import random

from textblob import TextBlob

from datetime import datetime

class UnifiedAIMLChatbot:

def \_\_init\_\_(self):

self.knowledge = self.\_init\_knowledge()

self.history = []

def \_init\_knowledge(self):

return {

"ai": {

"definition": "Artificial Intelligence (AI) refers to machines or systems that simulate human intelligence processes.",

"formulas": [

"Utility = Σ (Benefit × Probability)",

],

"subfields": ["Machine Learning", "Natural Language Processing", "Robotics", "Computer Vision"],

"examples": ["AI systems like chatbots, recommendation engines, autonomous vehicles."],

},

"machine\_learning": {

"definition": "Machine Learning (ML) is a subset of AI where systems learn patterns from data.",

"types": ["Supervised Learning", "Unsupervised Learning", "Reinforcement Learning"],

"formulas": [

"Linear Regression: y = β0 + β1x + ε",

"Logistic Regression: P(y=1|x) = 1 / (1 + e^(- (β0 + β1x)))",

"SVM: maximize margin = 2 / ||w||",

],

"example\_code": {

"Linear Regression": """\

# Linear Regression using sklearn

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(X\_train, y\_train)

preds = model.predict(X\_test)""",

"SVM": """\

# SVM using sklearn

from sklearn.svm import SVC

model = SVC(kernel='linear')

model.fit(X\_train, y\_train)

preds = model.predict(X\_test)""",

}

},

"deep\_learning": {

"definition": "Deep Learning uses neural networks with multiple layers to model complex data patterns.",

"architectures": ["Feedforward Neural Network", "Convolutional Neural Network (CNN)", "Recurrent Neural Network (RNN)"],

"example\_code": """\

# Simple PyTorch neural net

import torch.nn as nn

class Net(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.fc = nn.Linear(10, 1)

def forward(self, x):

return self.fc(x)

model = Net()""",

},

"nlp": {

"definition": "Natural Language Processing (NLP) deals with interactions between computers and human language.",

"techniques": ["Tokenization", "POS Tagging", "Named Entity Recognition", "Word Embeddings"],

"example\_code": """\

# Simple tokenization with nltk

from nltk.tokenize import word\_tokenize

text = "Hello world!"

tokens = word\_tokenize(text)

print(tokens)""",

},

"data\_science": {

"definition": "Data Science involves extracting knowledge from data using statistics, ML, visualization, and more.",

"workflow": ["Data Collection", "Cleaning", "Exploration", "Modeling", "Evaluation", "Deployment"],

"tools": ["Python", "R", "SQL", "Pandas", "NumPy", "Matplotlib"],

}

}

def \_detect\_intent(self, text):

text\_lower = text.lower()

greetings = ["hello", "hi", "hey"]

if any(g in text\_lower for g in greetings):

return "greeting"

keywords = {

"ai": ["artificial intelligence", "ai"],

"machine\_learning": ["machine learning", "ml"],

"deep\_learning": ["deep learning", "neural network", "cnn", "rnn"],

"nlp": ["natural language", "nlp"],

"data\_science": ["data science", "data analysis"],

"code": ["code", "example", "implement", "python", "pytorch", "tensorflow", "sklearn"],

"formula": ["formula", "equation", "math"],

"bye": ["bye", "exit", "quit", "goodbye"]

}

for intent, keys in keywords.items():

if any(k in text\_lower for k in keys):

return intent

# Sentiment fallback

polarity = TextBlob(text).sentiment.polarity

if polarity > 0.5:

return "positive"

if polarity < -0.5:

return "negative"

return "general"

def \_respond(self, intent, text):

if intent == "greeting":

return random.choice(["Hello! How can I assist with AI/ML/DS today?",

"Hi there! Want to know about AI, ML, or Data Science?",

"Hey! Ready to learn some AI/ML concepts?"])

if intent == "bye":

return "Goodbye! Keep learning AI and Data Science!"

if intent == "positive":

return "Glad you're excited about AI/ML! Ask me anything."

if intent == "negative":

return "Sorry to hear that. Let me know if I can help clarify something."

if intent in self.knowledge:

topic = self.knowledge[intent]

response = f"\*\*{intent.replace('\_', ' ').title()}\*\*:\n"

response += topic.get("definition", "No definition available.") + "\n"

if "formulas" in topic:

response += "Formulas:\n" + "\n".join(topic["formulas"]) + "\n"

if "types" in topic:

response += "Types:\n" + ", ".join(topic["types"]) + "\n"

if "subfields" in topic:

response += "Subfields:\n" + ", ".join(topic["subfields"]) + "\n"

if "architectures" in topic:

response += "Architectures:\n" + ", ".join(topic["architectures"]) + "\n"

if "workflow" in topic:

response += "Workflow:\n" + ", ".join(topic["workflow"]) + "\n"

if "tools" in topic:

response += "Tools:\n" + ", ".join(topic["tools"]) + "\n"

if "examples" in topic:

response += "Examples:\n" + ", ".join(topic["examples"]) + "\n"

if "example\_code" in topic:

if isinstance(topic["example\_code"], dict):

# Show all example codes for the topic

response += "Example Code Snippets:\n"

for k, v in topic["example\_code"].items():

response += f"\n{k}:\n{v}\n"

else:

response += "Example Code:\n" + topic["example\_code"] + "\n"

return response.strip()

if intent == "code":

return ("You can ask me for example code on ML topics like Linear Regression, SVM, or frameworks like PyTorch and TensorFlow.\n"

"For example, ask: 'Show me Linear Regression code' or 'Give me PyTorch example'")

if intent == "formula":

# Aggregate all formulas

all\_formulas = []

for k, v in self.knowledge.items():

if "formulas" in v:

all\_formulas.extend(v["formulas"])

return "Some useful formulas:\n" + "\n".join(all\_formulas)

return "Sorry, I didn't quite get that. You can ask me about AI, ML, DL, NLP, or Data Science."

def chat(self):

print("Welcome to the Unified AI/ML/DS Chatbot! Type 'exit' or 'bye' to quit.")

while True:

user\_input = input("You: ").strip()

if not user\_input:

continue

intent = self.\_detect\_intent(user\_input)

if intent == "bye":

print("Bot: Goodbye! Keep exploring AI and Data Science.")

break

response = self.\_respond(intent, user\_input)

print("Bot:", response, "\n")

self.history.append({"user": user\_input, "bot": response, "time": datetime.now().isoformat()})

if \_\_name\_\_ == "\_\_main\_\_":

bot = UnifiedAIMLChatbot()

bot.chat()