CODES TO MAKE A FOS BOT:

chat.py:

```
import random
import json
import Paraphraser
import Inverted_Indexing
import torch
import pyttsx3
import Speech
import Query_Indexing
from model import NeuralNet
from nltk_utils import bag_of_words, tokenize
import summarizer

def get_response():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    with open(r"C:\Users\badri\Downloads\Fos_bot\Fos_bot\venv\intents.json",
'r') as json_data:
        intents = json.load(json_data)
    FILE = r"C:\Users\badri\Downloads\Fos_bot\Fos_bot\venv\data.pth"
    data = torch.load(FILE)

    input_size = data["input_size"]
    hidden_size = data["hidden_size"]
    output_size = data["output_size"]
    all_words = data['all_words']
    tags = data['tags']
    model_state = data["model_state"]

    model = NeuralNet(input_size, hidden_size, output_size).to(device)
    model.load_state_dict(model_state)
    model.eval()

    engine = pyttsx3.init()

    bot_name = "FOS"
    print(f"{bot_name}:Let's chat! (Type 'Quit' to exit)")
    Speech.SpeakText("Let's chat! FOS at your service")

    while True:
        ##sentence = "do you use credit cards?"
        # sentence = input("You: ")
        sentence = Speech.Activate()
        if sentence.find("manual input")!= -1:
            print("PLZ give user input :")
```

```
            sentence = input("You: ")
        ##Speech.Activate
        if sentence == "Quit":
            break

        sentence = tokenize(sentence)
        X = bag_of_words(sentence, all_words)
        X = X.reshape(1, X.shape[0])
        X = torch.from_numpy(X).to(device)

        output = model(X)
        _, predicted = torch.max(output, dim=1)

        tag = tags[predicted.item()]
        #print(tag)

        probs = torch.softmax(output, dim=1)
        prob = probs[0][predicted.item()]
        if prob.item() > 0.75:

            if tag == "Paraphrase":
                print(f"{bot_name}: Enter as many lines of text as you want.")
                print(f"{bot_name}: When you're done, enter a single period on
a line by itself.")
                buffer = []
                while True:
                    print("> ", end="")
                    line = input()
                    if line == ".":
                        break
                    buffer.append(line)
                multiline_string = "\n".join(buffer)
                para = Paraphraser.Paraphraser()
                para.set_text(multiline_string)
                paraphrased_text = para.paraphrase()
                print(f"{bot_name}: " + paraphrased_text)
                engine.say(paraphrased_text)
                engine.runAndWait()
                continue


            if tag == "Summarizer":
                print(f"{bot_name}: Enter as many lines of text as you want.")
                print(f"{bot_name}: When you're done, enter a single period on
a line by itself.")
                buffer = []
                while True:
                    print("> ", end="")
```

```
                    line = input()
                    if line == ".":
                        break
                    buffer.append(line)
                multiline_string = "\n".join(buffer)
                summ = summarizer.Summarizer()
                summ.set_text(multiline_string)
                summarized_text = summ.summarize()
                print(f"{bot_name}: " + summarized_text)
                engine.say(summarized_text)
                engine.runAndWait()
                continue


            if tag == "Inverted Index":
                print(f"{bot_name}: Enter as many lines of text as you want.")
                print(f"{bot_name}: When you're done, enter a single period on
a line by itself.")
                buffer = []
                while True:
                    print("> ", end="")
                    line = input()
                    if line == ".":
                        break
                    buffer.append(line)
                multiline_string = "\n".join(buffer)
                index = Inverted_Indexing.Indexing()
                sent = index.Index(multiline_string)
                print(f"{bot_name}: " + sent)
                engine.say(sent)
                engine.runAndWait()
                continue


            if tag == "Query retrieval":
                print(f"{bot_name}: Enter as many lines of text as you want.")
                print(f"{bot_name}: When you're done, enter a single period on
a line by itself.")
                buffer = []
                while True:
                    print("> ", end="")
                    line = input()
                    if line == ".":
                        break
                    buffer.append(line)
                multiline_string = "\n".join(buffer)
                query = Query_Indexing.Query()
```

```
                question  = input(f"{bot_name}: I have processed the
document/paragraph, Shoot your questions lad I am here to help\n")
                answer = query.generate_answer(question,multiline_string)
                print(f"{bot_name}: " + answer)
                engine.say(answer)
                engine.runAndWait()
                continue

            for intent in intents['intents']:
                if tag == intent["tag"]:
                    response = random.choice(intent['responses'])
                    print(f"{bot_name}: " + response)
                    engine.say(response)
                    engine.runAndWait()
        else:
            print(f"{bot_name}: I do not understand...")

def new_func(FILE):
    return FILE
get_response()
```

Speech.py:

```
import speech_recognition as sr
import pyttsx3

r = sr.Recognizer()
def SpeakText(command):
    engine = pyttsx3.init()
    voice = engine.getProperty('voices')
    engine.setProperty('voice', voice[1].id)
    engine.say(command)
    engine.runAndWait()

def Activate():
    while(1):
        try:
            # SpeakText("Speak my son")
            with sr.Microphone() as source2:
                r.adjust_for_ambient_noise(source2, duration=0.2)
                audio2 = r.listen(source2)
                MyText = r.recognize_google(audio2)
                MyText = MyText.lower()
                print("Did you say ",MyText)
                # SpeakText(MyText)
                return MyText
        except sr.RequestError as e:
            print("Could not request results; {0}".format(e))
```

```
        except sr.UnknownValueError:
            print("Unknown error occurred")




# Activate()
```

train.py:

```
import numpy as np

import random
import json

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

from nltk_utils import bag_of_words, tokenize, stem
from model import NeuralNet

import json
with open(r"C:\Users\badri\Downloads\Fos_bot\Fos_bot\venv\intents.json", 'r')
as json_data:
#with open('intents.json') as f:
    intents = json.load(json_data)

all_words = []
tags = []
xy = []
for intent in intents['intents']:
    tag = intent['tag']
    tags.append(tag)
    for pattern in intent['patterns']:
        w = tokenize(pattern)
        all_words.extend(w)
        xy.append((w, tag))

ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
all_words = sorted(set(all_words))
tags = sorted(set(tags))

print(len(xy), "patterns")
print(len(tags), "tags:", tags)
print(len(all_words), "unique stemmed words:", all_words)

X_train = []
y_train = []
```

```
for (pattern_sentence, tag) in xy:
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
    label = tags.index(tag)
    y_train.append(label)

X_train = np.array(X_train)
y_train = np.array(y_train)

num_epochs = 1000
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
print(input_size, output_size)

class ChatDataset(Dataset):

    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__(self):
        return self.n_samples

dataset = ChatDataset()
train_loader = DataLoader(dataset=dataset, batch_size=batch_size,
                          shuffle=True, num_workers=0)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = NeuralNet(input_size, hidden_size, output_size).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)

        outputs = model(words)
        loss = criterion(outputs, labels)
```

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch+1) % 100 == 0:
        print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')


print(f'Final loss: {loss.item():.4f}')

data = {
"model_state": model.state_dict(),
"input_size": input_size,
"hidden_size": hidden_size,
"output_size": output_size,
"all_words": all_words,
"tags": tags
}

FILE = "data.pth"
torch.save(data, FILE)

print(f'Training complete! File saved to {FILE}')
```

Paraphraser.py:

```
import openai

openai.api_key = "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"

def paraphrase(text):
    response = openai.Completion.create(
        engine="text-davinci-002",
        prompt=f"Paraphrase: {text}",
        max_tokens=1024,
        n=1,
        stop=None,
        temperature=0.7,
    )
    return response.choices[0].text

class Paraphraser:
    def init(self):
        self.text = """"""
```

```
    def set_text(self,text):
        self.text = text

    def paraphrase(self):
        return paraphrase(self.text)

##sent = input()
##sent = """Eula Lawrence is a playable Cryo character in Genshin
Impact.Although a descendant of the infamous and tyrannical Lawrence Clan,
Eula severed her ties with the clan and became the captain of the
Reconnaissance Company with the Knights of Favonius."""
##paraphrased_text = paraphrase(sent)
##print(paraphrased_text)
```

Intents.json:

```
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": [
        "Hi",
        "Hey",
        "How are you",
        "Is anyone there?",
        "Hello",
        "Good day",
        "speak my son",
        "good evening",
        "what's up",
        "what are you doing ",
        "Yo !",
        "hows life ",
        "Knock Knock"
      ],
      "responses": [
        "Hey :-)",
        "Hello, thanks for visiting",
        "Hi there, what can I do for you?",
        "Hi there, how can I help?",
        "I am at your service",
        "Your wish is my command",
        "How may I help you today",
        "Hello, how're you doing ",
        "FOS bot is ready",
        "FOS bot is here",
        "FOS bot at the ready",
```

```
      "Fear not for FOS bot is here",
      "FOS is here to assist you"
    ]
  },
  {
    "tag": "goodbye",
    "patterns": [
      "Bye",
      "See you later",
      "Goodbye",
      "Sayonara",
      "bye bye!",
      "See you again",
      "tata",
      "have a great day",
      "See you again",
      "take care"
    ],
    "responses": [
      "See you later, thanks for visiting",
      "Have a nice day",
      "Bye! Come back again soon.",
      "I hope I made your day",
      "Good day !",
      "See you again"
    ]
  },
  {
    "tag": "thanks",
    "patterns": [
      "Thanks",
      "Thank you",
      "That's helpful",
      "Thank's a lot!",
      "I am greatful",
      "Thanks a lot",
      "Thank you for helping me ",
      "Exactly what I wanted ",
      "Thanks for the help"
    ],
    "responses": [
      "Happy to help!",
      "Any time!",
      "My pleasure",
      "The pleasure is all mine",
      "You're welcome!",
      "FOS bot is the best",
      "FOS bot is happy"
```

```
    ]
  },
  {
    "tag": "Inverted Index",
    "patterns": [
      "sentences with word",
      "all sentences with words",
      "find all sentences with",
      "find all sentences with the word",
      "I want all the sentences with ",
      "what are all the sentences that have the word",
      "find all sentences with",
      "which line has the word",
      "contains ",
      "contians the word",
      "that contains",
      "includes the word",
      "that includes",
      "search for",
      "I want to find",
      "look for the word",
      "probe for the word",
      "look for "
    ],
    "responses": ["Here is list of all the sentences : "]
  },
  {
    "tag": "Summarizer",
    "patterns": [
      "Summarize the following",
      "summarize ",
      "can you summarize ",
      "give me a summary of ",
      "what is the summary of  ",
      "summarize ",
      "summary of "
    ],
    "responses": ["Here is a summary :", "The summarized version : "]
  },

  {
    "tag": "Paraphrase",
    "patterns": [
      "Please paraphrase the sentence",
      "Please paraphrase the document",
      "paraphrase the document",
      "paraphrase the sentence",
      "paraphrase it",
```

```
        "paraphrase this",
        "can you paraphrase",
        "paraphrase the below document",
        "make it shorter",
        "make the content shorter",
        "make the content compact",
        "make it precise ",
        "shorten it",
        "rephrase this",
        "rephrase the passage",
        "can you make this document shorter",
        "reduce the lines ",
        "too long",
        "too complex to read"
      ],
      "responses": ["Sure, here is a paraphrased version :"]
    },
    {
      "tag": "Query retrieval",
      "patterns": [
        "I have a question",
        "Question",
        "find the answer for the following",
        "answer the query",
        "I have a query",
        "query retrieval",
        "query",
        "what is the answer for",
        "find the answer",
        "what is the answer for ",
        "I need the answer for"
      ],
      "responses": ["I have found that : "]
    },
    {
      "tag": "funny",
      "patterns": [
        "Tell me a joke!",
        "Tell me something funny!",
        "Do you know a joke?"
      ],
      "responses": [
        "Why did the hipster burn his mouth? He drank the coffee before it was
cool.",
        "What did the buffalo say when his son left for college? Bison."
      ]
    }
  ]
```

```
}
```

Query_Indexing.py:

```
import torch
from transformers import AutoTokenizer, AutoModelForQuestionAnswering

# Load the model and tokenizer
model_name = "distilbert-base-uncased-distilled-squad"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForQuestionAnswering.from_pretrained(model_name)

def generate_answer(question, context):
    # Tokenize the inputs
    inputs = tokenizer.encode_plus(question, context, return_tensors="pt")

    # Generate the answer
    start_scores, end_scores = model(**inputs,return_dict=False)

    start_scores = start_scores.clone().detach()
    end_scores = end_scores.clone().detach()

    start_index = torch.argmax(start_scores)
    end_index = torch.argmax(end_scores) + 1
    answer_tokens = inputs["input_ids"][0][start_index:end_index]
    answer_tokens = answer_tokens.tolist()
    # Convert the answer tokens back to text
    answer = tokenizer.decode(answer_tokens)
    answer = answer.replace("[CLS]", "").replace("[SEP]", "").strip()
    return answer




class Query:
    def _init_(self):
        self.context = [""""""]
    def generate_answer(self,query,text):
        self.context = text
        return generate_answer(query,self.context)

# Example usage

#q = Query()
#context = """Eula Lawrence is a playable Cryo character in Genshin Impact.
```

```
#Although a descendant of the infamous and tyrannical Lawrence Clan, Eula
severed her ties with the clan and became the captain of the Reconnaissance
Company with the Knights of Favonius."""
#question = "What element is Eula?"   # Don't forget to include '?' at the end
of the question
#answer = q.generate_answer(question,context)



#answer = generate_answer(question, context)
#print(answer)
```

summarizer.py:

```
#with open("venv\summ.txt", "r") as file:
    #text1 = file.read()
from transformers import pipeline
import os
import warnings


model_name = "sshleifer/distilbart-cnn-12-6"
model_revision = "a4f8f3e"
default_max_length = 1000

def summarize(text):
    os.environ["CUDA_VISIBLE_DEVICES"] = "0"
    summarizer = pipeline("summarization", model=model_name,
revision=model_revision)
    max_l = None
    # If max_length is not specified, calculate it based on the length of the
input text
    if max_l is None:
        input_length = len(text)
        max_l = min(default_max_length, input_length)

    summary_text = summarizer(text, max_length=max_l, min_length=5,
do_sample=False)[0]['summary_text']
    return summary_text

class Summarizer:
    def init(self):
        self.text = """"""

    def set_text(self,text):
        self.text = text

    def summarize(self):
```

```
        return summarize(self.text)

#sum = Summarizer()
#sum.set_text("""A Rebellious descendant of the old aristocracy who is always
out on the battlefield.
#As one born into the old aristocracy, carrying the bloodline of sinners,
#Eula has needed a unique approach to the world to navigate the towering walls
of prejudice peacefully.
#Of course, this did not prevent her from severing ties with her clan. As the
outstanding Spindrift Knight,
#she hunts down Mondstadt's enemies in the wild to exact her unique
"vengeance".""")
#print(sum.summarize())
```

app.py:

```
from flask import Flask, render_template, request, jsonify
from flask_cors import CORS
from chat import get_response

app = Flask(__name__)
CORS(app)

@app.post("venv/app/predict")

def predict():
    text = request.get_json().get("message")
    # TODO: check if text is valid
    response = get_response(text)
    message = {"answer": response}
    return jsonify(message)

predict()
```

text.py:

```
import random
import json
import Paraphraser
import Inverted_Indexing
import torch
import pyttsx3
import Speech
import Query_Indexing
from model import NeuralNet
from nltk_utils import bag_of_words, tokenize
import summarizer
```

```
def get_response():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    with open(r"C:\Users\badri\Downloads\Fos_bot\Fos_bot\venv\intents.json",
'r') as json_data:
        intents = json.load(json_data)
    FILE = r"C:\Users\badri\Downloads\Fos_bot\Fos_bot\venv\data.pth"
    data = torch.load(FILE)

    input_size = data["input_size"]
    hidden_size = data["hidden_size"]
    output_size = data["output_size"]
    all_words = data['all_words']
    tags = data['tags']
    model_state = data["model_state"]
    model = NeuralNet(input_size, hidden_size, output_size).to(device)
    model.load_state_dict(model_state)
    model.eval()

    engine = pyttsx3.init()

    bot_name = "FOS"
    print(f"{bot_name}:Let's chat! (Type 'Quit' to exit)")
    Speech.SpeakText("Let's chat! FOS at your service")

    while True:
        ##sentence = "do you use credit cards?"
        sentence = input("You: ")
        ##sentence = Speech.Activate()
        if sentence == "Quit":
            break

        sentence = tokenize(sentence)
        X = bag_of_words(sentence, all_words)
        X = X.reshape(1, X.shape[0])
        X = torch.from_numpy(X).to(device)

        output = model(X)
        _, predicted = torch.max(output, dim=1)

        tag = tags[predicted.item()]
        #print(tag)

        probs = torch.softmax(output, dim=1)
        prob = probs[0][predicted.item()]
        if prob.item() > 0.75:

            if tag == "Paraphrase":
```

```
                print(f"{bot_name}: Enter as many lines of text as you want.")
                print(f"{bot_name}: When you're done, enter a single period on
a line by itself.")
                buffer = []
                while True:
                    print("> ", end="")
                    line = input()
                    if line == ".":
                        break
                    buffer.append(line)
                multiline_string = "\n".join(buffer)
                para = Paraphraser.Paraphraser()
                para.set_text(multiline_string)
                paraphrased_text = para.paraphrase()
                print(f"{bot_name}: " + paraphrased_text)
                engine.say(paraphrased_text)
                engine.runAndWait()
                continue


            if tag == "Summarizer":
                print(f"{bot_name}: Enter as many lines of text as you want.")
                print(f"{bot_name}: When you're done, enter a single period on
a line by itself.")
                buffer = []
                while True:
                    print("> ", end="")
                    line = input()
                    if line == ".":
                        break
                    buffer.append(line)
                multiline_string = "\n".join(buffer)
                summ = summarizer.Summarizer()
                summ.set_text(multiline_string)
                summarized_text = summ.summarize()
                print(f"{bot_name}: " + summarized_text)
                engine.say(summarized_text)
                engine.runAndWait()
                continue


            if tag == "Inverted Index":
                print(f"{bot_name}: Enter as many lines of text as you want.")
                print(f"{bot_name}: When you're done, enter a single period on
a line by itself.")
                buffer = []
                while True:
                    print("> ", end="")
```

```
                    line = input()
                    if line == ".":
                        break
                    buffer.append(line)
                multiline_string = "\n".join(buffer)
                index = Inverted_Indexing.Indexing()
                sent = index.Index(multiline_string)
                print(f"{bot_name}: " + sent)
                engine.say(sent)
                engine.runAndWait()
                continue


            if tag == "Query retrieval":
                print(f"{bot_name}: Enter as many lines of text as you want.")
                print(f"{bot_name}: When you're done, enter a single period on
a line by itself.")
                buffer = []
                while True:
                    print("> ", end="")
                    line = input()
                    if line == ".":
                        break
                    buffer.append(line)
                multiline_string = "\n".join(buffer)
                query = Query_Indexing.Query()
                question  = input(f"{bot_name}: I have processed the
document/paragraph, Shoot your questions lad I am here to help\n")
                answer = query.generate_answer(question,multiline_string)
                print(f"{bot_name}: " + answer)
                engine.say(answer)
                engine.runAndWait()
                continue

            for intent in intents['intents']:
                if tag == intent["tag"]:
                    response = random.choice(intent['responses'])
                    print(f"{bot_name}: " + response)
                    engine.say(response)
                    engine.runAndWait()
        else:
            print(f"{bot_name}: I do not understand...")
get_response()
```