ScrapUncle Image Generation for FAQ's Pipeline

Overview

This project automates the generation of **cartoon-style, 3D vector images** for ScrapUncle FAQs by combining **natural language processing (NLP)** and **text-to-image generation**. It uses:

- **Gemini AI** to break down long answers into meaningful visual steps.
- **ClipDrop API** to turn each step into a high-quality cartoon image.
- **Python threading** to speed up the image generation process.
- **Batching** for grouped processing.
- **Structured folder management** for organizing output by categories.

Core Idea

Many FAQ answers are long and not directly visualizable. This pipeline transforms those answers into a series of **storyboard-style cartoon images** by:

1. **Extracting steps** using a language model (Gemini).
2. **Generating prompts** per step using category-aware templates.
3. **Generating images** using those prompts via ClipDrop API.
4. **Saving outputs** in organized folders for easy usage.

This is ideal for companies like ScrapUncle that want to visualize their support content, app tutorials, or marketing flows.

🗂️ **Folder & File Structure**

📁 **Input Folder**

- A CSV file with three columns:
    - question: The FAQ question.
    - answer: Textual answer to the FAQ.

- category: High-level topic the FAQ belongs to (e.g., "Pickup Process").

Example-

question,answer,category

"What is the minimum quantity for paper pickup?","You must have at least 5 kg", "Pickup Process"

📁 **Output Folder**

Outputs are stored in the following structure:

plaintext

outputs/

|

├── pickup_process/

| ├── q_00_What_is_minimum_quantity/

| | ├── 01_Bag_request.png

| | ├── 02_Agent_arrives.png

| | └── ...

| ├── q_01_How_quickly_can_you_pick_up/

| | ├── 01_Schedule_pickup.png

| | ├── 02_Weighing_scrap.png

| | └── ...

| └── ...

└── location/

└── ...

Each question creates its own folder inside a category, and each step becomes a uniquely named image.

🔧 Major Functions & Their Roles

Function Name	Purpose
prepare_scrapuncle_folders_and_dataframes()	Splits the CSV by category, creates subfolders, and maps each to a DataFrame.
get_steps(answer)	Sends the answer to Gemini and gets clean, concise, step-wise instructions.
generate_clipdrop_image()	Converts a single step into a high-quality 3D vector image using ClipDrop API.
generate_images_from_df()	For each question in a category, generates images in parallel for all steps.
generate(sample_df)	Orchestrates the full process — category by category.

Theoretical Concepts Behind Each Step

1. Answer Step Decomposition with Gemini

Instead of manually identifying visual elements in answers, we use Gemini to extract:

- **Logical Steps**
- **Scene Descriptions**
- **Visual Cues for Prompts**

This uses the **text summarization** and **semantic segmentation** capabilities of LLMs, transforming text into **visual blueprints**.

---

2. Prompt Engineering per Category

Each step is passed through a category-specific prompt template. For example:

Category: pickup_process_df

"Cartoon-style image showing: “{step}” during ScrapUncle pickup — agent collecting scrap from a doorstep, van parked, items in bags or boxes."

This ensures:

- Relevance to the ScrapUncle brand
- Style consistency across categories
- Higher quality results from ClipDrop

---

3. Image Generation via ClipDrop API

- ClipDrop’s text-to-image/v1 model is used.
- Returns a PNG image from the given prompt.
- ClipDrop supports **detailed cartoon vector style**, which matches our requirements better than most open-source models.

---

4. Parallelism with Threading

We use ThreadPoolExecutor to:

- Send multiple image generation requests simultaneously
- Significantly reduce generation time per question (from 3–5 minutes to ~30s–1min per question)

---

5. Batching (Optional)

Batching can be applied when processing multiple questions at once, grouping them for more efficient memory and API usage.

Currently used when run_full_scrapuncle_pipeline() is executed for all categories.

Usage Instructions

Step-by-Step

1. Load the CSV

```
import pandas as pd
from your_script import generate
df = pd.read_excel("new questions.xlsx")
sample_df = df.sample(3)  # Or use full df
```

generate(sample_df)

Key Features

- Fully automated
- Cartoonish vector images for brand storytelling
- Multi-threaded for speed
- Flexible per-question or per-category processing
- Prompt templating per category

API Requirements

You'll need:

- Gemini API access for step breakdown (Google Generative AI)
- ClipDrop API key (free/paid tier)

Set it up in your code:

```
API_KEY = "your-clipdrop-api-key"
```

Things You Can Extend

- Add multilingual support for step generation.
- Cache images to avoid redundant generation.
- Use Stable Diffusion locally as a fallback model.
- Add GUI to review generated prompts and images.