Dataset Preparation for Stable Diffusion Using Google Gemini

Stable Diffusion is a powerful generative artificial intelligence (AI) model that primarily uses a latent diffusion model (LDM) to produce high-quality, detailed images from text descriptions (text-to-image generation) and other image-based inputs.

- **Diffusion Process**: The core idea is based on diffusion models, which learn to iteratively reverse a process of gradually adding noise to an image.
  - **Forward Diffusion**: Gaussian noise is progressively added to the original image until it becomes pure random noise.
  - **Reverse Diffusion(Generation)** The model learns to reverse this process, starting from pure noise and using a U-Net noise predictor guided by a text prompt to "denoise" the image iteratively back into a coherent and detailed output image.
- **Latent Space**: Stable Diffusion operates in a lower-dimensional latent space (thanks to a Variational Autoencoder or VAE), rather than the high-dimensional pixel space. This significantly reduces computational requirements, making it more accessible.

Leveraging Google Gemini for Dataset Preparation

The key bottleneck in preparing a dataset is often the laborious and subjective process of manual image captioning. Google's Gemini model, being multimodal, is exceptionally useful for automating this step:

Gemini can process visual input (the image) and generate highly descriptive, natural-language text output (the caption), acting as a powerful and efficient auto-captioner.

How to Use Gemini for Captioning:

- Access the API: You can use the Gemini API (e.g., via the Google Gen AI SDK) which supports multimodal input.
    - How to Use Gemini for Captioning

      Step 1: Get a Gemini API Key

      To use Gemini programmatically, you need an API key:

      Go to the Gemini developer console: https://ai.google.dev

      Sign in with your Google account

      Click Get API Key
- Input: You pass the image file (or a base64 encoded version of the image) along with a text prompt that instructs the model on what kind of caption to generate.

This Python code is a comprehensive script designed to extend a public dataset of Flat UI screenshots by generating Stable Diffusion-ready text captions for some of the images using the Google Gemini API, and then uploading the augmented dataset to Kaggle.

1. Install & import everything
2. Setup Kaggle access

3. Load a base dataset (Flat UI images)
4. Visualize sample images
5. Use Gemini to generate captions for some images
6. Combine old + new data into one dataset
7. Save it to disk in Kaggle format
8. Upload to Kaggle as a new dataset

Step 1: Installation and Imports

The initial steps ensure all necessary libraries are installed and imported.

- `datasets` : For loading and manipulating datasets, particularly from Hugging Face.
- `google-generativeai` : The official Google library for interacting with the Gemini API.
- `kaggle` : The command-line interface tool to interact with Kaggle for dataset upload.
- `pillow` , `matplotlib` , `pandas` : Standard libraries for image handling, plotting, and data manipulation.

In [ ]:
```
# 1. Install everything
!pip install -q datasets google-generativeai kaggle pillow matplotlib pandas
```

In [ ]:
```
# Import required libraries
from datasets import load_dataset, concatenate_datasets, Dataset, DatasetDict
from PIL import Image
import io, os, time, json
import google.generativeai as genai
import matplotlib.pyplot as plt
import subprocess
from IPython.display import clear_output
```

2. Kaggle API Setup

This block sets up the necessary authentication for the Kaggle CLI to upload the dataset.

- `from google.colab import files` : Imports the utility to handle file uploads in a Google Colab environment.
- `files.upload()` : Prompts the user to upload their kaggle.json file, which contains their API credentials.
- `!mkdir -p ~/.kaggle` : Creates a hidden directory named .kaggle in the home directory. This is the standard location where the Kaggle CLI expects to find the credentials file.
- `!cp kaggle.json ~/.kaggle/` : Copies the uploaded kaggle.json file into the newly created .kaggle directory.
- `!chmod 600 ~/.kaggle/kaggle.json` : Changes the file permissions to ensure only the owner (the user) can read and write to the file, which is a security requirement for the Kaggle CLI.

In [ ]:
```
#2. Kaggle API setup
# You MUST upload your `kaggle.json` from https://www.kaggle.com/settings/account → API → Create Token
from google.colab import files

print("Upload your kaggle.json (download from https://www.kaggle.com/settings → API)")
files.upload()  # Opens a file picker to upload kaggle.json


# Move kaggle.json to correct location and set correct permissions
!mkdir -p ~/.kaggle                 # create Kaggle config folder if not exists
!cp kaggle.json ~/.kaggle/          # copy credentials there
!chmod 600 ~/.kaggle/kaggle.json    # secure the file so only you can access it (required by Kaggle)
```

```
Upload your kaggle.json (download from https://www.kaggle.com/settings → API)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle (3).json
```

3. Load Original Dataset

This step loads the base dataset from the Hugging Face Hub.

`old_dataset= load_dataset("bhomik7/flat-UI-dataset-small")` : Loads the "flat-UI-dataset-small" dataset, which contains UI screenshots as binary image data. The `old_dataset` is typically a `DatasetDict` containing splits like 'train', 'test', etc.

In [ ]:
```
# 3. Load original dataset
old_dataset = load_dataset("bhomik7/flat-UI-dataset-small")
#print(f"Original size: {len(old_dataset['train'])} images")
```

4. Show Samples

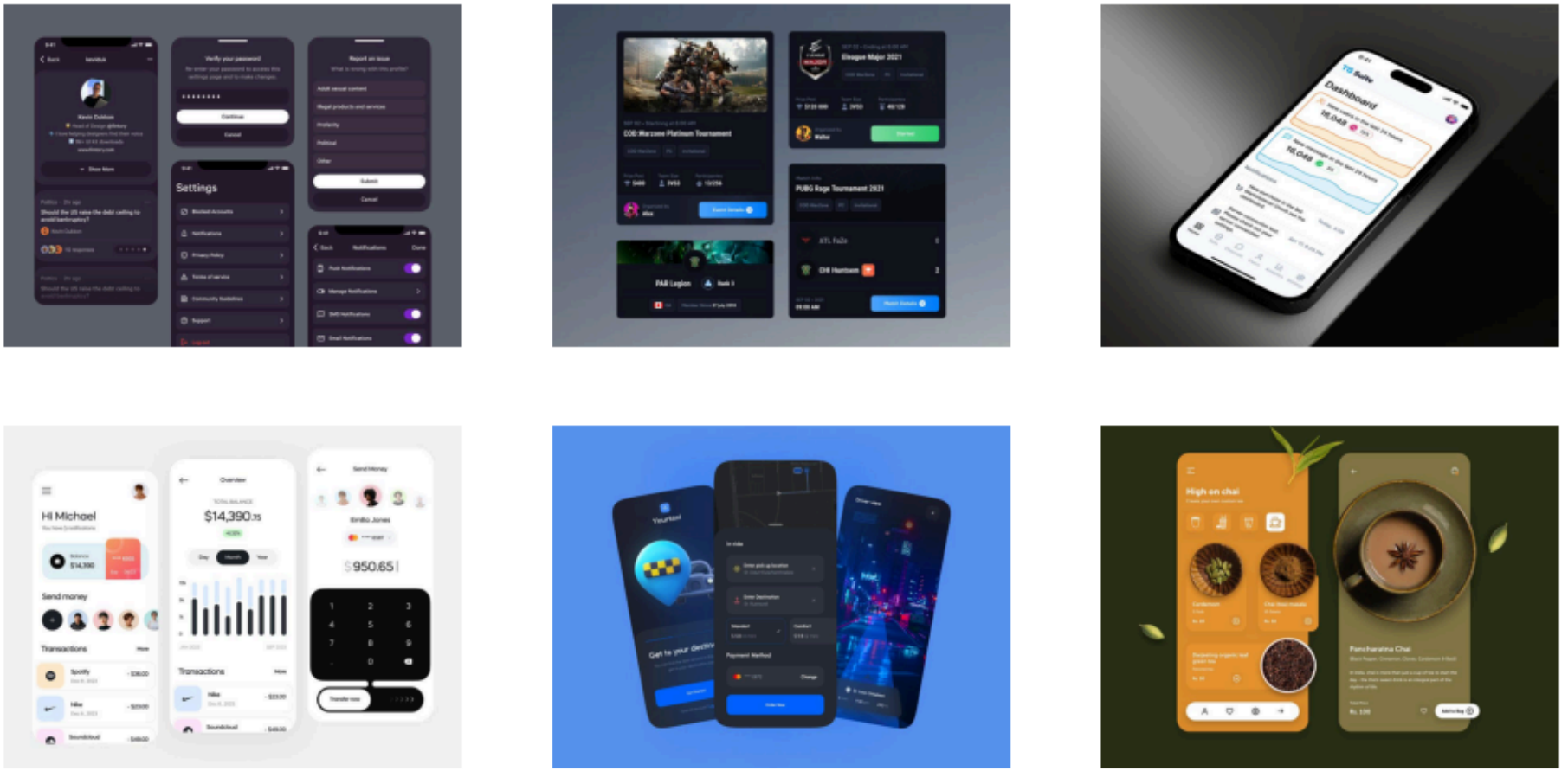
This defines a utility function to visually inspect a few images from the dataset.

- `def show_samples(ds, n=6): ...` : The function takes a dataset split (`ds`) and the number of samples (`n`) to show.
- It iterates, opens the image from its raw binary bytes using `io.BytesIO` and `PIL.Image.open()` , and plots them using `matplotlib.pyplot` in a grid format. This confirms the images loaded correctly.

In [ ]:
```
# 4. Show samples
def show_samples(ds, n=6):
    """
    Display n sample images from the dataset.
    ds: Hugging Face dataset or dict with "image" keys (raw bytes).
    n:  number of samples to show (default = 6)
    """
    plt.figure(figsize=(12,6))
    for i in range(n):
        # Convert raw image bytes into a displayable image
        img = Image.open(io.BytesIO(ds[i]["image"]))
        # Create subplot grid (2 rows x 3 columns)
        plt.subplot(2, 3, i+1)
        plt.imshow(img)
        plt.axis('off')
    plt.suptitle("Original Flat UI Samples")
    plt.show()
```

```
# Show example UI screenshots from the training set
show_samples(old_dataset["train"])
```

Original Flat UI Samples



5. Gemini Setup and Captioning Function

This block configures the Gemini API access and defines the core function for generating captions.

- `genai.configure(api_key="...")` : Initializes the Gemini API client with the user's API key.

- `model = genai.GenerativeModel(...)` : Creates an instance of the model to be used:
  - " `gemini-2.0-flash` ": A fast and capable multimodal model.
  - `system_instruction` : A powerful instruction to guide the model's behavior, directing it to output a short, precise description for a Stable Diffusion (SD) image generation model.
  - `generation_config` : Sets parameters like temperature (creativity) and max_output_tokens.
- `def gemini_caption(image_bytes)` : ...: This function takes raw image bytes and:
- Opens the image, converts it to RGB format and saves the image into an in-memory buffer (`io.bytes`) as a JPEG
- Uploads the buffered image data to the Gemini API using `genai.upload_file` (). This is necessary for the model to process the image.
- Calls `model.generate_content()` with both the uploaded image file and a specific prompt ("Give me a perfect SD prompt for this UI.").
- `time.sleep(2)` : Pauses execution to prevent hitting API rate limits (sending too many requests too quickly).

Returns the generated text caption after stripping whitespace.

In [ ]:
```python
# 5. Configure Gemini API with your personal key (replace placeholder)
genai.configure(api_key="put_your_gemini_API_key_here")

# Define the Gemini model and its behavior
model = genai.GenerativeModel(
    "gemini-2.0-flash",  # Gemini model name
    system_instruction=(
        "Describe this UI screenshot in <200 words so Stable Diffusion can recreate it exactly."
    ),
    generation_config={
        "temperature": 0.7,          # randomness in generation
        "max_output_tokens": 500     # maximum response length
    }
)

def gemini_caption(image_bytes):
    """
```

```
    Send an image to Gemini and get back a high-quality caption
    that can be used as a Stable Diffusion prompt.
    """
    # Load the image from bytes and convert to RGB
    img = Image.open(io.BytesIO(image_bytes)).convert("RGB")

    # Re-encode it as JPEG (Gemini needs a file-like input)
    buf = io.BytesIO()
    img.save(buf, format="JPEG")
    buf.seek(0)

    # Upload to Gemini
    file = genai.upload_file(buf, mime_type="image/jpeg")

    # Ask Gemini to describe it for Stable Diffusion
    response = model.generate_content([file, "Give me a perfect SD prompt for this UI."])

    # Small delay to respect API rate limits
    time.sleep(2)

    # Return the clean text response
    return response.text.strip()
```

6. Generate New Captions

This is the loop where the actual caption generation takes place.

- `for i in range(10):` : The script iterates through the first 10 images of the training split (the user notes this can be increased).
- Inside the loop
  - The raw image bytes are fetched.
  - `gemini_caption` is called to get a high-quality, Stable Diffusion-ready text description.
- The image bytes and the generated caption are stored in `new_images` and `new_texts` lists, respectively.
- A progress message is printed, showing the start of the caption.

In [ ]:

```
new_images = []
new_texts  = []

# Generate captions for first 10 images (can increase to 100 or 1000 later)
for i in range(10):
    print(f"Captioning image {i}...")

    # Get image bytes from dataset
    img_bytes = old_dataset["train"][i]["image"]

    # Use Gemini to create a caption (SD prompt)
    caption = gemini_caption(img_bytes)

    # Store the results
    new_images.append(img_bytes)
    new_texts.append(caption)

    # Show preview of the caption
    print(f"- {caption[:80]}...")
```

```
Captioning image 0...
- Here's a prompt to recreate the UI screenshot for Stable Diffusion:

**Detailed ...
Captioning image 1...
- A dark-themed UI design featuring tournament cards for various games. The top-le...
Captioning image 2...
- A close-up shot of a modern smartphone displaying a clean, minimalist UI for a "...
Captioning image 3...
- Here's a prompt to recreate the UI screens:

**UI Design, Mobile App Interface:*...
Captioning image 4...
- A conceptual UI design for a ride-sharing app. Three phone screens are displayed...
Captioning image 5...
- UI design for a mobile application, split screen layout, with a dark olive green...
Captioning image 6...
- Here is a prompt that should recreate the image well:

A UI mockup featuring thr...
Captioning image 7...
- UI design showcasing four modal windows on a dark background. The top-left modal...
Captioning image 8...
- Here's a prompt to recreate the UI screenshot:

**Prompt:**

UI design of a car ...
Captioning image 9...
- Here's a Stable Diffusion prompt to recreate the UI screenshot, optimized for ac...
```

7. Build Final Dataset

The newly captioned data is merged with the original, uncaptioned data to create a single, unified dataset.

- `data = { ... }`: A dictionary is constructed to hold the data:
    - `"image"`: Contains the 10 captioned images plus all images from the original dataset.

- "`text`": Contains the 10 new captions plus an empty string ("") for every image from the original dataset (which had no pre-existing caption).
- `full_dataset = Dataset.from_dict(data)`: Creates a Hugging Face Dataset object from the combined dictionary.

In [ ]:
```python
#7. Build one unified dataset with images and captions
data = {
    # Combine new and old image bytes
    "image": new_images + [row["image"] for row in old_dataset["train"]],
    # Use new captions + empty strings for old (uncaptioned) images
    "text":  new_texts  + [""] * len(old_dataset["train"])
}

# Convert to a Hugging Face Dataset object
full_dataset = Dataset.from_dict(data)

print(f"FINAL SIZE: {len(full_dataset)} images")
```

```
FINAL SIZE: 20 images
```

8. Save to Folder for Kaggle

The `full_dataset` is saved into a local directory structure required for a Kaggle dataset upload.

- `SAVE_DIR = "flat_UI_extended_gemini"`: Defines the name of the directory.
- `os.makedirs(SAVE_DIR, exist_ok=True)`: Creates the output directory.
- The loop iterates through every item in the full_dataset:
  - Image Saving: It opens the image bytes and saves it as a JPEG file (`.jpg`) inside the `SAVE_DIR`. File names are padded with leading zeros (e.g., 00000.jpg).
  - Caption Saving: It saves the corresponding caption into a matching `.txt` file. This structure is common for image-text datasets.

In [ ]:
```python
# 8. Save the dataset locally (for Kaggle upload)
SAVE_DIR = "flat_UI_extended_gemini"
```

```
os.makedirs(SAVE_DIR, exist_ok=True)

for i in range(len(full_dataset)):
    img_bytes = full_dataset[i]["image"]
    caption   = full_dataset[i]["text"] or "flat UI screenshot"

    # Save the image as a numbered JPG
    Image.open(io.BytesIO(img_bytes)).save(f"{SAVE_DIR}/{i:05d}.jpg")

    # Save the caption text (same filename, .txt extension)
    open(f"{SAVE_DIR}/{i:05d}.txt", "w").write(caption)
```

9. Kaggle Metadata

A `dataset-metadata.json` file is created. This file is crucial as the Kaggle CLI requires it to correctly title and license the uploaded dataset.

- The metadata includes a descriptive title, a unique ID (which includes the user's username, and the dataset name), and a license (here, the permissive CC0).

In [ ]:
```
# 9. Create Kaggle metadata (dataset info file)
meta = {
    "title": "Flat UI + Gemini Captions (Stable Diffusion Ready)",
    "id": "bhomik7/flat-ui-dataset-extended",   # Kaggle dataset slug
    "licenses": [{"name": "cc0"}]               # open license (public domain)
}

# Write metadata to JSON file (required by Kaggle)
json.dump(meta, open(f"{SAVE_DIR}/dataset-metadata.json", "w"), indent=2)
```

10. Kaggle Upload

This is the final, essential step where the local folder is published to Kaggle.

- `cmd = f'kaggle datasets create -p "{SAVE_DIR}"'`: The command to attempt to create a new dataset. The `-p` flag specifies the local folder to upload.

- `subprocess.run(...)`: Executes the command.
- The output is checked for the message "already exists":
  - If it exists, the command `!kaggle datasets version -p "{SAVE_DIR}" -m "..."` is run to upload the data as a new version of the existing dataset. This is the correct way to update an existing Kaggle dataset.
  - If it does not exist, the dataset is created for the first time.
- Finally, the live link and instructions for loading the new dataset using the datasets library are printed.

In [ ]:
```
SAVE_DIR = "flat_UI_extended_gemini"

# Run Kaggle dataset creation quietly
result = subprocess.run(
    f'kaggle datasets create -p "{SAVE_DIR}"',
    shell=True,
    capture_output=True,
    text=True
)

# Clear the verbose output from the cell
clear_output(wait=True)

# Check if dataset already exists
if "already exists" in result.stdout.lower():
    print("Dataset exists → creating NEW VERSION")
    # Create a new version quietly
    result_version = subprocess.run(
        f'kaggle datasets version -p "{SAVE_DIR}" -m "10 fresh Gemini captions + original flat UI"',
        shell=True,
        stdout=subprocess.DEVNULL,  # hide standard output
        stderr=subprocess.DEVNULL   # hide errors (optional)
    )
else:
    print("Dataset created successfully!")

# Print only final info and link
print("\n Upload done.")
```

```
print(" Dataset ready on Kaggle.")
print("Load later with: kaggle://bhomik7/flat-ui-dataset-extended")
```

```
Dataset created successfully!

 Upload done.
 Dataset ready on Kaggle.
Load later with: kaggle://bhomik7/flat-ui-dataset-extended
```

11. Preview Dataset Images

In [ ]:

```
# 11. Get the 20th image (index 19)
image_bytes = full_dataset[19]["image"]
caption = full_dataset[19]["text"]

# Convert to displayable format
img = Image.open(io.BytesIO(image_bytes))

# Display image and caption
display(img)
print(caption)
```

9:41

Welcome to Medico reminder app

Reminders for taking medications, tracks dosage history and health reports.

Get Started →

Already have an account? Sign In.

9:41

Forgot Password

Select an option to reset your password.

Password

Mobile OTP

Google Mail

Send Password

9:41

Set Reminder

Select Duration

15 Aug. Th to 09 Sep. Mo

Repeat Reminder

All Days | Sun | Mon | Tue | Wed | Thru | Fri | Sat

Select Time

09.00 am 10.30 pm +

Write Instructions

Set Reminder →