PYTHON – PROJECT REPORT

TITLE: MEDVISION: INTELLIGENT MEDICINE IDENTIFICATION AND INFORMATION SYSTEM

DOMAIN: ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

AIM:

This project leverages Optical Character Recognition (OCR), Levenshtein similarity, and speech recognition technologies to identify medicine names from text, speech, or video inputs. It provides users with detailed information about the identified medicine and supports follow-up queries through an AI-powered conversational system.

ALGORITHM:

1. **Input Handling:**
    - Accept input in three modes:
        - **Live video feed** (real-time recognition of text from labels).
        - **Audio input** (speech converted to text).
        - **Manual text input** (user-typed text).
2. **Preprocessing:**
    - For video:
        - Convert video frames to grayscale and denoise for clarity.
        - Apply adaptive thresholding to enhance text visibility.
    - For text:
        - Normalize and clean extracted text (remove special characters, standardize case).

- For speech:
    - Transcribe spoken words using speech-to-text technology.

3. **Matching:**
    - Compare the extracted text with a set of predefined medicine names using Levenshtein similarity.
    - Identify the closest match if the similarity score exceeds a defined threshold.
4. **Information Retrieval:**
    - If a valid match is found, provide detailed information about the identified medicine.
    - If no match is found, query an AI-powered system to retrieve information online.
5. **User Interaction:**
    - Display the retrieved information to the user.
    - Facilitate follow-up conversations, allowing users to ask additional questions about the identified medicine.
6. **Execution Flow:**
    - Begin with the user's input type selection.
    - Process the input accordingly.
    - Retrieve information based on text matching or AI-powered search.
    - Allow interactive conversations with the system for further queries.

TOOLS AND LIBRARIES USED:

1. **Python**
    - The primary programming language used for implementing the project logic and functionality.
2. **OpenCV**
    - A computer vision library used for real-time video processing, such as capturing live video feed, preprocessing frames, and improving the quality of text extraction.
3. **Pytesseract**

- A Python wrapper for the Tesseract OCR engine, employed to extract text from images and video frames.

4. **SpeechRecognition**
    - A library used for converting speech (audio input) into text, enabling voice commands for the system.
5. **NumPy**
    - A library for numerical computing, utilized for handling array-based operations and performing similarity calculations.
6. **Pandas**
    - A data manipulation library used for handling structured datasets (such as the list of medicine names), allowing easy data analysis and retrieval.
7. **Levenshtein**
    - A library used for computing the Levenshtein distance (edit distance) to measure the similarity between two strings, helping match extracted text with dataset entries.
8. **re (Regular Expressions)**
    - A Python module for text cleaning and preprocessing, used to remove unnecessary characters and ensure clean, usable input for further processing.
9. **Requests**
    - A library used for making HTTP requests, employed to interact with external APIs (e.g., OpenAI API) for fetching additional information.

INPUT:

1. **Video Input:**
    - Real-time feed from the user's webcam.
    - Processed to extract readable text using OCR.
2. **Audio Input:**
    - Speech recorded through the microphone.
    - Converted to text for analysis.
3. **Text Input:**

- Manually entered medicine name.

4. **API Interaction:**
   - When local information is unavailable, an external AI-powered service is used to retrieve relevant details.

PROGRAM:

```
import cv2
import pytesseract
import numpy as np
import time
import pandas as pd
import re
import requests
import Levenshtein
import speech_recognition as sr
# Tesseract OCR path setup
pytesseract.pytesseract.tesseract_cmd = r'C:/Program Files/Tesseract-OCR/tesseract.exe'
# Dataset loading
df = pd.read_csv("E:/PROJECTS AND WORKS/PROJECT IDEA SUBMISSION/SIH'24 MEDBOT/medical_dataset.csv")


# OpenAI API function for searching online
def query_openai_api(message_content):
```

```
    api_key = 'your_api_key_here'  # Replace with your OpenAI API key
    url = "https://api.openai.com/v1/chat/completions"
    headers = {
        "Authorization": f"Bearer {api_key}",
        "Content-Type": "application/json"
    }
    data = {
        "model": "gpt-4",
        "messages": [{"role": "user", "content": message_content}],
        "max_tokens": 100
    }
    response = requests.post(url, headers=headers, json=data)
    if response.status_code == 200:
        result = response.json()['choices'][0]['message']['content']
        print(result)
        return result
    else:
        print("Failed to retrieve information from OpenAI.")
        return None
# Conversation handling for the medicine
def handle_conversation(medicine_name):
    print(f"You can now ask additional questions about {medicine_name}. Type 'exit' to end the conversation.")
```

```
    while True:
        user_query = input("Ask a question: ")
        if user_query.lower() == 'exit':
            print("Ending conversation. Goodbye!")
            break
        else:
            query_openai_api(f"Regarding {medicine_name}, {user_query}")
# Text preprocessing function
def clean_extracted_text(text):
    text = text.strip()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    text = re.sub(r'\s+', ' ', text)
    return text.lower()
# Levenshtein similarity for medicine name matching
def closest_match_levenshtein(extracted_text, medical_names):
    similarities = [Levenshtein.ratio(extracted_text, name) for name in medical_names]
    max_similarity_index = np.argmax(similarities)
    return max_similarity_index, similarities[max_similarity_index]
# Improved preprocessing the frame for better OCR
def preprocess_frame(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Denoise using a median filter
    denoised = cv2.medianBlur(gray, 3)
```

```
    # Adaptive thresholding for better text visibility
    thresh = cv2.adaptiveThreshold(denoised, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
    return thresh
# Extracting text from the live video feed (only when the correct text is detected)
def extract_text_from_live_video():
    cap = cv2.VideoCapture(0)  # Open the webcam
    print("Capturing live video. Press 'q' to stop early.")
    frame_skip = 5
    frame_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        # Skip some frames to improve processing time
        frame_count += 1
        if frame_count % frame_skip != 0:
            continue
        # Display the live video feed
        cv2.imshow('Live Video Feed', frame)
        # Preprocess the frame to improve OCR results
        processed_frame = preprocess_frame(frame)
        # Extract text from the processed frame
```

```
        extracted_text_frame = pytesseract.image_to_string(processed_frame, config='--psm 6 --
oem 3')

        # Clean the extracted text
        cleaned_text = clean_extracted_text(extracted_text_frame)
        # Condition to check if the text is valid and matches the dataset
        if len(cleaned_text) > 2:
            print(f"Detected Text: {cleaned_text}")
            # Check if the extracted text matches any medicine name in the dataset
            index, similarity_score = closest_match_levenshtein(cleaned_text, df['Medicine
Name'].values)

            if similarity_score > 0.5:  # Found a valid match
                cap.release()
                cv2.destroyAllWindows()
                return df.iloc[index], similarity_score  # Return the medicine data and similarity score
        # Stop capturing if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
    return None, None
# Extract text from live audio (speech recognition)
def extract_text_from_live_audio():
    recognizer = sr.Recognizer()
```

```
    with sr.Microphone() as source:
        print("Listening for 10 seconds... Speak now!")
        try:
            audio = recognizer.record(source, duration=10)  # Limit recording to 10 seconds
            return recognizer.recognize_google(audio)
        except sr.UnknownValueError:
            return "Unable to recognize speech."
        except sr.WaitTimeoutError:
            return "No speech detected."
# Main function to handle user input and process accordingly
def main():
    input_type = input("Enter 'audio', 'video', or 'text': ").strip().lower()
    if input_type == 'audio':
        extracted_text = extract_text_from_live_audio()
        similarity_score = None
    elif input_type == 'video':
        extracted_text, similarity_score = extract_text_from_live_video()
    elif input_type == 'text':
        extracted_text = input("Please type the name of the medicine: ").strip()
        extracted_text = clean_extracted_text(extracted_text)
        similarity_score = None
    else:
        print("Invalid input type. Please enter 'audio', 'video', or 'text'.")
```

```
        return
    if extracted_text is not None:
        if similarity_score is None:  # For text or audio input, calculate similarity
            index, similarity_score = closest_match_levenshtein(extracted_text, df['Medicine Name'].values)
            extracted_text = df.iloc[index] if similarity_score > 0.5 else None
        if similarity_score > 0.5:
            medicine_name = extracted_text['Medicine Name']
            print(f"Medicine Found: {medicine_name}")
            print(extracted_text)
            print(f"Similarity Score: {similarity_score:.2f}")
            handle_conversation(medicine_name)
        else:
            print("No close match found. Searching online...")
            result = query_openai_api(
                f"Please provide information about the medicine '{extracted_text}' including its uses, side effects, and composition.")
            if result:
                print("Online Search Result:")
                print(result)
                handle_conversation(extracted_text)
    else:
        print("No text could be extracted.")
```

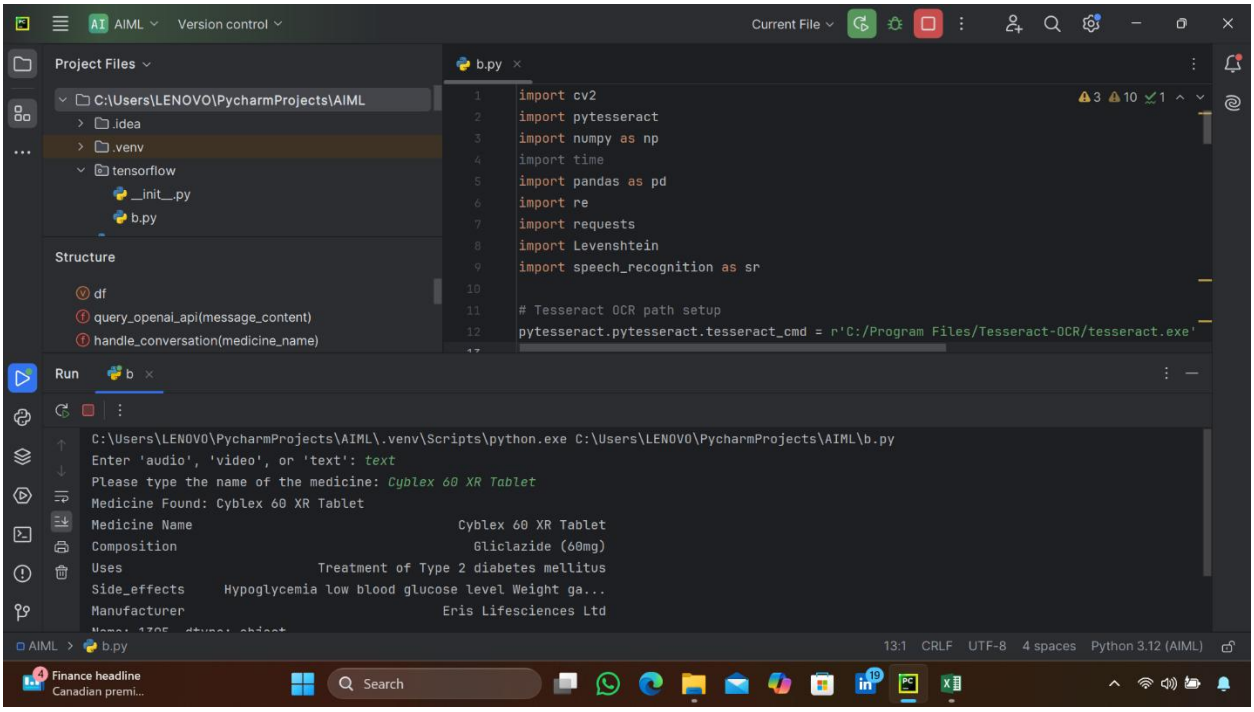
```
if __name__ == "__main__":
    main()
```

EXECUTION STEPS:

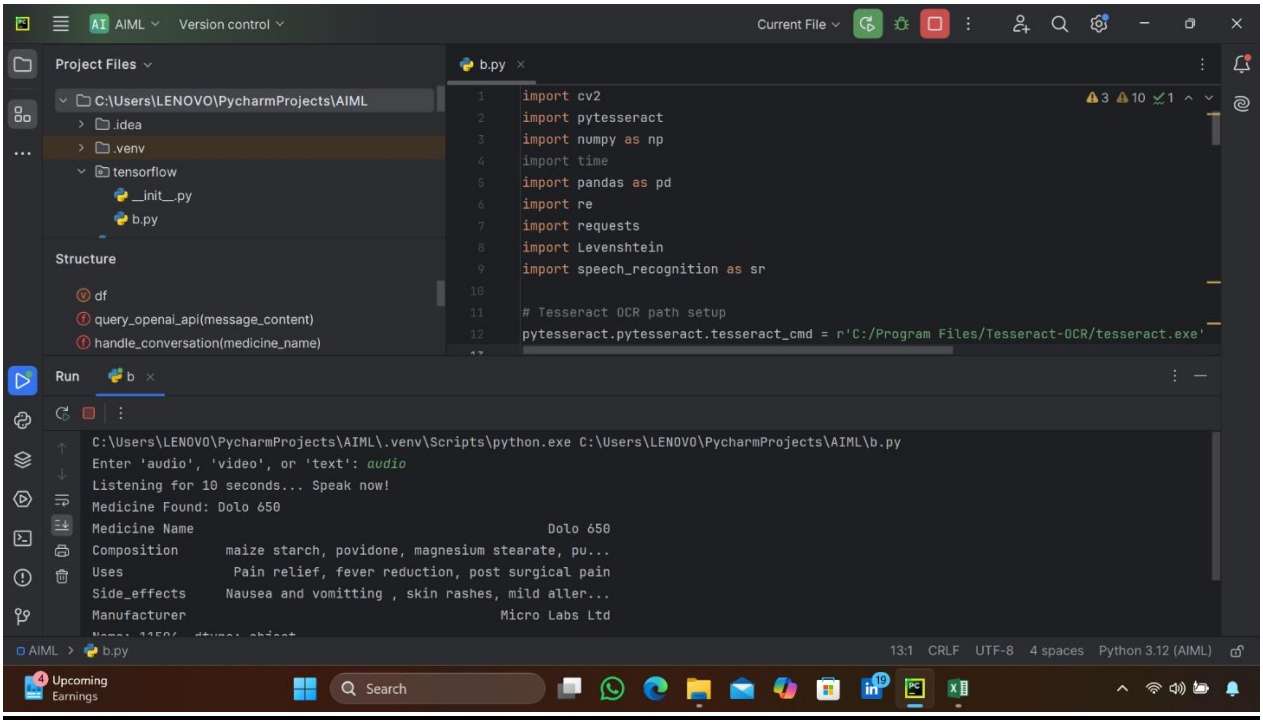
1. **Setup:**
    - Configure the OCR system for text recognition.
    - Initialize the application with predefined medicine data and AI integration.
2. **User Input:**
    - Prompt the user to select the input mode (video, audio, or text).
    - Capture the input (real-time video, speech, or manual entry).
3. **Text Processing:**
    - For video: Process frames and extract text using OCR.
    - For audio: Transcribe speech using speech-to-text.
    - Normalize and clean the extracted or entered text.
4. **Matching and Retrieval:**
    - Compare processed text with predefined medicine names.
    - If a match is found, display the relevant details.
    - If no match is found, use AI to search online for information.
5. **Interactive Feedback:**
    - Display extracted text and matching details in real-time.
    - Enable follow-up questions to provide additional insights about the identified medicine.
6. **Terminate:**
    - Allow users to exit the application or the interactive conversation.

OUTPUT:

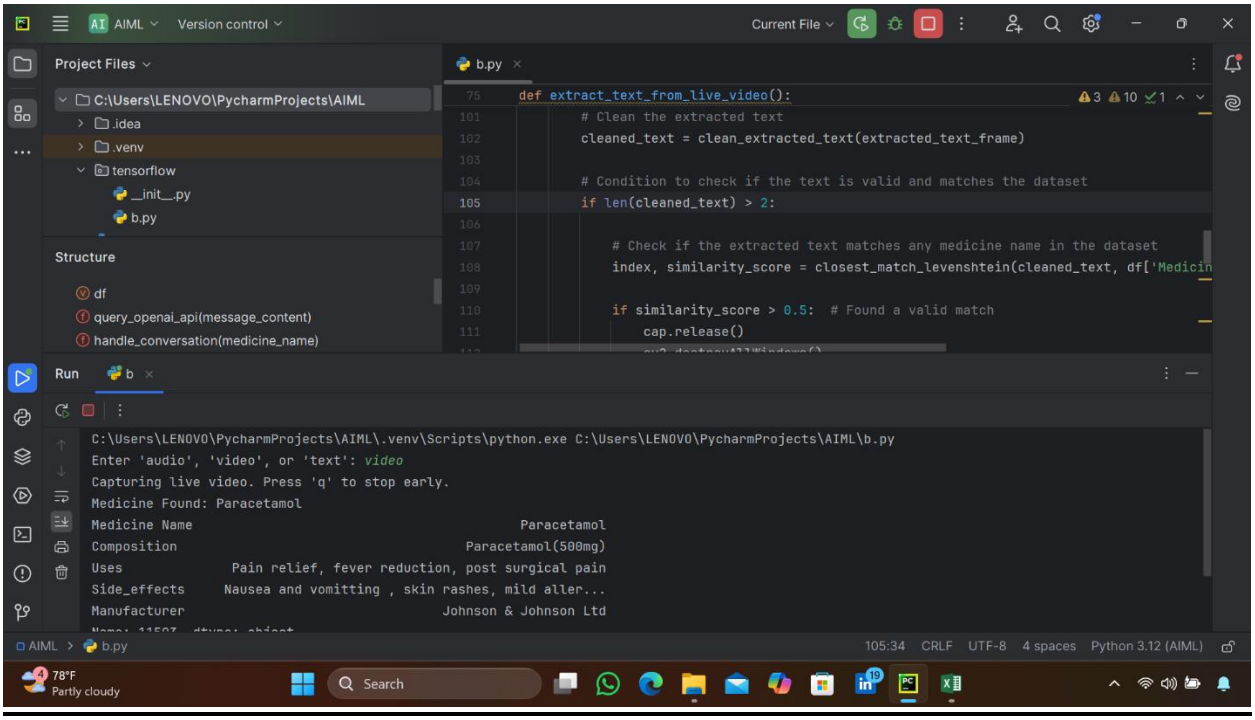
TRIAL - TEXT



TRIAL - AUDIO



TRIAL - VIDEO


AIML
Version control
Current File
Project Files
C:\Users\LENOVO\PycharmProjects\AIML
.idea
.venv
tensorflow
__init__.py
b.py
Structure
df
query_openai_api(message_content)
handle_conversation(medicine_name)
b.py
def extract_text_from_live_video():
# Clean the extracted text
cleaned_text = clean_extracted_text(extracted_text_frame)
# Condition to check if the text is valid and matches the dataset
if len(cleaned_text) > 2:
# Check if the extracted text matches any medicine name in the dataset
index, similarity_score = closest_match_levenshtein(cleaned_text, df['Medicin
if similarity_score > 0.5: # Found a valid match
cap.release()
Run
b
C:\Users\LENOVO\PycharmProjects\AIML\.venv\Scripts\python.exe C:\Users\LENOVO\PycharmProjects\AIML\b.py
Enter 'audio', 'video', or 'text': video
Capturing live video. Press 'q' to stop early.
Medicine Found: Paracetamol
Medicine Name Paracetamol
Composition Paracetamol(500mg)
Uses Pain relief, fever reduction, post surgical pain
Side_effects Nausea and vomitting , skin rashes, mild aller...
Manufacturer Johnson & Johnson Ltd
AIML > b.py
105:34 CRLF UTF-8 4 spaces Python 3.12 (AIML)
78°F
Partly cloudy
Search

USER CASE:

Pharmacy Student Reference

A pharmacy student uses the system to quickly retrieve detailed information about a medicine for their studies or during exams. By either scanning the medicine label or typing its name, they can access details such as its uses, side effects, and dosage instructions.

General Public Medicine Inquiry

An individual who is not familiar with a particular medicine can use the system to ask about its uses, side effects, and precautions. They can either speak or type the name of the medicine, and the system will provide the necessary information for better understanding.

RESULT:

The project effectively uses OCR and speech recognition to identify medicines from multiple input sources and provides relevant information. It ensures accurate matching and enhances user interaction through AI-driven features.