data[i, t] = input\_token\_index[tok]

encoder\_input\_data[i, t+1:] = input\_token\_index[pad\_tok]

for t, tok in enumerate(target\_tok\_split\_fn(target\_text)):

# decoder\_target\_data is ahead of decoder\_input\_data by one timestep

decoder\_input\_data[i, t] = target\_token\_index[tok]

if t > 0:

# decoder\_target\_data will be ahead by one timestep

# and will not include the start character.

decoder\_target\_data[i, t - 1, target\_token\_index[tok]] = 1.

decoder\_input\_data[i, t+1:] = target\_token\_index[pad\_tok]

decoder\_target\_data[i, t:, target\_token\_index[pad\_tok]] = 1.

return encoder\_input\_data, decoder\_input\_data, decoder\_target\_data

encoder\_input\_data, decoder\_input\_data, decoder\_target\_data = vectorize(lines, max\_encoder\_seq\_length, max\_decoder\_seq\_length, num\_decoder\_tokens, input\_tok\_split\_fn=tok\_split\_fn, target\_tok\_split\_fn=tok\_split\_fn)

def seq2seq\_attention(num\_encoder\_tokens, num\_decoder\_tokens, emb\_sz, latent\_dim):

# Define an input sequence and process it.

encoder\_inputs = Input(shape=(None,), dtype='float32')

encoder\_inputs\_ = Embedding(num\_encoder\_tokens, emb\_sz, mask\_zero=True)(encoder\_inputs)

encoder = Bidirectional(LSTM(latent\_dim, return\_state=True, return\_sequences=True)) # Bi LSTM

encoder\_outputs, state\_f\_h, state\_f\_c, state\_b\_h, state\_b\_c = encoder(encoder\_inputs\_)# Bi LSTM

state\_h = Concatenate()([state\_f\_h, state\_b\_h])# Bi LSTM

state\_c = Concatenate()([state\_f\_c, state\_b\_c])# Bi LSTM

# We discard `encoder\_outputs` and only keep the states.

encoder\_states = [state\_h, state\_c]# Bi GRU, LSTM, BHi LSTM

print(encoder\_states)

decoder\_inputs = Input(shape=(None,))

decoder\_inputs\_ = Embedding(num\_decoder\_tokens, emb\_sz, mask\_zero=True)(decoder\_inputs)

# We set up our decoder to return full output sequences,

# and to return internal states as well. We don't use the

# return states in the training model, but we will use them in inference.

decoder\_lstm = LSTM(latent\_dim\*2, return\_sequences=True, return\_state=True)# Bi LSTM

decoder\_outputs, \_, \_ = decoder\_lstm(decoder\_inputs\_, initial\_state=encoder\_states)

# Equation (7) with 'dot' score from Section 3.1 in the paper.

# Note that we reuse Softmax-activation layer instead of writing tensor calculation

print(decoder\_outputs)

print(encoder\_outputs)

att\_dot = Dot(axes=[2, 2])

attention = att\_dot([decoder\_outputs, encoder\_outputs])

att\_activation = Activation('softmax', name='attention')

attention = att\_activation(attention)

print('attention', attention)

context\_dot = Dot(axes=[2,1])

context = context\_dot([attention, encoder\_outputs])

att\_context\_concat = Concatenate()

decoder\_combined\_context = att\_context\_concat([context, decoder\_outputs])

# Has another weight + tanh layer as described in equation (5) of the paper

decoder\_dense = Dense(num\_decoder\_tokens, activation='softmax')

#decoder\_outputs = decoder\_dense(decoder\_outputs)

decoder\_outputs = decoder\_dense(decoder\_combined\_context)

# Define the model that will turn

# `encoder\_input\_data` & `decoder\_input\_data` into `decoder\_target\_data`

model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)

model.compile(optimizer=tf.keras.optimizers.Adam(lr = 0.00001), loss='categorical\_crossentropy', metrics=['acc'])

print('encoder-decoder model:')

print(model.summary())

print(encoder\_inputs)

print(encoder\_outputs)

print(encoder\_states)

encoder\_model = Model(encoder\_inputs, [encoder\_outputs] + encoder\_states)

decoder\_encoder\_inputs = Input(shape=(None, latent\_dim\*2,))

decoder\_state\_input\_h = Input(shape=(latent\_dim\*2,))# Bi LSTM

decoder\_state\_input\_c = Input(shape=(latent\_dim\*2,)) # Bi LSTM

decoder\_states\_inputs = [decoder\_state\_input\_h, decoder\_state\_input\_c]

decoder\_outputs, state\_h, state\_c = decoder\_lstm(decoder\_inputs\_, initial\_state=decoder\_states\_inputs)

decoder\_states = [state\_h, state\_c]

# Equation (7) with 'dot' score from Section 3.1 in the paper.

# Note that we reuse Softmax-activation layer instead of writing tensor calculation

attention = att\_dot([decoder\_outputs, decoder\_encoder\_inputs])

attention = att\_activation(attention)

context = context\_dot([attention, decoder\_encoder\_inputs])

decoder\_combined\_context = att\_context\_concat([context, decoder\_outputs])

# Has another weight + tanh layer as described in equation (5) of the paper

decoder\_outputs = decoder\_dense(decoder\_combined\_context)

decoder\_model = Model(

[decoder\_inputs, decoder\_encoder\_inputs] + decoder\_states\_inputs,

[decoder\_outputs, attention] + decoder\_states)

return model, encoder\_model, decoder\_model

model, encoder\_model, decoder\_model = seq2seq\_attention(num\_encoder\_tokens, num\_decoder\_tokens, emb\_sz=emb\_sz, latent\_dim=emb\_sz)

print(model.summary())

plot\_model(model, show\_shapes=True, show\_layer\_names=True)

[<KerasTensor: shape=(None, 40) dtype=float32 (created by layer 'concatenate')>, <KerasTensor: shape=(None, 40) dtype=float32 (created by layer 'concatenate\_1')>]

KerasTensor(type\_spec=TensorSpec(shape=(None, None, 40), dtype=tf.float32, name=None), name='lstm\_1/PartitionedCall:1', description="created by layer 'lstm\_1'")

KerasTensor(type\_spec=TensorSpec(shape=(None, None, 40), dtype=tf.float32, name=None), name='bidirectional/concat:0', description="created by layer 'bidirectional'")

attention KerasTensor(type\_spec=TensorSpec(shape=(None, None, None), dtype=tf.float32, name=None), name='attention/Softmax:0', description="created by layer 'attention'")

encoder-decoder model:

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_1 (InputLayer) [(None, None)] 0 []

embedding (Embedding) (None, None, 20) 46840 ['input\_1[0][0]']

input\_2 (InputLayer) [(None, None)] 0 []

bidirectional (Bidirectional) [(None, None, 40), 6560 ['embedding[0][0]']

(None, 20),

(None, 20),

(None, 20),

(None, 20)]

embedding\_1 (Embedding) (None, None, 20) 48080 ['input\_2[0][0]']

concatenate (Concatenate) (None, 40) 0 ['bidirectional[0][1]',

'bidirectional[0][3]']

concatenate\_1 (Concatenate) (None, 40) 0 ['bidirectional[0][2]',

'bidirectional[0][4]']

lstm\_1 (LSTM) [(None, None, 40), 9760 ['embedding\_1[0][0]',

(None, 40), 'concatenate[0][0]',

(None, 40)] 'concatenate\_1[0][0]']

dot (Dot) (None, None, None) 0 ['lstm\_1[0][0]',

'bidirectional[0][0]']

attention (Activation) (None, None, None) 0 ['dot[0][0]']

dot\_1 (Dot) (None, None, 40) 0 ['attention[0][0]',

'bidirectional[0][0]']

concatenate\_2 (Concatenate) (None, None, 80) 0 ['dot\_1[0][0]',

'lstm\_1[0][0]']

dense (Dense) (None, None, 2404) 194724 ['concatenate\_2[0][0]']

==================================================================================================

Total params: 305,964

Trainable params: 305,964

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

KerasTensor(type\_spec=TensorSpec(shape=(None, None), dtype=tf.float32, name='input\_1'), name='input\_1', description="created by layer 'input\_1'")

KerasTensor(type\_spec=TensorSpec(shape=(None, None, 40), dtype=tf.float32, name=None), name='bidirectional/concat:0', description="created by layer 'bidirectional'")

[<KerasTensor: shape=(None, 40) dtype=float32 (created by layer 'concatenate')>, <KerasTensor: shape=(None, 40) dtype=float32 (created by layer 'concatenate\_1')>]

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_1 (InputLayer) [(None, None)] 0 []

embedding (Embedding) (None, None, 20) 46840 ['input\_1[0][0]']

input\_2 (InputLayer) [(None, None)] 0 []

bidirectional (Bidirectional) [(None, None, 40), 6560 ['embedding[0][0]']

(None, 20),

(None, 20),

(None, 20),

(None, 20)]

embedding\_1 (Embedding) (None, None, 20) 48080 ['input\_2[0][0]']

concatenate (Concatenate) (None, 40) 0 ['bidirectional[0][1]',

'bidirectional[0][3]']

concatenate\_1 (Concatenate) (None, 40) 0 ['bidirectional[0][2]',

'bidirectional[0][4]']

lstm\_1 (LSTM) [(None, None, 40), 9760 ['embedding\_1[0][0]',

(None, 40), 'concatenate[0][0]',

(None, 40)] 'concatenate\_1[0][0]']

dot (Dot) (None, None, None) 0 ['lstm\_1[0][0]',

'bidirectional[0][0]']

attention (Activation) (None, None, None) 0 ['dot[0][0]']

dot\_1 (Dot) (None, None, 40) 0 ['attention[0][0]',

'bidirectional[0][0]']

concatenate\_2 (Concatenate) (None, None, 80) 0 ['dot\_1[0][0]',

'lstm\_1[0][0]']

dense (Dense) (None, None, 2404) 194724 ['concatenate\_2[0][0]']

==================================================================================================

Total params: 305,964

Trainable params: 305,964

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

1. Access Control:
   * Ensure that the evaluators have the necessary permissions to access your repository and evaluate your project.

After completing these steps, the evaluators will be able to access your GitHub repository and evaluate your project as needed. Make sure to communicate with them to let them know that you've added them as collaborators and that they should expect an invitation from GitHub.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_