**AI Storyboarder – A Comprehensive Guide for Future Students**

**Introduction**

Welcome to the "AI Storyboarder" Toolbox! This document is your complete guide to understanding, recreating, and enhancing an innovative project that uses Artificial Intelligence (AI) to streamline the storyboarding process for creative media, such as films, animations, and games. Whether you're a beginner or have some experience with AI, this toolbox provides everything you need to dive into this exciting intersection of technology and creativity.

**What is AI Storyboarder?**

The AI Storyboarder is a web-based application that leverages AI to generate storyboards from a user’s story idea. It combines ChatGPT 3.5-turbo for text generation (scene descriptions, dialogue, and emotions) and DALL-E 3 for image generation (visual frames) to produce a sequence of scenes and shots. Users can:

- Input a story idea.

- Define a main character via a Character Editor.

- Select a visual style (e.g., cinematic, anime) and camera angles.

- Generate, edit, and refine the storyboard interactively.

The result is a professional-quality storyboard that saves time and inspires creativity.

**Purpose and Benefits**

Storyboarding is a critical step in creative industries, traditionally requiring artistic skills and significant time investment. The AI Storyboarder transforms this process by:

- Saving Time: Generating a storyboard in minutes rather than hours.

- Inspiring Creativity: Providing AI-driven suggestions to spark new ideas.

- Enhancing Accessibility: Allowing non-artists to create compelling visual narratives.

This project is especially relevant for Creative Media & Game Technologies students, showcasing how AI can enhance creative workflows and preparing you for an AI-driven industry future.

**Understanding the Technology**

**Core AI Technologies**

The AI Storyboarder is powered by two advanced AI models from OpenAI:

**1. ChatGPT 3.5-turbo (Text Generation):**

- Role: Generates structured text for scenes, shots, dialogue, and emotions based on user inputs.

- How It Works: Built on a transformer architecture, GPT-3.5-turbo uses self-attention mechanisms to process and generate coherent text. It excels at understanding context over long sequences, making it ideal for creating detailed storyboard descriptions in a JSON format.

- Key Insight: Transformers enable parallel processing of input data, improving efficiency and quality of output compared to older models like RNNs.

**2. DALL-E 3 (Image Generation):**

- Role: Creates visual frames for each shot based on text prompts derived from GPT’s output, incorporating style, character details, camera angles, and emotions.

- How It Works: Utilizes a diffusion model, which starts with random noise and iteratively refines it into a coherent image matching the text prompt. This process leverages vast datasets of text-image pairs for high-quality generation.

- Key Insight: Diffusion models offer superior image diversity and detail, though they require careful prompt design for consistency.

**Theoretical Foundations**

- Transformers (ChatGPT 3.5-turbo): Introduced by Vaswani et al. (2017), transformers rely on self-attention to weigh word importance, enabling robust text generation. In this project, they ensure scene descriptions align with the story idea.

- Diffusion Models (DALL-E 3): As explored by Ramesh et al. (2022), diffusion models reverse a noise-adding process, creating images that match textual descriptions. This underpins the tool’s ability to visualize scenes accurately.

**Supporting Technologies**

- Flask (Backend): A Python micro-framework that hosts the server, manages API requests, and integrates with OpenAI’s APIs.

- HTML/CSS/JavaScript (Frontend): Provides an intuitive interface for user inputs and storyboard display.

- Fetch API: Facilitates asynchronous communication between the frontend and backend.

**Step-by-Step Guide to Deploy AI Storyboarder**

**Prerequisites**

To get started, ensure you have:

- Python 3.8+ (download from <python.org>).

- A code editor (e.g., Visual Studio Code).

- An OpenAI API Key (obtainable from <openai.com>; note: requires a paid account as of 2025).

- Basic familiarity with Python, HTML, CSS, and JavaScript.

**Step 1: Set Up Your Environment**

1. Create a Project Folder:

- Name it «AI-Storyboarder» and navigate to it in your terminal.

2. Set Up a Virtual Environment:

- Run: *python -m venv venv*

- Activate it:

- Windows: *venv\Scripts\activate*

- Mac/Linux: *source venv/bin/activate*

3. Install Dependencies:

- Run: *pip install flask flask-cors openai*

4. Configure the OpenAI API Key:

- Set it as an environment variable:

- Windows: *set OPENAI\_API\_KEY=your\_api\_key*

- Mac/Linux: *export OPENAI\_API\_KEY=your\_api\_key*

- Replace *your\_api\_key* with your actual key.

**Step 2: Build the Backend**

1. Create app.py:

- In your project folder, create app.py with the following structure:

*python*

*from flask import Flask, request, jsonify*

*from flask\_cors import CORS*

*import openai*

*import time*

*import os*

app = Flask(\_\_name\_\_)

CORS(app) # Enable CORS for frontend communication

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

def dalle\_prompt(style, character, angle, description, emotion):

return f"A {style} style shot of {character}, {angle} camera angle, {description}, expressing {emotion}"

@app.route('/generate-storyboard', methods=['POST'])

def generate\_storyboard():

data = request.json

story = data['story']

character = data['character']

style = data['style']

angle = data['angle']

# GPT prompt for structured storyboard

prompt = f"Generate a storyboard as a JSON array with 3 scenes, each with 2 shots, for this story: {story}. Include description, emotion, and dialogue for each shot."

response = openai.ChatCompletion.create(

model="gpt-3.5-turbo",

messages=[{"role": "user", "content": prompt}]

)

storyboard = response.choices[0].message['content']

storyboard\_data = eval(storyboard) # Convert string to Python list

# Add images using DALL·E 3

for scene in storyboard\_data:

for shot in scene['shots']:

image\_prompt = dalle\_prompt(style, character, angle, shot['description'], shot['emotion'])

image\_response = openai.Image.create(prompt=image\_prompt, n=1, size="1024x1024")

shot['image\_url'] = image\_response['data'][0]['url']

time.sleep(12) # Respect rate limits

return jsonify(storyboard\_data)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

2. \*\*Key Notes:\*\*

- \*\*Prompt Engineering:\*\* The dalle\_prompt function ensures detailed, consistent image prompts.

- \*\*Rate Limiting:\*\* A 12-second delay prevents exceeding OpenAI’s API limits.

3. \*\*Test the Server:\*\*

- Run: python app.py

- Verify it runs at http://127.0.0.1:5000.

### \*\*Step 3: Build the Frontend\*\*

1. \*\*Create index.html:\*\*

- Add this basic structure:

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>AI Storyboarder</title>

<style>

body { font-family: Arial, sans-serif; padding: 20px; }

.shot { border: 1px solid #ccc; margin: 10px; padding: 10px; }

</style>

</head>

<body>

<h1>AI Storyboarder</h1>

<textarea id="story" rows="4" cols="50" placeholder="Enter your story idea"></textarea><br>

<input id="character" type="text" placeholder="Describe your main character"><br>

<select id="style">

<option value="cinematic">Cinematic</option>

<option value="anime">Anime</option>

</select>

<select id="angle">

<option value="wide">Wide</option>

<option value="close">Close</option>

</select><br>

<button onclick="generateStoryboard()">Generate</button>

<button onclick="clearStoryboard()">Clear</button>

<div id="storyboard"></div>

<script src="script.js"></script>

</body>

</html>

2. \*\*Create script.js:\*\*

- Add this JavaScript:

javascript

async function generateStoryboard() {

const story = document.getElementById('story').value;

const character = document.getElementById('character').value;

const style = document.getElementById('style').value;

const angle = document.getElementById('angle').value;

const response = await fetch('http://127.0.0.1:5000/generate-storyboard', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ story, character, style, angle })

});

const data = await response.json();

const storyboardDiv = document.getElementById('storyboard');

storyboardDiv.innerHTML = '';

data.forEach((scene, i) => {

const sceneDiv = document.createElement('div');

sceneDiv.innerHTML = <h2>Scene ${i + 1}</h2>;

scene.shots.forEach(shot => {

const shotDiv = document.createElement('div');

shotDiv.className = 'shot';

shotDiv.innerHTML =

<p><strong>Description:</strong> ${shot.description}</p>

<p><strong>Emotion:</strong> ${shot.emotion}</p>

<p><strong>Dialogue:</strong> ${shot.dialogue}</p>

<img src="${shot.image\_url}" width="300">

;

sceneDiv.appendChild(shotDiv);

});

storyboardDiv.appendChild(sceneDiv);

});

}

function clearStoryboard() {

document.getElementById('storyboard').innerHTML = '';

document.getElementById('story').value = '';

document.getElementById('character').value = '';

}

3. \*\*Run the Frontend:\*\*

- Use a local server (e.g., VS Code’s Live Server) to open index.html.

### \*\*Step 4: Test the Application\*\*

- Start the Flask server (python app.py).

- Open index.html in a browser.

- Input a story idea, character, style, and angle, then click "Generate" to see your storyboard.

---

## \*\*Iterations and Improvements (Competency B2)\*\*

### \*\*Implemented Enhancements\*\*

The project evolved through several iterations:

- \*\*Character Editor:\*\* Ensures consistent character descriptions across shots.

- \*\*Prompt Refinement:\*\* Structured prompts for DALL·E 3 now include style, character, angle, and emotion for better image quality.

- \*\*Interactive Features:\*\* Added "Regenerate Shot," "Add Shot Below," and "Delete Shot" options for user control.

### \*\*Proposed Future Enhancements\*\*

- \*\*PDF Export:\*\* Export storyboards as downloadable PDFs.

- \*\*Image Editing:\*\* Allow users to tweak AI-generated images within the tool.

- \*\*Cost Reduction:\*\* Explore open-source AI models to minimize API dependency.

---

## \*\*Challenges, Tips, and Reflections\*\*

### \*\*Challenges\*\*

- \*\*API Rate Limits:\*\* DALL·E 3’s limits require delays between requests.

- \*\*Prompt Consistency:\*\* Vague prompts lead to erratic outputs.

- \*\*Visual Coherence:\*\* Maintaining character consistency across shots is challenging.

### \*\*Tips\*\*

- Use specific prompts (e.g., “A cinematic wide shot of a brave knight, short black hair, in a forest, feeling determined”).

- Cache images during testing to reduce API calls.

- Test with small storyboards first to refine prompts.

### \*\*Reflections\*\*

- \*\*AI’s Power and Limits:\*\* The tool excels at rapid generation but struggles with fine-grained consistency, highlighting the need for human oversight.

- \*\*Iterative Design:\*\* User feedback drove key improvements, underscoring its value in development.

---

## \*\*Added Value and Real-World Impact (Competency E3)\*\*

### \*\*Value Proposition\*\*

- \*\*Efficiency:\*\* Cuts storyboarding time dramatically (e.g., a three-scene storyboard in under five minutes).

- \*\*Creativity Boost:\*\* AI suggestions inspire novel ideas.

- \*\*Inclusivity:\*\* Empowers non-artists to visualize stories.

### \*\*Applications\*\*

- \*\*Film Pre-Production:\*\* Quick scene visualization.

- \*\*Game Design:\*\* Rapid narrative prototyping.

- \*\*Education:\*\* Teaches AI integration in creative workflows.

### \*\*Limitations\*\*

- \*\*Cost:\*\* API usage can be expensive for frequent use.

- \*\*Ethics:\*\* Potential biases in AI outputs require monitoring.

---

## \*\*Conclusion\*\*

The AI Storyboarder exemplifies AI’s transformative potential in creative industries. This toolbox equips you to recreate it, understand its technologies, and push its boundaries—preparing you for a future where AI and creativity converge.

---

### \*\*References\*\*

- Vaswani, A., et al. (2017). "Attention is All You Need." \*Advances in Neural Information Processing Systems\*.

- Ramesh, A., et al. (2022). "Hierarchical Text-Conditional Image Generation with CLIP Latents." \*arXiv preprint arXiv:2204.06125\*.

- OpenAI Documentation: [https://platform.openai.com/docs/](https://platform.openai.com/docs/)