GOVERNMENT COLLEGE OF ENGINEERING

TIRUNELVELI- 627007

DEPARTMENT OF

MECHANICAL ENGINEERING

NM1081 – GENERATIVE AI

B.E – FOURTH SEMESTER

MAY 2025

ANNA UNIVERSITY – CHENNAI

REGULATION 2021

GOVERNMENT COLLEGE OF ENGINEERING

TIRUNELVELI - 627 007

2024 - 2025

DEPARTMENT OF MECHANICAL ENGINEERING

REGISTER NO:

CERTIFICATE

This is a bonafide record of work done by _____ for _____ in Government College Of Engineering, Tirunelveli during the year 2024-2025.

Place :

Date :

Staff-in-charge Head of the Department

Submitted for the Anna University practical examination held at Government College Of Engineering, Tirunelveli On…….

Internal Examiner External Examiner

TABLE OF CONTENTS

Ex.No.1 AI-Powered Content Creation for Marketing

Aim

To create openai code for AI-Powered Content Creation for Marketing.

Requirements

Python 3.7+

openai package (pip install openai)

OpenAI API key

Python Code:

```
import openai

# Set your OpenAI API key
openai.api_key = "your-api-key-here"

def generate_marketing_content(prompt, tone="professional", max_tokens=150):
    """
    Generate marketing content using OpenAI's GPT model.

    Parameters:
    - prompt (str): The idea or product to promote.
    - tone (str): Desired tone of the content ("professional", "funny", "persuasive").
    - max_tokens (int): Length of the output.

    Returns:
    - str: Generated marketing content.
    """
```

```
    formatted_prompt = (
        f"Write a {tone} marketing message about the following:\n"
        f"{prompt}\n"
        f"Make it catchy and suitable for use on a website or social media."
    )

    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[
            {"role": "system", "content": "You are a marketing copywriter."},
            {"role": "user", "content": formatted_prompt}
        ],
        max_tokens=max_tokens,
        temperature=0.8,
        n=1
    )

    content = response['choices'][0]['message']['content']
    return content.strip()

# Example usage
if __name__ == "__main__":
    product_idea = "A new AI-powered analytics tool that helps small businesses understand customer behavior"
    generated_copy = generate_marketing_content(product_idea, tone="persuasive")
    print("Generated Marketing Content:\n")
```

```
print(generated_copy)
```

Output Examples

Prompt:

"A skincare brand launching a new anti-aging serum"

Tone:

"Luxury"

Generated Output:

"Experience timeless beauty with our revolutionary anti-aging serum. Infused with clinically proven ingredients and cutting-edge AI-formulated peptides, this serum works with your skin—not against it—to restore youthful radiance. Because age is just a number, and beauty is eternal."

Ex.No.2 Realistic Image Synthesis for e-commerce generative

Aim

To Create realistic image synthesis for e-commerce using Generative AI involves using models like Stable Diffusion, GANs (e.g., StyleGAN), or ControlNet, depending on the type of product and realism required. Below is a Python code example using Stable Diffusion (via Hugging Face diffusers library) to generate e-commerce-style product images.

Prerequisites

Make sure to install these first:

```
pip install diffusers transformers accelerate torch
```

Python Code: Product Image Generator with Prompt

```
from diffusers import StableDiffusionPipeline
import torch
from PIL import Image


# Load pre-trained Stable Diffusion model
pipe = StableDiffusionPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5",  # You can try other checkpoints for more realism
    torch_dtype=torch.float16
).to("cuda")


# Function to generate product image
def generate_product_image(prompt, output_path="product_image.png", guidance_scale=8.5):
    image = pipe(prompt, guidance_scale=guidance_scale).images[0]
    image.save(output_path)
    return image
```

Example eCommerce prompt

prompt = "high-quality photo of a modern leather handbag on a white studio background, e-commerce product shot"

Generate image

image = generate_product_image(prompt)

image.show()

Prompt Engineering Tips for eCommerce

Use structured prompts:

"studio shot of a [product], isolated on white background"

"realistic lighting, high resolution, catalog style"

"360 spin view of [product] with shadows"

Examples

"minimalist running shoes, floating, shadow, isolated, white background"

"smartphone with case, top view, on a white table"

"realistic luxury wristwatch, close-up, e-commerce style"

Optional: Use ControlNet or Inpainting for Customization

For:

Pose control (e.g., clothing on models)

Background removal

Brand overlays

Ex.No.3 **AI-Generated Music Composition**

Aim

To create Python code for AI-generated music composition using generative AI techniques. This example leverages:

Music21: for handling music theory and MIDI output.

TensorFlow/Keras: for training a simple neural network.

NumPy: for data processing.

This code will create a LSTM-based sequence generator for simple melodies.

Prerequisites

Make sure to install the required libraries:

```
pip install music21 tensorflow numpy
```

AI Music Composer (Python Code)

```
import numpy as np
from music21 import instrument, note, stream, chord
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense, Activation
from tensorflow.keras.utils import to_categorical
import random

# Generate dummy note sequences for demonstration
def generate_dummy_notes(n=1000):
```

```
    pitch_names = ['C', 'D', 'E', 'F', 'G', 'A', 'B']
    octaves = [3, 4, 5]
    return [random.choice(pitch_names) + str(random.choice(octaves)) for _ in range(n)]

notes = generate_dummy_notes()

# Map notes to integers
pitchnames = sorted(set(notes))
note_to_int = {note: number for number, note in enumerate(pitchnames)}

# Prepare sequences
sequence_length = 100
network_input = []
network_output = []

for i in range(len(notes) - sequence_length):
    seq_in = notes[i:i + sequence_length]
    seq_out = notes[i + sequence_length]
    network_input.append([note_to_int[char] for char in seq_in])
    network_output.append(note_to_int[seq_out])

n_patterns = len(network_input)
n_vocab = len(pitchnames)

X = np.reshape(network_input, (n_patterns, sequence_length, 1))
X = X / float(n_vocab)
```

```
y = to_categorical(network_output)
# Build the model
model = Sequential([
    LSTM(256, input_shape=(X.shape[1], X.shape[2]), return_sequences=True),
    Dropout(0.3),
    LSTM(256),
    Dense(256),
    Dropout(0.3),
    Dense(n_vocab),
    Activation('softmax')
])
model.compile(loss='categorical_crossentropy', optimizer='adam')

# Train the model (for demo, only 1 epoch)
model.fit(X, y, epochs=1, batch_size=64)

# Generate music
start = np.random.randint(0, len(network_input)-1)
pattern = network_input[start]
output_notes = []

for _ in range(100):
    prediction_input = np.reshape(pattern, (1, len(pattern), 1))
    prediction_input = prediction_input / float(n_vocab)

    prediction = model.predict(prediction_input, verbose=0)
```

```
    index = np.argmax(prediction)
    result = pitchnames[index]
    output_notes.append(note.Note(result))

    pattern.append(index)
    pattern = pattern[1:]
# Create MIDI stream
midi_stream = stream.Stream(output_notes)
midi_stream.write('midi', fp='generated_music.mid')
print("MIDI file saved as 'generated_music.mid'")
```

Output

The model learns from synthetic notes and generates a short melody.

Saves it as a .mid file which you can play in any MIDI player or DAW.

Ex.No.4 **AI-Assisted Video Dubbing and Subtitling**

Aim

Python script using OpenAI tools (like Whisper for transcription and translation) and moviepy for editing video. This code does the following:

- ✓ Transcribes the original video audio using Whisper.
- ✓ Translates the transcript (optional).
- ✓ Adds subtitles to the video.

Optionally generates dubbed audio using a TTS model (e.g., OpenAI's TTS, ElevenLabs, or Coqui).

Requirements

- ✓ moviepy
- ✓ openai (for Whisper and TTS)
- ✓ ffmpeg (installed and available in PATH)

Optionally, gtts or other TTS systems for voice dubbing

Install dependencies:

pip install moviepy openai gtts

Python Script: AI-Assisted Video Dubbing and Subtitling

```
import os

import openai

from moviepy.editor import VideoFileClip, TextClip, CompositeVideoClip

from gtts import gTTS

import tempfile

# Set your OpenAI API Key

openai.api_key = "YOUR_OPENAI_API_KEY"

def transcribe_audio(audio_path):

    """Transcribe audio using Whisper via OpenAI API"""
```

```
    with open(audio_path, "rb") as f:
        transcript = openai.Audio.transcribe("whisper-1", f)
    return transcript["text"]


def generate_tts_audio(text, lang='en', output_path="dubbed_audio.mp3"):
    """Generate TTS audio using gTTS"""
    tts = gTTS(text=text, lang=lang)
    tts.save(output_path)
    return output_path
def create_subtitled_video(video_path, transcript_text,
output_path="output_with_subs.mp4"):
    """Add subtitles to the video"""
    video = VideoFileClip(video_path)
        subtitle = TextClip(transcript_text, fontsize=24, color='white', bg_color='black',
size=video.size)
    subtitle = subtitle.set_duration(video.duration).set_position(("center", "bottom"))


    final = CompositeVideoClip([video, subtitle])
    final.write_videofile(output_path, codec='libx264', audio_codec='aac')
    return output_path
def extract_audio_from_video(video_path, audio_path="temp_audio.wav"):
    """Extract audio from video file using ffmpeg"""
    os.system(f"ffmpeg -y -i \"{video_path}\" -q:a 0 -map a \"{audio_path}\"")
    return audio_path
def replace_audio_in_video(video_path, new_audio_path,
output_path="dubbed_video.mp4"):
    """Replace the audio in video with new dubbed audio"""
```

```
    os.system(f"ffmpeg -y -i \"{video_path}\" -i \"{new_audio_path}\" -c:v copy -map 0:v:0 -map 1:a:0 -shortest \"{output_path}\"")
    return output_path


# Main pipeline
def ai_dub_and_subtitle(video_path, lang='en'):
    with tempfile.TemporaryDirectory() as tmp:
        audio_path = os.path.join(tmp, "audio.wav")
        extract_audio_from_video(video_path, audio_path)


        transcript = transcribe_audio(audio_path)
        print(f"Transcript: {transcript}")
        # Optional: Translate transcript (e.g., using openai.ChatCompletion or DeepL API)
        subtitled_video = create_subtitled_video(video_path, transcript, os.path.join(tmp, "with_subs.mp4"))
        # Generate new audio
        tts_audio = generate_tts_audio(transcript, lang=lang, output_path=os.path.join(tmp, "dubbed.mp3"))
        # Replace audio with dubbed
        final_video = replace_audio_in_video(subtitled_video, tts_audio, output_path="final_dubbed_video.mp4")
        print(f"Final dubbed and subtitled video saved to: {final_video}")
# Example usage
ai_dub_and_subtitle("example_video.mp4", lang='en')
```

Can Improve or Customize

Translation: Use OpenAI's GPT-4 or DeepL API for multilingual translation.

Advanced TTS: Replace gTTS with more natural TTS (OpenAI’s new TTS or ElevenLabs).

SRT File Export: Save transcript as an .srt file for better control.

Speaker Diarization: Use tools like PyAnnote or AssemblyAI for multi-speaker transcripts.

Ex.No.5 **Ethical Chatbot Design for Healthcare**

Aim

Designing an ethical chatbot for healthcare using OpenAI's tools involves ensuring the system is:

- ✓ Privacy-aware (compliant with HIPAA or relevant laws)
- ✓ Transparent (discloses it is an AI, not a medical professional)
- ✓ Helpful, not harmful (avoids diagnosis, offers general support)
- ✓ Fallback-aware (refers to a human or emergency help if needed)

Ethical Healthcare (Python Code)

```
import openai


# Initialize OpenAI API key (set securely in your environment)

openai.api_key = "your-api-key"


# Emergency or red flag keywords

EMERGENCY_KEYWORDS = ["suicidal", "overdose", "can't breathe", "heart attack", "bleeding heavily"]

# Basic system prompt with ethical guardrails

SYSTEM_PROMPT = """

You are a helpful, ethical, and privacy-conscious virtual health assistant.

You do not diagnose, prescribe, or offer emergency medical advice.

Always recommend speaking to a licensed healthcare provider for medical issues.

If the user mentions an emergency, advise them to call emergency services immediately.

"""

def check_emergency(user_input):

    """Check if the message may indicate an emergency."""

    for keyword in EMERGENCY_KEYWORDS:
```

```
        if keyword.lower() in user_input.lower():
            return True
    return False


def chat_with_bot(user_input):
    """Chat function that interacts with the OpenAI API."""
        if check_emergency(user_input):
        return (
            "⚠ It sounds like you might be experiencing a medical emergency. "
            "Please call your local emergency services or go to the nearest hospital immediately."
        )
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[
            {"role": "system", "content": SYSTEM_PROMPT},
            {"role": "user", "content": user_input}
        ],
        temperature=0.5
    )
    return response['choices'][0]['message']['content'].strip()
# Example usage
if __name__ == "__main__":
    print("Welcome to HealthBot (AI). I can provide general wellness info.")
    while True:
        user_input = input("\nYou: ")
```

```
    if user_input.lower() in ["exit", "quit"]:
        print("HealthBot: Stay well! Always talk to a professional for medical help.")
        break
    reply = chat_with_bot(user_input)
    print(f"HealthBot: {reply}")
```

Key Ethical Safeguards:

- ✓ Emergency detection with a keyword list.
- ✓ System prompt guardrails to prevent diagnosis/prescription.
- ✓ Temperature control (0.5) for balanced, safe outputs.
- ✓ Clear identity disclosure (“I am an AI assistant, not a doctor”)

Deployment :

- ✓ Deploy on HTTPS-secured environments.
- ✓ Log conversations responsibly (anonymize or don't store if unnecessary).
- ✓ Pair with a user agreement / disclaimer on the UI.

Ex.No.6 Generative AI for Story Writing and Plot Generation

Aim

Python example using OpenAI's GPT API to build a generative AI tool for story writing and plot generation.

Features

- ✓ Generate story ideas or plots.
- ✓ Generate full story content based on a prompt or idea.
- ✓ Adjustable creativity using temperature.

Prerequisites

Install OpenAI's library:

```
pip install openai
```

Python Code

```
import openai

# Set your OpenAI API key

openai.api_key = "your-api-key"

def generate_plot(genre="fantasy", style="dramatic", length="short"):
    prompt = (
        f"Generate a {length} {genre} story plot with a {style} tone. "
        "Include a beginning, middle, and twist ending."
    )
        response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[
            {"role": "system", "content": "You are a creative story writer AI."},
            {"role": "user", "content": prompt}
        ],
```

```
        temperature=0.8,
        max_tokens=300
    )
        return response['choices'][0]['message']['content']
def write_story(plot_description, length="medium"):
    prompt = (
        f"Write a {length} story based on this plot:\n\n"
        f"{plot_description}\n\n"
        "Make it immersive with character development, dialogue, and vivid description."
    )
        response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[
            {"role": "system", "content": "You are a master fiction author."},
            {"role": "user", "content": prompt}
        ],
        temperature=0.9,
        max_tokens=1000
    )
        return response['choices'][0]['message']['content']
# Example usage
if __name__ == "__main__":
    plot = generate_plot(genre="sci-fi", style="mysterious", length="short")
    print("Generated Plot:\n", plot)
    story = write_story(plot_description=plot, length="long")
    print("\nGenerated Story:\n", story)
```

Options for Tweak

- ✓ temperature: 0.7–1.0 for creative writing.
- ✓ max_tokens: Increase for longer stories.
- ✓ genre, style, and length: Customize input for genre-specific output

Ex.No.7 **Personalized Product Recommendations**

Aim

To create a personalized product recommendation system using generative AI. This version uses OpenAI's GPT API (you can simulate locally if API access is not needed) to generate tailored product recommendations based on user profiles.

Requirements

Install required packages

```
pip install openai pandas
```

Code: Generative Personalized Product Recommender

```
import openai
import pandas as pd
# Replace with your own OpenAI API key
openai.api_key = "your_openai_api_key"
# Sample user profile
user_profile = {
    "name": "Alice",
    "age": 30,
    "interests": ["fitness", "technology", "sustainable living"],
    "past_purchases": ["yoga mat", "smartwatch", "reusable water bottle"]
}
# Sample product catalog
product_catalog = pd.DataFrame([
    {"product": "Eco-Friendly Yoga Pants", "category": "fitness"},
    {"product": "Wireless Earbuds", "category": "technology"},
    {"product": "Smart Thermostat", "category": "technology"},
    {"product": "Organic Protein Powder", "category": "fitness"},
```

```
    {"product": "Solar-Powered Charger", "category": "sustainable living"},
    {"product": "Stainless Steel Lunchbox", "category": "sustainable living"},
])
def generate_prompt(user_profile, product_catalog):
    catalog_str = "\n".join(f"- {row['product']} ({row['category']})"
                            for _, row in product_catalog.iterrows())

    prompt = f"""
You are a personalized product recommendation assistant.
User Profile:
- Name: {user_profile['name']}
- Age: {user_profile['age']}
- Interests: {', '.join(user_profile['interests'])}
- Past Purchases: {', '.join(user_profile['past_purchases'])}
Product Catalog:
{catalog_str}
Based on the user profile and product catalog, recommend the top 3 most suitable
products. Just list the product names.
"""
    return prompt.strip()
def get_recommendations(prompt):
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[{"role": "user", "content": prompt}],
        temperature=0.7,
    )
```

```
    return response.choices[0].message["content"].strip()
# Generate prompt and get recommendations
prompt = generate_prompt(user_profile, product_catalog)
recommendations = get_recommendations(prompt)
print("Recommended Products:")
print(recommendations)
```

Output Example:

Recommended Products:

- ✓ Eco-Friendly Yoga Pants
- ✓ Solar-Powered Charger
- ✓ Organic Protein Powder

Ex.No.8 Creating Synthetic Training Data for Computer Vision Tasks

---

Aim

To Create synthetic training data for computer vision tasks using generative AI involves using techniques like:

- ✓ Image augmentation (classical and AI-based),
- ✓ Procedural generation (e.g., rendering scenes),
- ✓ Generative models like GANs or diffusion models.

Requirements

Install required packages:

```
pip install diffusers transformers accelerate torch
```

Python Code – Generate Synthetic Images with Text Prompts

```
from diffusers import StableDiffusionPipeline
import torch
import os
from PIL import Image
# Load Stable Diffusion model
pipe = StableDiffusionPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5",
    torch_dtype=torch.float16,
    use_auth_token=True
).to("cuda")
# Output folder
output_dir = "synthetic_data"
os.makedirs(output_dir, exist_ok=True)
# Define prompts
prompts = [
```

```
    "A red sports car on a sunny road",
    "A cat sitting on a wooden table",
    "A person riding a bike in the mountains",
    "An industrial robot in a factory",
]
# Generate images
for i, prompt in enumerate(prompts):
    image = pipe(prompt, num_inference_steps=30, guidance_scale=7.5).images[0]
    filename = os.path.join(output_dir, f"synthetic_{i:03d}.png")
    image.save(filename)
    print(f"Saved: {filename}")
```

Optional Enhancements

Generate Multiple Variations per Prompt:

Use different seeds or apply augmentations with imgaug or Albumentations.

Add Annotations:

Use models like Grounding DINO + Segment Anything to generate bounding boxes or masks automatically.

Train-Test Split:

Create synthetic datasets with class-wise folders for classification or COCO/YOLO formats for detection.

Ex.No.9 **Speech Synthesis for Accessibility**

Aim

OpenAI and other libraries to implement speech synthesis for accessibility. This will take user input or text (e.g., from a webpage or app), summarize it or rephrase it using OpenAI, and then convert it to speech using a text-to-speech engine like pyttsx3 or gTTS.

Requirements

Install the necessary libraries first:

pip install openai pyttsx3

Alternatively, for gTTS (Google Text-to-Speech, which can save audio to a file):

pip install openai gTTS playsound

Python Code: Speech Synthesis for Accessibility

Using pyttsx3 for local speech synthesis:

```
import openai
import pyttsx3


# Set your OpenAI API key
openai.api_key = "your-openai-api-key"
# Function to get simplified or summarized version of the input text
def get_accessible_text(input_text):
   response = openai.ChatCompletion.create(
      model="gpt-4",
      messages=[
         {"role": "system", "content": "You simplify text for better accessibility."},
         {"role": "user", "content": f"Simplify or summarize this: {input_text}"}
      ]
   )
```

```
    simplified_text = response['choices'][0]['message']['content']
    return simplified_text.strip()
# Function to convert text to speech
def speak_text(text):
    engine = pyttsx3.init()
    engine.setProperty('rate', 150)  # Slower speech rate
    engine.say(text)
    engine.runAndWait()
# Main function
def main():
    input_text = input("Enter text to simplify and read aloud:\n")
    simplified_text = get_accessible_text(input_text)
    print("\nSimplified Text:\n", simplified_text)
    speak_text(simplified_text)
if __name__ == "__main__":
    main()
Alternate: Save to MP3 with gTTS
from gtts import gTTS
import openai
import os
from playsound import playsound

openai.api_key = "your-openai-api-key"
def get_accessible_text(input_text):
    response = openai.ChatCompletion.create(
        model="gpt-4",
```

```
        messages=[
            {"role": "system", "content": "You simplify text for better accessibility."},
            {"role": "user", "content": f"Simplify or summarize this: {input_text}"}
        ]
    )
    return response['choices'][0]['message']['content'].strip()
def speak_text_gtts(text, filename="output.mp3"):
    tts = gTTS(text)
    tts.save(filename)
    playsound(filename)
    os.remove(filename)
def main():
    input_text = input("Enter text to simplify and read aloud:\n")
    simplified_text = get_accessible_text(input_text)
    print("\nSimplified Text:\n", simplified_text)
    speak_text_gtts(simplified_text)
if __name__ == "__main__":
    main()
```

Ex.No.10 Predictive Text Generation for Custom Applications

Aim

To create a predictive text generation system using OpenAI's API in Python, you can use the openai Python library. Below is a sample script that demonstrates how to set up predictive text generation for a custom application, such as autocompletion in a chat app or email draft helper.

Prerequisites

Install the OpenAI library:

pip install openai

Set up your OpenAI API key securely in your environment:

export OPENAI_API_KEY="your-api-key-here"

Python Code

```
import openai
import os
# Load your OpenAI API key from environment variable
openai.api_key = os.getenv("OPENAI_API_KEY")


def generate_predictive_text(prompt, max_tokens=50, temperature=0.7, model="gpt-4"):
    """
    Generate predictive text from a given prompt.
        Args:
        prompt (str): The input text to complete.
        max_tokens (int): Max number of tokens to predict.
        temperature (float): Sampling temperature for creativity.
        model (str): OpenAI model to use (e.g., "gpt-3.5-turbo", "gpt-4").
```

```
    Returns:
        str: Generated continuation text.
    """
    response = openai.ChatCompletion.create(
        model=model,
        messages=[
            {"role": "system", "content": "You are a helpful text autocompletion assistant."},
            {"role": "user", "content": prompt}
        ],
        max_tokens=max_tokens,
        temperature=temperature
    )
    return response['choices'][0]['message']['content'].strip()
# Example usage
if __name__ == "__main__":
    user_input = "Once upon a time in a distant galaxy,"
    prediction = generate_predictive_text(user_input)
    print("Predicted continuation:\n", prediction)
```