AI- ASSIGNMENTS

->Al Engineer Assignment: Development of a Fitness Chatbot

**Objective:** The primary goal of this assignment is to design and implement a simple chatbot that can provide fitness-related advice, including workout recommendations and dietary guidance based on user inputs. This project will test your skills in Al model development, cloud services, DevOps practices, and your foundational knowledge in mathematics as applied to machine learning and Al.

**Part-1:**

->Chatbot Development

Task: Create a chatbot using a suitable language model (e.g., GPT-3 or an open-source alternative) that can answer queries related to fitness and nutrition.

Requirements:

The chatbot should understand questions about workout routines, dietary advice, and general fitness tips.

Integrate any free fitness-related data or APIs available online to enhance the chatbot’s responses.

Develop a basic interface for interacting with the chatbot. A simple command-line interface or a basic web application is acceptable.

**->**Creating a fitness and nutrition chatbot involves several steps:

1. \*\*Choose a Language Model:\*\* You can use GPT-3, or if you prefer open-source alternatives, you might consider models like GPT-2, OpenAI’s GPT Neo, or Hugging Face’s transformers library.
2. \*\*Training Data:\*\* Collect or curate a dataset of fitness and nutrition-related questions and answers to fine-tune your language model if necessary.
3. \*\*Integrate APIs:\*\* Look for free fitness-related APIs that provide data on workout routines, dietary advice, nutrition facts, etc. Some popular ones include the Nutritionix API for nutrition data and the Workout Tracker API for exercise routines.

4. \*\*Develop the Interface:\*\*

- For a command-line interface (CLI), you can use Python and libraries like `cmd` or `argparse` to handle user inputs and display responses.

- For a basic web application, you can use frameworks like Flask or Django for the backend and HTML/CSS/JavaScript for the frontend.

1. \*\*Implement Natural Language Processing (NLP):\*\* Use NLP techniques to understand user queries and extract relevant information. Libraries like spaCy or NLTK in Python can be helpful for this.
2. \*\*Backend Logic:\*\* Implement the logic to process user queries, fetch data from APIs, and generate responses accordingly. This involves handling different types of questions related to workouts, diets, and general fitness tips.
3. \*\*Error Handling:\*\* Consider scenarios where the user input may not be clear or the API request fails. Implement error handling to provide meaningful responses in such cases.
4. \*\*Testing and Iteration:\*\* Test the chatbot thoroughly with various queries to ensure that it provides accurate and helpful responses. Iterate on the design and logic based on user feedback and performance.

Here’s a simplified example of how you might implement a basic CLI interface in Python:

```python

Import requests

Def get\_nutrition\_info(food):

Api\_key = ‘YOUR\_API\_KEY’

url = f’https://api.nutritionix.com/v1\_1/search/{food}?results=0:1&fields=item\_name,brand\_name,nf\_calories&appId=YOUR\_APP\_ID&appKey={api\_key}’

response = requests.get(url)

data = response.json()

if ‘hits’ in data and data[‘hits’]:

item = data[‘hits’][0][‘fields’]

return f”Nutrition info for {item[‘item\_name’]} – Calories: {item[‘nf\_calories’]}”

else:

return “Sorry, I couldn’t find nutrition info for that food.”

Def main():

Print(“Welcome to FitnessBot!”)

While True:

User\_input = input(“What would you like to know? (Type ‘quit’ to exit)\n”)

If user\_input.lower() == ‘quit’:

Print(“Goodbye!”)

Break

Else:

Response = get\_nutrition\_info(user\_input)

print(response)

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

main()

```

**Part-2:**

Personalization

Task: Extend the chatbot’s functionality to offer personalized workout and diet plans based on user inputs such as body type, fitness goals, and dietary restrictions.

Requirements:

Utilize any freely available dataset for workout routines and nutritional information or leverage free APIs that provide such data.

The system should ask the user a set of questions to gather their preferences and restrictions before making recommendations.

Explain the logic behind how recommendations are personalized in your documentation.

->

To extend the chatbot’s functionality to offer personalized workout and diet plans, we can follow these steps:

1. \*\*Data Collection:\*\* Gather or use freely available datasets for workout routines and nutritional information. Also, identify or utilize free APIs that provide such data.
2. \*\*User Input Gathering:\*\* Ask the user a set of questions to gather their preferences and restrictions. This can include questions about their fitness goals, current fitness level, dietary restrictions, available equipment, workout preferences, etc.

3. \*\*Personalization Logic:\*\*

- \*\*Fitness Goals:\*\* Based on the user’s input regarding their fitness goals (e.g., weight loss, muscle gain, general fitness), the chatbot can recommend specific workout routines and dietary plans tailored to those goals.

- \*\*Body Type:\*\* Understanding the user’s body type (e.g., ectomorph, mesomorph, endomorph) can help personalize workout plans to optimize results based on their genetic predispositions.

- \*\*Dietary Restrictions:\*\* Take into account any dietary restrictions or preferences provided by the user (e.g., vegetarian, vegan, gluten-free) to generate meal plans that adhere to those restrictions while still meeting nutritional needs.

- \*\*Equipment Availability:\*\* If the user specifies the equipment they have access to (e.g., gym equipment, no equipment), the chatbot can recommend workouts that utilize those resources effectively.

1. \*\*Generate Recommendations:\*\* Based on the user’s input and preferences, the chatbot generates personalized workout routines and dietary plans. This can involve querying APIs for workout routines, nutritional information, and recipe databases.

Here’s a simplified example of how you might implement this in Python:

```python

Def get\_personalized\_recommendations():

# Gather user input

Fitness\_goal = input(“What is your fitness goal? (weight loss/muscle gain/general fitness): “)

Body\_type = input(“What is your body type? (ectomorph/mesomorph/endomorph): “)

Dietary\_restrictions = input(“Do you have any dietary restrictions? (vegetarian/vegan/gluten-free/none): “)

Equipment\_available = input(“What equipment do you have access to? (gym equipment/no equipment): “)

# Based on user input, generate personalized recommendations

# This is a simplified example, in a real-world scenario, you’d query APIs or databases for recommendations

Workout\_plan = f”Here’s your personalized workout plan for {fitness\_goal} goals and {body\_type} body type.”

Diet\_plan = f”Here’s your personalized diet plan for {fitness\_goal} goals and {dietary\_restrictions} dietary restrictions.”

Return workout\_plan, diet\_plan

Def main():

Print(“Welcome to FitnessBot!”)

While True:

User\_input = input(“Would you like personalized recommendations? (yes/no): “)

If user\_input.lower() == ‘no’:

Print(“Goodbye!”)

Break

Elif user\_input.lower() == ‘yes’:

Workout\_plan, diet\_plan = get\_personalized\_recommendations()

Print(workout\_plan)

Print(diet\_plan)

Else:

Print(“Invalid input. Please enter ‘yes’ or ‘no’.”)

If \_\_name\_\_ == “\_\_main\_\_”:

Main()

```

**Part-3**

Deployment and DevOps

Task: Deploy your chatbot application to a cloud platform using any available free credits (Google Cloud, AWS, Azure, etc.). If cloud deployment is not feasible, prepare a detailed walkthrough of your application running locally. and make a loom video.

Requirements:

Set up a basic CI/CD pipeline for your application. This can be documented if actual deployment is not possible. Include instructions for setting up and running your application both locally and

on the cloud (if applicable). Prepare a brief video (using tools like Loom) demonstrating the functionality of your chatbot, including the deployment process and any cloud services used.

->

\*\*Deployment and DevOps for FitnessBot:\*\*

1. \*\*Cloud Deployment (Google Cloud Platform – GCP):\*\*

- Sign up for a GCP account and redeem any available free credits.

- Create a new project in the GCP console.

- Set up a CI/CD pipeline using Cloud Build and Cloud Functions:

- Create a Cloud Build trigger linked to your source code repository (e.g., GitHub).

- Configure the build steps in cloudbuild.yaml to install dependencies, run tests, and deploy the application.

- Deploy the application as a Cloud Function, which can be triggered by HTTP requests.

- Once deployed, access the chatbot via the provided HTTP endpoint.

2. \*\*Local Setup:\*\*

- Clone the repository containing the FitnessBot code.

- Install the necessary dependencies using pip or conda.

- Set up environment variables for any sensitive information (e.g., API keys).

- Run the application locally using Python.

- Access the chatbot via the command line interface or a web browser.

3. \*\*CI/CD Pipeline (Documentation):\*\*

- Document the steps to set up a CI/CD pipeline using tools like GitHub Actions or Jenkins.

- Define the build and deployment process in a YAML file (e.g., .github/workflows/main.yml for GitHub Actions).

- Specify the trigger conditions (e.g., push to the main branch) and the actions to perform (e.g., install dependencies, run tests, deploy).

4. \*\*Video Demonstration:\*\*

- Record a video walkthrough using a screen recording tool like Loom.

- Start by demonstrating the functionality of the FitnessBot locally.

- Show the process of setting up the CI/CD pipeline.

- Demonstrate the deployment process to GCP and accessing the deployed chatbot.

\*\*Walkthrough:\*\*

1. \*\*Local Setup:\*\*

- Clone the FitnessBot repository from GitHub.

- Navigate to the project directory.

- Install dependencies using pip:

```

Pip install -r requirements.txt

```

- Set up environment variables (if applicable).

- Run the application:

```

Python app.py

```

* Access the chatbot via the command line interface or a web browser.

2. \*\*CI/CD Pipeline (GitHub Actions):\*\*

- In your GitHub repository, navigate to the Actions tab.

- Click on “Set up a workflow yourself” to create a new workflow file.

- Define the workflow steps in YAML format, including the build and deployment process.

- Commit and push the workflow file to trigger the CI/CD pipeline.

3. \*\*Cloud Deployment (GCP – Cloud Build and Cloud Functions):\*\*

- Set up a new project in the GCP console.

- Enable the necessary APIs (Cloud Build, Cloud Functions).

- Create a Cloud Build trigger linked to your GitHub repository.

- Configure the build steps in cloudbuild.yaml.

- Deploy the application as a Cloud Function.

- Access the chatbot via the provided HTTP endpoint.