Vision Novel

Project Introduction

Vision Novel is an **AI-powered visual novel generation tool** designed to help creators **visualize their novel content**, enabling multiple expressive forms for distribution. This project integrates **text processing, knowledge vectorization, script generation, image creation, speech synthesis, and video synthesis**, allowing novels to be transformed into **dynamic visual novel videos**, enhancing their reach and viewer experience.

Vision Novel: Unlocking the New Era of AI-Powered Visual Novels!

---

1. Core Technologies

🔷 LangChain

1. **LangChain** serves as the **core AI orchestration framework**.
2. It **splits long text** while maintaining semantic integrity, making it compatible with different AI models' input length.
3. By integrating **GraphRAG**, it vectorizes text to enable **long-text memory**, improving retrieval efficiency. This ensures AI-generated scripts are **more accurate and aligned with the main story** while minimizing **hallucinations**.
4. **Using LangChain's Prompt + Memory**, it structures **intelligent dialogue chains**, allowing GPT to generate **scripts in line with the visual novel style**.

🔷 OpenAI

1. **Automatically splits novel content** into **scenes** and **events**.
2. **Generates character dialogues**, making characters more vivid.
3. **Creates AI-generated illustration prompts** for image generation.
4. **Formats scripts** (Scene → Dialogues) for easier visualization.
5. **Adjusts tone and emotion** to make dialogues more natural.

🔷 DALL·E 3 + Stable Diffusion

1. Enables **AI to generate illustrations that match the novel's atmosphere**.
2. **Automatically generates character portraits** based on novel descriptions.

3. **Ensures a consistent artistic style** across all images by **using JSON-based user configurations and GPT-controlled visual descriptions**.
4. **Reduces the need for manual illustration**, accelerating visual novel production.
5. **Creates key scene illustrations** (e.g., battles, plot twists) to enhance storytelling.

🔷 TTS (Text-to-Speech)

1. **Provides unique AI voices for each character**, enhancing personalization.
2. **Adapts voice styles** based on character traits (e.g., deep, passionate, soft, low-pitched) for a more immersive experience.
3. **Brings character dialogues to life**, making them more dramatic.
4. **Allows novels to "speak,"** offering creators a richer storytelling approach.

🔷 FFmpeg + Stable Video Diffusion

1. **Combines TTS-generated speech with AI-generated images into video sequences**.
2. **Synchronizes audio and video**, ensuring dialogues match the visuals.
3. Supports **subtitles, visual effects, and background music** integration.
4. **Uses AI to create short videos**, turning static novel illustrations into dynamic animations.
5. Can process **a single image and generate an animated video**.

---

2. Project Goals

🔷 **Lower the creative barrier**: Helps **independent authors** and **small studios** quickly convert novels into visualized works.
🔷 **Enhance distribution and engagement**: Uses a **multi-modal format (visuals + audio + video)** to make novels more appealing, suitable for **short videos, animations, and game cutscenes**.
🔷 **Intelligent workflow**: Automates **text analysis, image generation, voice synthesis, and video production**, significantly increasing efficiency.

---

3. Project Features

- 📖 **Knowledge Vectorization**
  - Extracts key information from novels and **constructs a knowledge graph** to support further processing.
- ✂️ **Text Splitting**

    - **Automatically segments long-form novels** into **chapters, scenes, and dialogues**, optimizing later processes.
- 🎭 **Script Generation**
    - Uses **Natural Language Processing (NLP)** to **convert novel text into script format**, including **character dialogues, scene descriptions, and narrations**.
- 🎨 **Visual Element Generation**
    - Uses **AI-powered image models** to generate **illustrations that match the scene's atmosphere**, ensuring a **consistent artistic style**.
- 🔊 **Speech Synthesis (TTS)**
    - **Generates high-quality voiceovers for character dialogues and narrations**, enhancing immersion.
- 🎬 **Video Synthesis**
    - **Combines text, images, and audio** to **automatically generate visual novel videos**, presenting stories dynamically.

---

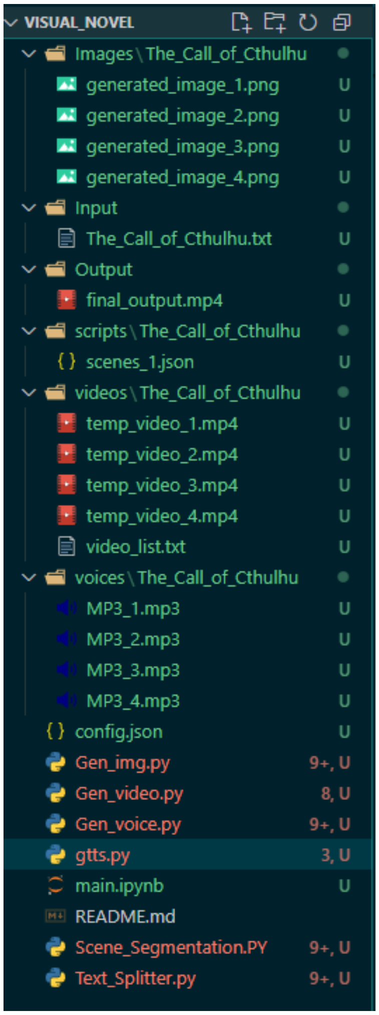
4. Applicable Scenarios

- 📖 **Web Novel Adaptation** → Transform novels into visual formats.
- 🎮 **Game Story Demonstration** → Provide **cutscene animations** for independent games.
- 🎬 **Animated Short Film Production** → Combine AI-generated scripts and illustrations to quickly create visualized stories.
- 📺 **New Media Content Creation** → Suitable for platforms such as **Bilibili, YouTube, and TikTok**, enhancing content reach and engagement.

🎯 Future Development

🔷 **Multi-Style AI Generation** → Adapt to **various artistic styles** for visual novels.
🔷 **Interactive VN (Visual Novel Game)** → Allow users to **interactively experience the story**.
🔷 **Enhanced AI Voice Acting** → Improve **TTS emotional expression**, creating more immersive storytelling.
🔷 **Smart Editing & Video Optimization** → Make videos **better aligned with the storytelling style** of visual novels.

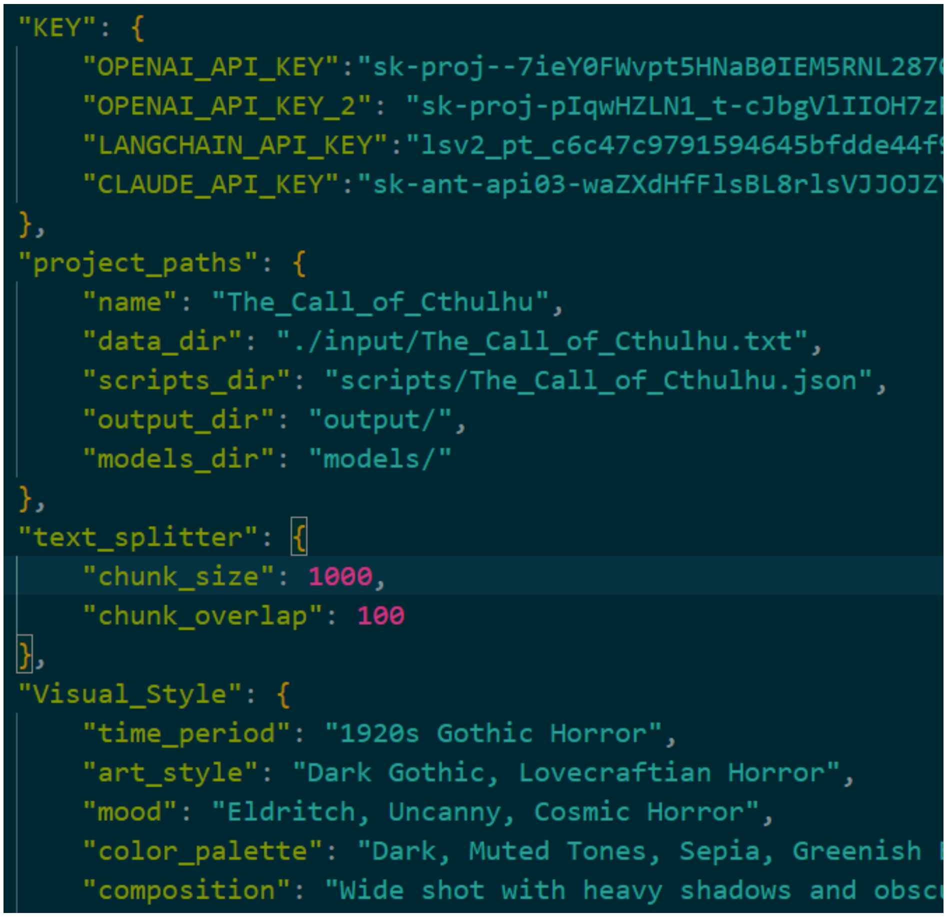
Project Core Code Explanation and Demonstration of Results:



```
VISUAL_NOVEL
  Images \ The_Call_of_Cthulhu
    generated_image_1.png          U
    generated_image_2.png          U
    generated_image_3.png          U
    generated_image_4.png          U
  Input
    The_Call_of_Cthulhu.txt        U
  Output
    final_output.mp4               U
  scripts \ The_Call_of_Cthulhu
    { } scenes_1.json              U
  videos \ The_Call_of_Cthulhu
    temp_video_1.mp4               U
    temp_video_2.mp4               U
    temp_video_3.mp4               U
    temp_video_4.mp4               U
    video_list.txt                 U
  voices \ The_Call_of_Cthulhu
    MP3_1.mp3                      U
    MP3_2.mp3                      U
    MP3_3.mp3                      U
    MP3_4.mp3                      U
  { } config.json                  U
  Gen_img.py                   9+, U
  Gen_video.py                  8, U
  Gen_voice.py                 9+, U
  gtts.py                       3, U
  main.ipynb                       U
  README.md
  Scene_Segmentation.PY        9+, U
  Text_Splitter.py             9+, U
```

- `Gen_img.py` – Generates AI-illustrated images for each scene in the story.
- `Gen_video.py` – Creates short video clips for individual scenes and then merges them into a complete video.
- `Gen_voice.py` – Generates dialogues and voiceovers for each scene using text-to-speech technology.
- `gtts.py` – Provides an alternative method for generating speech using Google Text-to-Speech (`gTTS`). This is useful in cases where some OpenAI library versions (e.g., the latest ones) do not support built-in audio generation.
- `Scene_Segmentation.py` – Processes the novel text, segments it into structured scenes, and formats the output as a JSON-based script.
- `Text_Splitter.py` – Splits long text while preserving semantic integrity, ensuring compatibility with different AI models that have input length constraints.
- `RGA.ipynb` – Utilizes **GraphRAG** to analyze and structure long-form text, establishing scene relationships and **enabling memory retention for the AI model**.
- `main.ipynb` – Serves as the **central project workflow**. While each `.py` file implements a specific function, this Jupyter Notebook integrates them, allowing seamless execution of the entire project pipeline. Due to file size limitations, only a **single excerpt** from the novel was used to demonstrate the project workflow.

1. config.json:

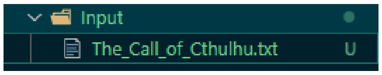


```
"KEY": {
    "OPENAI_API_KEY":"sk-proj--7ieY0FWvpt5HNaB0IEM5RNL287
    "OPENAI_API_KEY_2": "sk-proj-pIqwHZLN1_t-cJbgVlIIOH7z
    "LANGCHAIN_API_KEY":"lsv2_pt_c6c47c9791594645bfdde44f
    "CLAUDE_API_KEY":"sk-ant-api03-waZXdHfFlsBL8rlsVJJOJZ
},
"project_paths": {
    "name": "The_Call_of_Cthulhu",
    "data_dir": "./input/The_Call_of_Cthulhu.txt",
    "scripts_dir": "scripts/The_Call_of_Cthulhu.json",
    "output_dir": "output/",
    "models_dir": "models/"
},
"text_splitter": {
    "chunk_size": 1000,
    "chunk_overlap": 100
},
"Visual_Style": {
    "time_period": "1920s Gothic Horror",
    "art_style": "Dark Gothic, Lovecraftian Horror",
    "mood": "Eldritch, Uncanny, Cosmic Horror",
    "color_palette": "Dark, Muted Tones, Sepia, Greenish
    "composition": "Wide shot with heavy shadows and obsc
```

As shown in the image, the configuration file primarily defines **API keys, file paths, text processing parameters, and visual style** along with other key information.

2. Text Splitter

This project uses the short story *The Call of Cthulhu* as a demonstration:



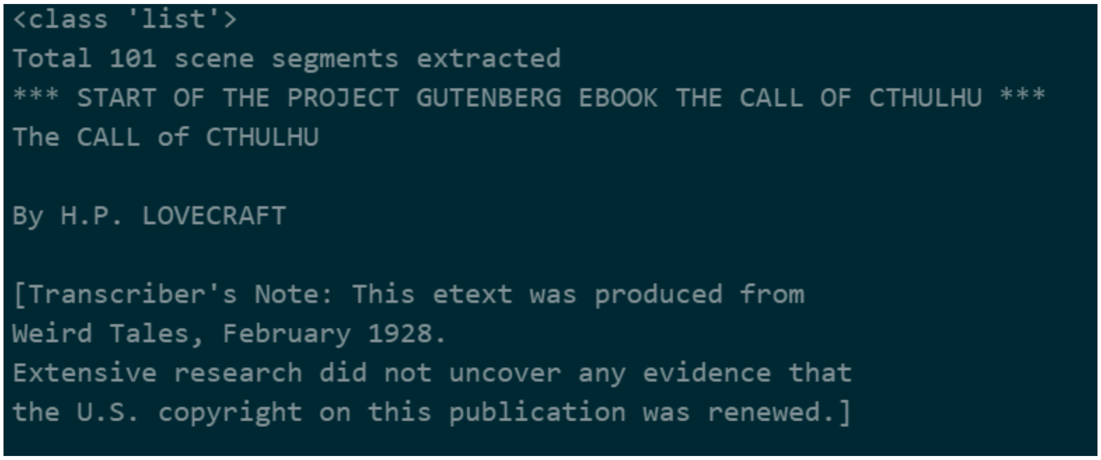
This is the file's location path.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

def load_and_split_text(file_path, chunk_size=2000, chunk_overlap=200):
    # Read the novel text file
    with open(file_path, "r", encoding="utf-8") as f:

        text = f.read()
    # Use recursive text splitting to maintain coherence
    splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size,
chunk_overlap=chunk_overlap)

    chunks = splitter.split_text(text)

    return chunks
```

- **Function Purpose**
  - The function **loads a text file** (e.g., a novel) and **splits it into smaller, structured chunks**.
  - This is useful for **processing large texts with AI models**, which have **token length limits**.
- **Parameters**:
  - `file_path` → Path to the text file.
  - `chunk_size` → The maximum number of characters in each chunk (**default: 2000**).
  - `chunk_overlap` → The number of overlapping characters between chunks to **preserve context** (**default: 200**).
- **Steps in the Code**:
  - 📌 **Reads the text file** (`text = f.read()`)
  - 📌 **Initializes** `RecursiveCharacterTextSplitter`, which intelligently **splits the text while keeping the meaning intact**.
  - 📌 **Returns a list of text chunks** for further processing.

Verify the output:

```
# ✅ Verify the output

print(type(novel_chunks))  # Check the data type of the output

print(f"Total {len(novel_chunks)} scene segments extracted")  # Display the number of text chunks

print(novel_chunks[0])  # Preview the first chunk
```



```
<class 'list'>
Total 101 scene segments extracted
*** START OF THE PROJECT GUTENBERG EBOOK THE CALL OF CTHULHU ***
The CALL of CTHULHU

By H.P. LOVECRAFT

[Transcriber's Note: This etext was produced from
Weird Tales, February 1928.
Extensive research did not uncover any evidence that
the U.S. copyright on this publication was renewed.]
```

3. Designing PromptTemplate for scene segmentation.

```
from langchain.prompts import PromptTemplate
prompt_template = PromptTemplate(

    input_variables=["text"],

    template="""
You are a professional scriptwriter. Analyze the following novel text and divide it into
multiple scenes.
Each scene should include:
- **Scene ID**
- **Scene Summary** (a brief description of what happens)
- **Main Characters**
- **Main Location**
- **Key Events**
- **Scene Transition Reason** (Why is this a new scene?)
- **Original Text** (The original text corresponding to this scene)

**Always return a strict JSON format** with **no extra text or explanations**, only pure
JSON.
Return the output in **JSON format array**, following this example:
"scenes":[
    {{
        "scene_id": 1,
        "summary": "The protagonist finds a mysterious letter at home.",
        "characters": ["Protagonist"],
        "location": "Protagonist's house",
        "events": ["Finds the letter", "Reads the content"],
        "atmosphere": "Mysterious",
        "transition_reason": "A new event begins",
        "original_text": "He entered his home, only to find a dusty envelope on the
table."
    }},
    {{
        "scene_id": 2,
        "summary": "The protagonist visits the mysterious location.",
        "characters": ["Protagonist", "Antagonist"],
        "location": "Mysterious forest",
        "events": ["Meets the antagonist", "Fights the antagonist"],
        "atmosphere": "Tense",
        "transition_reason": "The protagonist arrives at the location",
        "original_text": "He entered the forest, where he met the antagonist."
    }}
    ...
]
Here is the novel text:
{text}
"""
)
```

4. splits novel content **into** scenes **and** events**

```
import openai
import json

class Chatbot:
    def __init__(self, system_prompt):
        self.system_prompt = system_prompt

    def generate_response(self, prompt):
        response = openai.ChatCompletion.create(
            model="gpt-4-turbo",  # Uses the GPT-4 Turbo model for optimized performance
            messages=[
                {"role": "system", "content": self.system_prompt},  # Defines system-level behavior
                {"role": "user", "content": prompt}  # User input prompt
            ],
            temperature=0.5,  # Lowers randomness to ensure structured and stable responses
            top_p=0.9,  # Prevents extreme or highly unlikely outputs
            n=1,  # Generates only one response
            response_format={"type": "json_object"},  # Forces GPT to return a JSON object
            presence_penalty=0.2,  # Slightly encourages new content in responses
            frequency_penalty=0.2,  # Slightly reduces repetitive phrases
            stop=["\n\n"]  # Stops response at the end of a paragraph
        )
        try:
            # Convert the output from JSON string format to a Python dictionary
            # return json.loads(response.choices[0].message["content"])
            return response.choices[0].message["content"]
        except json.JSONDecodeError as e:
            print(f"JSON ERROR: {e}")  # Prints error message if JSON decoding fails
            return None
```

1️⃣ Class Initialization (`__init__`)

- The `Chatbot` class is initialized with a **system prompt**.
- The **system prompt** sets **rules and constraints** for the chatbot (e.g., defining the chatbot's personality, tone, or knowledge limitations).

2️⃣ `generate_response()` Function

- Accepts a **user input (** `prompt` **)** and generates a response using OpenAI's `gpt-4-turbo`.
- **Key model parameters:**

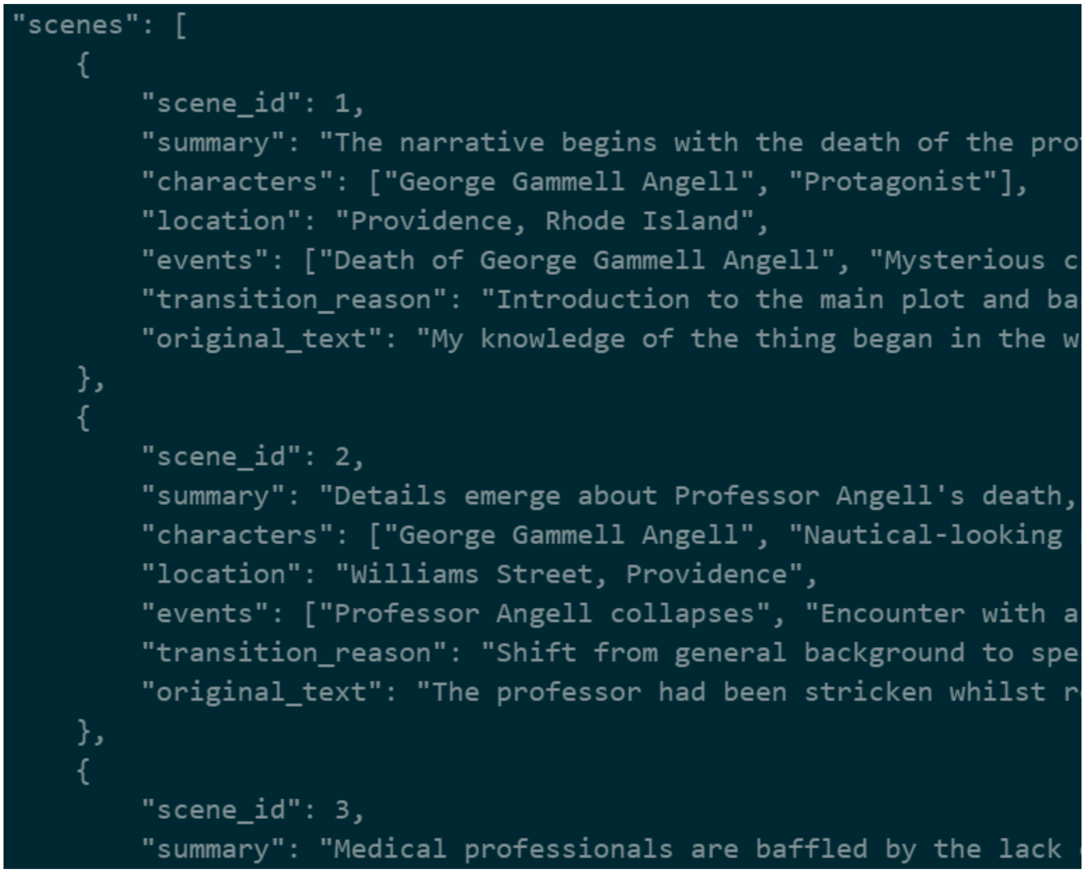
- `temperature=0.5` → Controls randomness (**lower = more deterministic responses**).
- `top_p=0.9` → Prevents **unlikely outputs** (reduces extreme responses).
- `n=1` → Generates **only one response**.
- `response_format={"type": "json_object"}` → Ensures **JSON output** for structured data processing.
- `presence_penalty=0.2` → Slightly encourages **more diverse responses**.
- `frequency_penalty=0.2` → Prevents **repetitive responses**.

3️⃣ Handling the Response

- Tries to extract the chatbot’s response from **GPT's API output**.
- **Uses a** `try-except` **block** to handle potential JSON decoding errors.

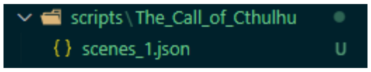
```
# ✅ Generate the prompt required for GPT
prompt = prompt_template.format(text=novel_text)

# ✅ Get the JSON data generated by GPT after processing the prompt
scene_data = chatbot.generate_response(prompt)

# ✅ Output the parsed result
print(scene_data)
```

Then We get:



```
"scenes": [
    {
        "scene_id": 1,
        "summary": "The narrative begins with the death of the pro
        "characters": ["George Gammell Angell", "Protagonist"],
        "location": "Providence, Rhode Island",
        "events": ["Death of George Gammell Angell", "Mysterious c
        "transition_reason": "Introduction to the main plot and ba
        "original_text": "My knowledge of the thing began in the w
    },
    {
        "scene_id": 2,
        "summary": "Details emerge about Professor Angell's death,
        "characters": ["George Gammell Angell", "Nautical-looking
        "location": "Williams Street, Providence",
        "events": ["Professor Angell collapses", "Encounter with a
        "transition_reason": "Shift from general background to spe
        "original_text": "The professor had been stricken whilst r
    },
    {
        "scene_id": 3,
        "summary": "Medical professionals are baffled by the lack
```

Finally, save the response as a JSON file to facilitate further creation, such as image generation and voice synthesis.



5. Designing PromptTemplate for Images.

```
from langchain.prompts import PromptTemplate
import json

# ✅ Load the Visual_Style configuration file

with open("config.json", "r", encoding="utf-8") as f:
    config = json.load(f)
visual_style = config["Visual_Style"]  # Retrieve the Lovecraftian horror style settings
```

```
# ✅ Load the scenes_1.json file
with open("scripts/The_Call_of_Cthulhu/scenes_1.json", "r", encoding="utf-8") as f:
    scene_list = json.load(f)  # Load all scene details
# ✅ Generate image prompts
prompts = []
for scene in scene_list:
    # Extract scene details
    scene_desc = scene["summary"]
    location = scene["location"]
    characters = ", ".join(scene["characters"])  # Convert character list to a string
    events = ", ".join(scene["events"])  # Convert event list to a string
    # ✅ Construct the final prompt using visual style settings
    prompt = (
        f"A {visual_style['mood']} scene set in {visual_style['time_period']}. "
        f"The art style is {visual_style['art_style']}, using
{visual_style['color_palette']} colors. "
        f"The environment is {visual_style['details']['environment']} under
{visual_style['details']['weather']}. "
        f"The setting is {location}, featuring {characters}. "
        f"Key events happening in this scene: {events}. "
        f"The scene is illuminated by {visual_style['details']['lighting']}. "
        f"Scene description: {scene_desc}."
    )
    prompts.append(prompt)
# ✅ Output all generated prompts
for i, p in enumerate(prompts):
    print(f"Prompt {i + 1}: {p}\n")
```

1️⃣ Load Configuration File (`config.json`)

- The script **reads the** `config.json` **file** to extract **the predefined visual style settings**.
- The `Visual_Style` section defines:
  - **Mood** (`mood`) → The overall atmosphere (e.g., Eldritch Horror).
  - **Art Style** (`art_style`) → The preferred artistic theme (e.g., Lovecraftian Horror).
  - **Color Palette** (`color_palette`) → The color scheme (e.g., Dark, Sepia, Greenish Black).
  - **Details** (`details`) → Includes settings for **environment, weather, lighting**, and **clothing**.

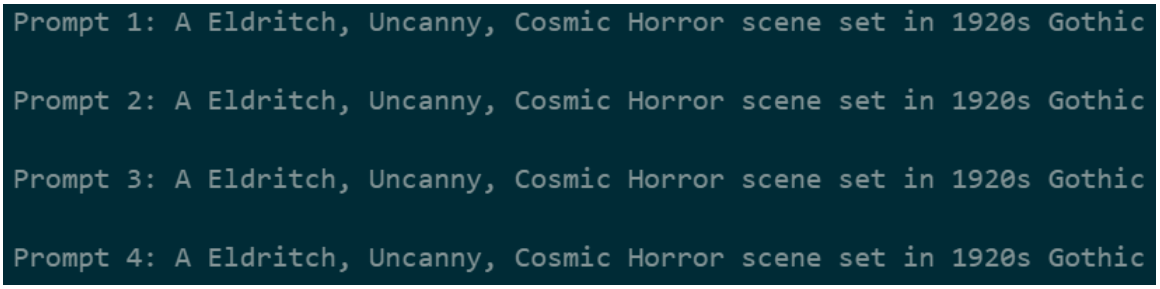
2️⃣ Load Scene Data (`scenes_1.json`)

- Reads **scene descriptions, characters, and events** from the file.
- Converts **lists (e.g., characters and events)** into comma-separated strings.

3️⃣ Generate AI Art Prompts

- Constructs **detailed prompts** for AI image generation by:
  - **Combining visual style settings with scene descriptions**.
  - Ensuring the **environment, lighting, and atmosphere match the Lovecraftian horror theme**.

4️⃣ Example Generated Prompt



```
Prompt 1: A Eldritch, Uncanny, Cosmic Horror scene set in 1920s Gothic

Prompt 2: A Eldritch, Uncanny, Cosmic Horror scene set in 1920s Gothic

Prompt 3: A Eldritch, Uncanny, Cosmic Horror scene set in 1920s Gothic

Prompt 4: A Eldritch, Uncanny, Cosmic Horror scene set in 1920s Gothic
```

```
Prompt 1: A Eldritch, Uncanny, Cosmic Horror scene set in 1920s Gothic Horror. The art style is Dark Gothic, Lovecraftian Horror, using Dark, Muted Tones, Sepia, Greenish Black colors. The environment is Foggy coastal town, ancient cyclopean ruins, deep-sea abyss under Stormy night, fog-covered city, eerie full moon. The setting is Providence, Rhode Island, featuring George Gammell Angell, Protagonist. Key events happening in this scene: Death of George Gammell Angell, Mysterious circumstances surrounding the death. The scene is illuminated by Dim gas lamps, eerie green glow, unnatural shadows. Scene description: The narrative begins with the death of the protagonist's grand-uncle, Professor George Gammell Angell, under mysterious circumstances..
```

6. Generate Illustrated Images

```python
import requests
import time
import os  # Used to create directories
import openai

# **📂 Directory to save generated images**
save_dir = "images/The_Call_of_Cthulhu"

# **📌 Check and create directory if it does not exist**
os.makedirs(save_dir, exist_ok=True)

# **🎨 Loop through prompts to generate images**
for i, prompt in enumerate(prompts):
    try:
        print(f"Generating image {i + 1} / {len(prompts)}: {prompt}")

        # **🖼️ Generate the image using DALL·E 3**
        response = openai.Image.create(
            prompt=prompt,
            model="dall-e-3",  # Uses the DALL·E 3 model
            n=1,  # Generates 1 image per request
            size="1024x1024"  # Image resolution
        )

        # **🔗 Retrieve the image URL from the response**
        image_url = response["data"][0]["url"]
        print(f"✅ Image {i + 1} generated successfully, URL: {image_url}")

        # **📥 Download the image**
        img_data = requests.get(image_url).content
        file_name = os.path.join(save_dir, f"generated_image_{i + 1}.png")
        with open(file_name, "wb") as img_file:
            img_file.write(img_data)

        print(f"📂 Image saved as {file_name}\n")

        # **⏳ Delay to avoid API rate limits**
        time.sleep(2)

    except Exception as e:
        print(f"❌ Error generating image {i + 1}: {e}\n")
```

📌 Code Explanation

1️⃣ Create Directory (`os.makedirs(save_dir, exist_ok=True)`)

- **Ensures that the `images/The_Call_of_Cthulhu/` directory exists**.
- If the directory **does not exist**, it is **automatically created**.

2️⃣ Loop Through Prompts to Generate Images (`for i, prompt in enumerate(prompts)`)

- Iterates through each **AI-generated prompt** to request an **image from DALL·E 3**.

3️⃣ Generate Images Using OpenAI's DALL·E API

```
response = openai.Image.create(
    prompt=prompt,
    model="dall-e-3",  # Use the DALL·E 3 model
    n=1,  # Generate only 1 image per request
    size="1024x1024"  # Image resolution
)
```

- Uses **DALL·E 3** to generate an image based on the given prompt.
- The **model generates 1 image per request** (`n=1`).
- The **image resolution is set to 1024x1024 pixels**.

4️⃣ Retrieve Image URL and Download the Image

```
image_url = response["data"][0]["url"]
img_data = requests.get(image_url).content
```

- Extracts **the image URL** from the API response.
- Downloads the **image data** using `requests.get()` .

5️⃣ Save the Image Locally (`with open(file_name, "wb") as img_file`)

- The image is saved **in the** `images/The_Call_of_Cthulhu/` **directory**.

- Each file is named as `generated_image_{i + 1}.png` (e.g., `generated_image_1.png`).

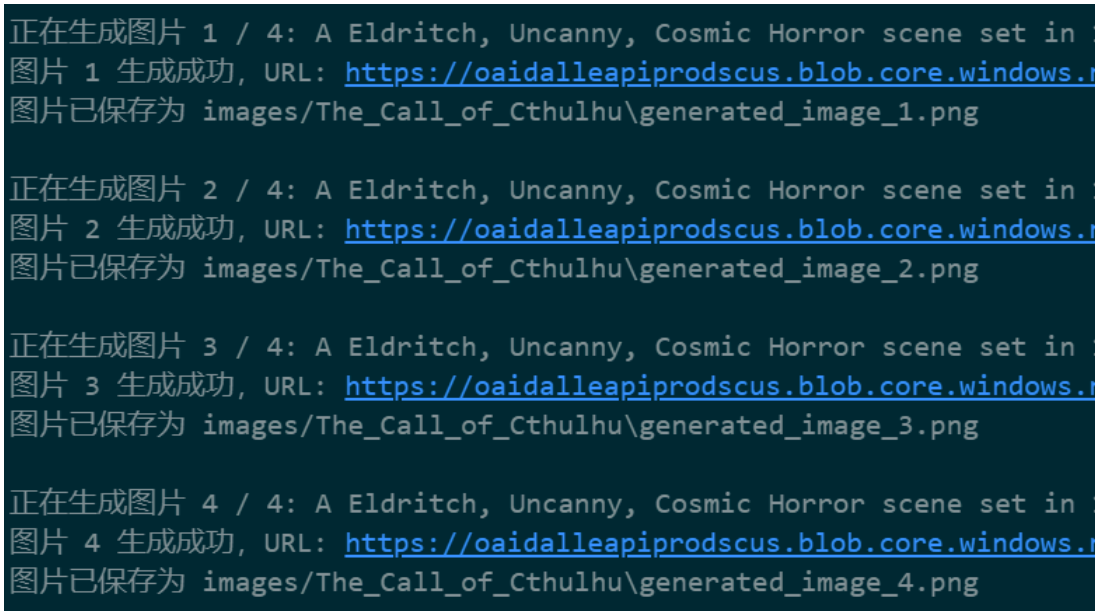
6️⃣ **Introduce a Delay (`time.sleep(2)`)**

- **Prevents exceeding OpenAI's API rate limits** by waiting **2 seconds** before making the next request.

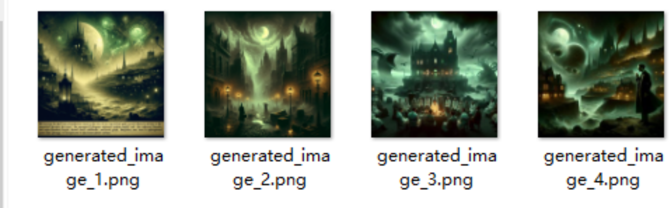
7️⃣ **Handle Errors Gracefully (`except Exception as e`)**

- If an error occurs (e.g., API failure, connection issue), it is **caught and printed**.
- The script **continues to the next prompt instead of stopping**.

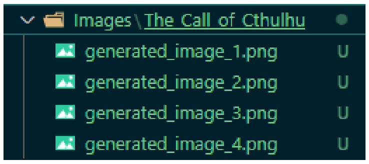
✅ **Example Console Output**



```
正在生成图片 1 / 4: A Eldritch, Uncanny, Cosmic Horror scene set in
图片 1 生成成功, URL: https://oaidalleapiprodscus.blob.core.windows.
图片已保存为 images/The_Call_of_Cthulhu\generated_image_1.png

正在生成图片 2 / 4: A Eldritch, Uncanny, Cosmic Horror scene set in
图片 2 生成成功, URL: https://oaidalleapiprodscus.blob.core.windows.
图片已保存为 images/The_Call_of_Cthulhu\generated_image_2.png

正在生成图片 3 / 4: A Eldritch, Uncanny, Cosmic Horror scene set in
图片 3 生成成功, URL: https://oaidalleapiprodscus.blob.core.windows.
图片已保存为 images/The_Call_of_Cthulhu\generated_image_3.png

正在生成图片 4 / 4: A Eldritch, Uncanny, Cosmic Horror scene set in
图片 4 生成成功, URL: https://oaidalleapiprodscus.blob.core.windows.
图片已保存为 images/The_Call_of_Cthulhu\generated_image_4.png
```



generated_image_1.png generated_image_2.png generated_image_3.png generated_image_4.png



```
Images\The_Call_of_Cthulhu
    generated_image_1.png    U
    generated_image_2.png    U
    generated_image_3.png    U
    generated_image_4.png    U
```

7. Speech Generation

```python
from pathlib import Path
import openai
import json
from gtts import gTTS  # Google Text-to-Speech (optional alternative)

# ✅ Load the scenes_1.json file
with open("scripts/The_Call_of_Cthulhu/scenes_1.json", "r", encoding="utf-8") as f:
    scene_list = json.load(f)  # Read all scene data

inputs = []  # Store scene text inputs for speech synthesis

# ✅ Iterate through each scene to generate speech
```

```
for i, scene in enumerate(scene_list):
    # Extract the original scene text
    input_text = scene["original_text"]

    # ✅ Generate speech using OpenAI TTS
    response = openai.Audio.speech.create(
        model="tts-1",  # Select the text-to-speech model (options: tts-1, tts-1-hd)
        voice="onyx",  # Choose a voice (options: alloy, echo, fable, onyx, nova,
shimmer)
        input=input_text  # The text content to be converted into speech
    )

    # ✅ Save the generated audio file
    output_path = f"voice/The_Call_of_Cthulhu/MP3_{i + 1}.mp3"
    response.stream_to_file(output_path)  # Stream the response directly to a file

    print(f"♪ Audio saved as {output_path}")

# ✅ List available voices in OpenAI TTS
openai.audio.speech.list()
```

1️⃣ Load Scene Data (`scenes_1.json`)

- Reads a JSON file containing **scene descriptions** and **dialogue**.
- Extracts the **original scene text** (`original_text`) for speech synthesis.

2️⃣ Generate Speech Using OpenAI TTS

```
response = openai.Audio.speech.create(
    model="tts-1",
    voice="onyx",
    input=input_text
)
```

- Uses **OpenAI's text-to-speech (TTS) API** to **generate high-quality speech**.
- `model="tts-1"` → Uses the standard TTS model (or `"tts-1-hd"` for higher quality).
- `voice="onyx"` → Selects a specific AI-generated voice.

3️⃣ Save the Generated Audio

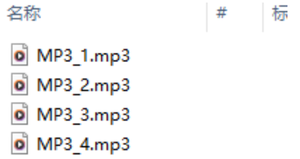
```
response.stream_to_file(output_path)
```

- Saves the AI-generated **MP3 audio file** in the `"voice/The_Call_of_Cthulhu/"` directory.
- Each file is named **MP3_1.mp3, MP3_2.mp3, …** according to the scene number.

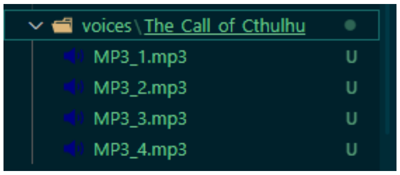
4️⃣ List Available TTS Voices

```
openai.audio.speech.list()
```

- Retrieves **a list of available AI voices** for text-to-speech conversion.

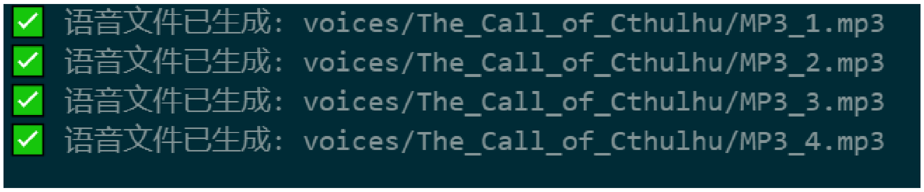


MP3_1.mp3
MP3_2.mp3
MP3_3.mp3
MP3_4.mp3


voices\The_Call_of_Cthulhu
MP3_1.mp3 U
MP3_2.mp3 U
MP3_3.mp3 U
MP3_4.mp3 U

In some versions of OpenAI, the model does not support the `openai.audio` feature. If this occurs, please use the following code:

```
from gtts import gTTS
import json

# ✅ Load the scenes_1.json file
with open("scripts/The_Call_of_Cthulhu/scenes_1.json", "r", encoding="utf-8") as f:
    scene_list = json.load(f)  # Read all scene data

inputs = []  # Store scene text inputs for speech synthesis

# ✅ Iterate through each scene to generate speech
for i, scene in enumerate(scene_list):
    input_text = scene["original_text"]  # Extract the original scene text

    # ✅ Generate speech using Google Text-to-Speech (gTTS)
    tts = gTTS(text=input_text, lang="en")

    # ✅ Save the generated audio file
    output_path = f"voices/The_Call_of_Cthulhu/MP3_{i + 1}.mp3"
    tts.save(output_path)

    print(f"✅ Audio file generated: {output_path}")
```



```
✅ 语音文件已生成: voices/The_Call_of_Cthulhu/MP3_1.mp3
✅ 语音文件已生成: voices/The_Call_of_Cthulhu/MP3_2.mp3
✅ 语音文件已生成: voices/The_Call_of_Cthulhu/MP3_3.mp3
✅ 语音文件已生成: voices/The_Call_of_Cthulhu/MP3_4.mp3
```

8. Video Generation

```python
import os
import subprocess

# ✅ Number of files (Modify this according to your dataset)
num_files = 4
video_list = []  # Store generated video filenames

# ✅ Ensure the output directory exists
os.makedirs("videos/The_Call_of_Cthulhu", exist_ok=True)

# ✅ Loop through each scene to generate individual videos
for i in range(1, num_files + 1):
    img_file = f"images/The_Call_of_Cthulhu/generated_image_{i}.png"  # Input image file
    audio_file = f"voices/The_Call_of_Cthulhu/MP3_{i}.mp3"  # Input audio file
    output_video_file = f"videos/The_Call_of_Cthulhu/temp_video_{i}.mp4"  # Output video file

    # ✅ Generate a single video using FFmpeg
    ffmpeg_cmd = [
        "ffmpeg", "-loop", "1", "-i", img_file,  # Load image as a still frame
        "-i", audio_file,  # Load audio file
        "-c:v", "libx264", "-tune", "stillimage",  # Encode video using H.264 codec optimized for still images
        "-c:a", "aac", "-b:a", "192k",  # Encode audio using AAC codec with 192kbps bitrate
        "-pix_fmt", "yuv420p",  # Set pixel format for broad compatibility
        "-shortest", output_video_file  # Ensure video duration matches audio length
    ]

    # ✅ Execute FFmpeg command
    subprocess.run(ffmpeg_cmd, check=True)
    video_list.append(f"temp_video_{i}.mp4")  # Store generated video filename

    print(f"✅ Video generated: {output_video_file}")

# ✅ Create a file list for concatenation
concat_file = "videos/The_Call_of_Cthulhu/video_list.txt"
with open(concat_file, "w") as f:
    for video in video_list:
        f.write(f"file '{video}'\n")

# ✅ Concatenate all videos into a final output file
final_output = "Output/final_output.mp4"
ffmpeg_concat_cmd = [
    "ffmpeg", "-f", "concat", "-safe", "0", "-i", concat_file,  # Use FFmpeg to concatenate videos
    "-c", "copy", final_output  # Copy streams without re-encoding
]
```

```
subprocess.run(ffmpeg_concat_cmd, check=True)
print(f"✅ Video concatenation complete: {final_output}")
```

1️⃣ Ensure Output Directory Exists

```
os.makedirs("videos/The_Call_of_Cthulhu", exist_ok=True)
```

- Creates the **output directory** (`videos/The_Call_of_Cthulhu`) if it does not exist.

2️⃣ Loop Through Image-Audio Pairs to Generate Videos

```
for i in range(1, num_files + 1):
    img_file = f"images/The_Call_of_Cthulhu/generated_image_{i}.png"
    audio_file = f"voices/The_Call_of_Cthulhu/MP3_{i}.mp3"
    output_video_file = f"videos/The_Call_of_Cthulhu/temp_video_{i}.mp4"
```

- Reads **image-audio pairs** and defines the output **video file name**.

3️⃣ Generate a Video Using FFmpeg

```
ffmpeg_cmd = [
    "ffmpeg", "-loop", "1", "-i", img_file,  # Load image as a still frame
    "-i", audio_file,  # Load audio file
    "-c:v", "libx264", "-tune", "stillimage",  # Optimize video encoding for still images
    "-c:a", "aac", "-b:a", "192k",  # Encode audio in AAC format (192kbps)
    "-pix_fmt", "yuv420p",  # Set a compatible pixel format
    "-shortest", output_video_file  # Ensure video matches audio length
]
```

- **Creates a video** from a **still image and audio file**.
- **Ensures the video duration does not exceed the audio length** (`-shortest`).

4️⃣ Execute FFmpeg to Generate Video

```
subprocess.run(ffmpeg_cmd, check=True)
video_list.append(f"temp_video_{i}.mp4")
```

- Runs the **FFmpeg command** to generate a **video file**.
- Stores the **video filename for later concatenation**.

5️⃣ Create a File List for Video Concatenation

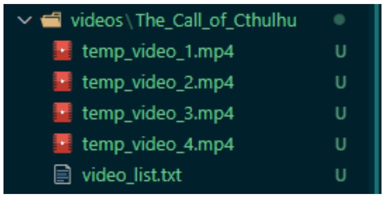
```
concat_file = "videos/The_Call_of_Cthulhu/video_list.txt"
with open(concat_file, "w") as f:
    for video in video_list:
        f.write(f"file '{video}'\n")
```

- **Generates a text file** listing all the **temporary video files**.

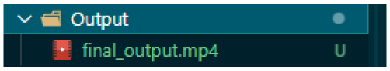
6️⃣ Concatenate All Videos into a Single Output File

```
final_output = "Output/final_output.mp4"
ffmpeg_concat_cmd = [
    "ffmpeg", "-f", "concat", "-safe", "0", "-i", concat_file,
    "-c", "copy", final_output
]
```

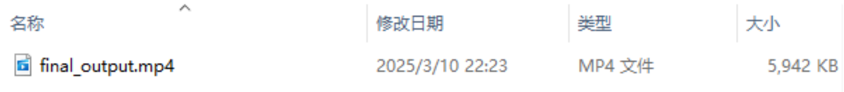
- Uses **FFmpeg's** `concat` **mode** to merge all the videos.
- **No re-encoding** (`-c copy`), ensuring **fast processing and no quality loss**.



```
videos \ The_Call_of_Cthulhu
    temp_video_1.mp4    U
    temp_video_2.mp4    U
    temp_video_3.mp4    U
    temp_video_4.mp4    U
    video_list.txt      U
```



```
Output
    final_output.mp4    U
```



名称	修改日期	类型	大小
final_output.mp4	2025/3/10 22:23	MP4 文件	5,942 KB