Ex.No: **Date:**

Exploration of Prompting Techniques for AI-Powered Content Creation

Aim:

The aim of this experiment is to investigate how different prompting techniques influence AI-generated content across various formats (text, images, audio, and video). By leveraging models like **GPT-4, DALL·E, RunwayML, and ElevenLabs**, we will analyze how variations in prompts affect output quality, creativity, and relevance. This study will help optimize AI-assisted content creation for marketing, education, and entertainment.

Procedure:

1. Define Content Goals

- Identify use cases:
  - **Text:** Blog posts, social media captions, technical writing.
  - **Images:** Logos, illustrations, concept art.
  - **Audio:** Voiceovers, podcasts, sound effects.
  - **Video:** Short clips, animations, ads.

2. Experiment with Prompting Techniques

Test different prompt structures:

- **Descriptive Prompts:**
  - *"A high-tech lab with glowing blue vials, sci-fi style."* (for DALL·E)
- **Instruction-Based Prompts:**
  - *"Write a 300-word blog intro about sustainable fashion."* (for GPT-4)
- **Emotion-Driven Prompts:**
  - *"A melancholic piano track for a rainy day scene."* (for AudioGen)
- **Iterative Refinement:**
  - Start with a basic prompt, then add details like style, tone, or length.

3. Develop a Multi-Model Content Generator

- Use Python to interact with multiple AI APIs (OpenAI, ElevenLabs, RunwayML).
- Compare outputs from different models for the same prompt.

4. Evaluate and Optimize

- Assess output quality, coherence, and adherence to prompts.
- Adjust prompts iteratively for better results.

5. Deploy (Optional)

- Build a **Streamlit app** for users to generate content interactively.

Program (Python Code for Multi-Format Content Generation):

Python Code

```python
import openai

from elevenlabs import generate, set_api_key

import requests

import logging

from dotenv import load_dotenv


# Load environment variables (API keys)

load_dotenv()


# Configure logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')

logger = logging.getLogger(__name__)


# API Keys (Replace in .env file)

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

ELEVENLABS_API_KEY = os.getenv("ELEVENLABS_API_KEY")

RUNWAYML_API_KEY = os.getenv("RUNWAYML_API_KEY")


# Initialize APIs

openai.api_key = OPENAI_API_KEY

set_api_key(ELEVENLABS_API_KEY)
```

```
def generate_text(prompt, model="gpt-4"):
    """Generate text using OpenAI's GPT-4."""
    try:
        response = openai.ChatCompletion.create(
            model=model,
            messages=[{"role": "user", "content": prompt}],
            max_tokens=500
        )
        return response.choices[0].message['content'].strip()
    except Exception as e:
        logger.error(f"GPT-4 Error: {e}")
        return "Failed to generate text."

def generate_image(prompt, model="dall-e-3"):
    """Generate an image using DALL·E."""
    try:
        response = openai.Image.create(
            model=model,
            prompt=prompt,
            n=1,
            size="1024x1024"
        )
        return response.data[0].url
    except Exception as e:
        logger.error(f"DALL·E Error: {e}")
        return "Failed to generate image."

def generate_audio(prompt, voice="Bella"):
    """Generate speech using ElevenLabs."""
    try:
        audio = generate(
            text=prompt,
            voice=voice,
            model="eleven_monolingual_v2"
        )
        with open("output_audio.mp3", "wb") as f:
            f.write(audio)
        return "Audio generated successfully."
    except Exception as e:
        logger.error(f"ElevenLabs Error: {e}")
        return "Failed to generate audio."

def generate_video(prompt, model="runwayml/gen-2"):
    """Generate video using RunwayML."""
    try:
        headers = {"Authorization": f"Bearer {RUNWAYML_API_KEY}"}
        payload = {"prompt": prompt, "model": model}
        response = requests.post(
            "https://api.runwayml.com/v1/video/generate",
            headers=headers,
            json=payload
        )
        if response.status_code == 200:
            return response.json().get("output_url")
        else:
            logger.error(f"RunwayML Error: {response.text}")
            return "Failed to generate video."
    except Exception as e:
        logger.error(f"API Error: {e}")
        return "Video generation failed."
```

```
def main():
    print("=== AI Content Generation Explorer ===")
    print("Generate text, images, audio, or videos using AI.")
    print("Type 'quit' to exit.\n")

    while True:
        content_type = input("Choose type (text/image/audio/video): ").strip().lower()

        if content_type == "quit":
            print("Exiting...")
            break

        prompt = input("Enter your prompt: ").strip()

        if not prompt:
            print("Please enter a valid prompt.")
            continue

        if content_type == "text":
            print("\nGenerating text...")
            result = generate_text(prompt)
            print("\nGenerated Text:\n", result)

        elif content_type == "image":
            print("\nGenerating image...")
            result = generate_image(prompt)
            print("\nImage URL:", result)

        elif content_type == "audio":
            print("\nGenerating audio...")
            result = generate_audio(prompt)
            print("\nResult:", result)

        elif content_type == "video":
            print("\nGenerating video... (This may take a few minutes)")
            result = generate_video(prompt)
            print("\nVideo URL:", result)

        else:
            print("Invalid content type. Try again.")

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

Output Examples:

1. Text Generation (GPT-4)

Prompt:
"Write a catchy Instagram caption for a coffee brand."
Output:
"Brewed to perfection. Sipped with love. #CoffeeTime"

2. Image Generation (DALL·E 3)

Prompt:
"A minimalist logo for a yoga studio, green and white tones."
Output:

(Simulated URL)

3. Audio Generation (ElevenLabs)

Prompt:
"A warm, friendly voice says: 'Welcome to our podcast on mindfulness.'"
Output:
[Downloadable MP3 file with natural-sounding voice]

4. Video Generation (RunwayML)

Prompt:
"A tranquil forest scene with sunlight filtering through trees."
Output:
[Short video clip of a serene forest]

Result:

Successful content generation across multiple formats.
Key Findings:

- **Detailed prompts** yield higher-quality outputs.
- **Style modifiers** (e.g., *"minimalist," "cinematic"*) significantly alter results.
- **Iterative refinement** improves output relevance.