**Proof of Concept Report: Poster Generation Pipeline (Phase 1)**

**Date:** July 04, 2025

**Executive Summary**

This report summarizes the successful completion of Phase 1 of the Poster Generation Proof of Concept (PoC) for a Generative AI Bootcamp targeting a Gen Z audience. The pipeline, implemented in a Kaggle Notebook (draft (5).ipynb), leverages Stable Diffusion XL (SDXL) for base image generation and a custom rendering script for text placement, QR code integration, and social media logos. The pipeline meets the visual and functional requirements for creating an EdTech poster but faces limitations due to Kaggle’s free-tier constraints. Phase 2, as outlined in the accompanying *Project Deployment Report-POSTER GENERATON.docx*, proposes transitioning to Paperspace’s paid tier with Stable Diffusion 3.5 Large (SD 3.5 Large) for enhanced capabilities and Streamlit app deployment. This report provides a detailed analysis of the code, execution flow, and insights from Phase 1, aligning with the requirements for Phase 2.

**Phase 1 Overview**

The PoC was developed to create a visually appealing 1024x1024 poster for an EdTech Generative AI Bootcamp. The pipeline consists of:

* **Base Image Generation**: Using SDXL to generate a photorealistic image based on an enhanced prompt.
* **Layout Generation**: Utilizing a LLaMA-based model (simulated as "Llama 4 Maverick") to define six text zones in a JSON layout.
* **Poster Rendering**: Applying text, QR code, and social media logos (LinkedIn and X) using PIL, with debug and clean outputs.
* **Directory Cleanup**: Clearing the Kaggle working directory to manage storage constraints.

The pipeline successfully generated edtech\_1024.png (base image), layout.json (text zones), final\_poster\_clean.png (final output), and visualization\_maverick.png (debug output).

**Code Analysis**

Below is a line-by-line breakdown of the Kaggle Notebook (draft (5).ipynb), including commented sections, to provide insights into the implementation and its alignment with project goals.

**Cell 1: Installing Necessary Libraries**

!pip install -U diffusers transformers accelerate scipy ftfy safetensors huggingface\_hub

!pip install groq

!pip install requests

!pip install -U "peft>0.15"

!pip install qrcode

* **Purpose**: Installs dependencies for image generation (diffusers, transformers, accelerate, safetensors), API interaction (groq, requests), fine-tuning (peft), and QR code generation (qrcode).
* **Output**: Successfully upgrades libraries to the latest versions (e.g., diffusers-0.34.0, transformers-4.53.0), though dependency conflicts arise with scipy-1.16.0 (incompatible with gensim, ydata-profiling, etc.).
* **Insight**: The conflicts highlight Kaggle’s environment limitations, as older package versions (e.g., scipy<1.16) are required by other libraries. Paperspace’s customizable environment (Phase 2) can resolve this by allowing isolated dependency management.
* **Phase 2 Relevance**: Dependencies will be re-installed on Paperspace with Python 3.10+, ensuring compatibility with SD 3.5 Large and Streamlit.

**Cell 2: Base Image Generation Using SDXL**

import os

import gc

import torch

from groq import Groq

from diffusers import DiffusionPipeline

from huggingface\_hub import HfFolder, hf\_hub\_download

from transformers import CLIPTokenizerFast

from PIL import Image

import json

import time

import requests

from requests.adapters import HTTPAdapter

from urllib3.util.retry import Retry

* **Imports**: Essential libraries for image generation, API calls, and file handling.
* **Configuration**:
* os.environ['GROQ\_API\_KEY'] = 'gsk\_1ffxCAp2CjMJchP3wBlIWGdyb3FY3WVN9Z0RfD1BbpbxTJGfuqZY'
* client = Groq()
* hf\_token = "hf\_aTyTSImrSyGuqVxHNOlXfEywPgjJTxlVLY"
* HfFolder.save\_token(hf\_token)
  + Sets up API keys for Groq (LLaMA access) and Hugging Face (model downloads).
  + **Insight**: Hardcoding API keys is a security risk; Phase 2 should use environment variables or secrets management.
* **Retry Session**:
* session = requests.Session()
* retry\_strategy = Retry(total=3, backoff\_factor=1, status\_forcelist=[500, 502, 503, 504])
* adapter = HTTPAdapter(max\_retries=retry\_strategy)
* session.mount("https://", adapter)
* session.mount("http://", adapter)
  + Implements retry logic for robust Hugging Face downloads, addressing network instability.
  + **Insight**: Reliable for Kaggle’s flaky internet; Paperspace’s better connectivity may reduce the need for retries.
* **CLIP Tokenizer Loading**:
* def load\_clip\_tokenizer():
* max\_retries = 3
* for attempt in range(max\_retries):
* try:
* print(f"🔄 Attempt {attempt + 1}/{max\_retries} to load CLIP tokenizer...")
* return CLIPTokenizerFast.from\_pretrained(
* "openai/clip-vit-large-patch14",
* cache\_dir="/kaggle/working/cache",
* local\_files\_only=False,
* token=hf\_token,
* timeout=20
* )
* except requests.exceptions.RequestException as e:
* print(f"⚠️ Error loading tokenizer (attempt {attempt + 1}): {e}")
* if attempt == max\_retries - 1:
* raise
* time.sleep(2 \*\* attempt)
* return None
* clip\_tokenizer = load\_clip\_tokenizer()
  + Loads CLIP tokenizer with retry logic to handle network issues, caching to /kaggle/working/cache.
  + **Output**: Successfully loads tokenizer, as shown by progress bars for vocab.json, merges.txt, etc.
  + **Insight**: Caching mitigates Kaggle’s storage limits but requires cleanup (handled in Cell 5). Paperspace’s larger storage will simplify caching.
* **System Prompt**:
* system\_prompt = '''
* You are a world-class prompt engineer for Stable Diffusion...
* '''
  + Defines a detailed prompt for LLaMA to enhance user input into a visual-only prompt under 77 CLIP tokens, ensuring a clean, poster-friendly EdTech scene.
  + **Insight**: The 77-token limit (SDXL constraint) restricts detail; SD 3.5 Large’s 250-token limit in Phase 2 will allow richer prompts.
* **Raw Prompt**:
* raw\_prompt = """
* A cheerful young boy holding a laptop...
* """
  + Describes a boy with a laptop displaying robotics, set against a colorful, tech-inspired background.
  + **Insight**: The prompt is well-crafted but limited by SDXL’s token cap, aligning with the need for SD 3.5 Large.
* **Prompt Enhancement**:
* def enhance\_prompt(user\_prompt):
* max\_retries = 3
* for attempt in range(max\_retries):
* try:
* print(f"🔄 Attempt {attempt + 1}/{max\_retries} to enhance prompt...")
* response = client.chat.completions.create(
* model="llama3-70b-8192",
* messages=[...],
* temperature=0.6,
* max\_tokens=250,
* timeout=20
* )
* return response.choices[0].message.content.strip()
* except Exception as e:
* print(f"⚠️ Error enhancing prompt (attempt {attempt + 1}): {e}")
* if attempt == max\_retries - 1:
* raise
* time.sleep(2 \*\* attempt)
* return ""
  + Enhances the raw prompt using LLaMA 3 (70B), ensuring compliance with SDXL’s constraints.
  + **Output**: Produces a 64-token prompt: "A confident boy in a white shirt and blue jeans holds a laptop with a futuristic robotics design...".
  + **Insight**: Robust retry logic ensures reliability; Phase 2 may use a newer LLaMA model for better prompt quality.
* **Prompt Trimming**:
* def truncate\_to\_clip\_limit(prompt):
* tokens = clip\_tokenizer(prompt)["input\_ids"]
* if len(tokens) > 77:
* trimmed = clip\_tokenizer.decode(tokens[:77], skip\_special\_tokens=True).strip()
* print(f"⚠️ Prompt clipped to 77 tokens (original: {len(tokens)}).")
* return trimmed
* else:
* print(f"✅ Prompt within 77-token limit.")
* return prompt.strip()
  + Ensures the prompt fits within SDXL’s 77-token limit.
  + **Output**: Confirms the enhanced prompt is within limits (64 tokens).
  + **Insight**: Unnecessary for SD 3.5 Large, which supports 250 tokens, simplifying Phase 2.
* **SDXL Model Loading**:
* print("🔄 Loading Stable Diffusion XL...")
* pipe = DiffusionPipeline.from\_pretrained(
* "/kaggle/input/stable-diffusion-xl/pytorch/base-1-0/1",
* torch\_dtype=torch.float16,
* use\_safetensors=True,
* variant="fp16"
* )
* pipe.to("cuda")
* print("✅ SDXL loaded!")
  + Loads SDXL with FP16 precision for GPU efficiency.
  + **Output**: Successfully loads the model, despite CUDA warnings about factory registration.
  + **Insight**: CUDA warnings are benign but indicate Kaggle’s environment quirks; Paperspace’s A5000 GPU will handle SD 3.5 Large more reliably.
* **Image Generation**:
* print("🎨 Generating 1024x1024 base image with SDXL...")
* negative\_prompt = "..."
* image = pipe(
* prompt=final\_prompt,
* negative\_prompt=negative\_prompt,
* num\_inference\_steps=25,
* guidance\_scale=6.5,
* height=1024,
* width=1024,
* generator=torch.manual\_seed(42)
* ).images[0]
* image\_path\_1024 = "/kaggle/working/edtech\_1024.png"
* image.save(image\_path\_1024)
* print(f"✅ 1024x1024 base image saved: {image\_path\_1024}")
  + Generates a 1024x1024 image with a negative prompt to avoid artifacts.
  + **Output**: Saves edtech\_1024.png successfully.
  + **Insight**: The 1024x1024 resolution and 25 inference steps balance quality and speed on Kaggle’s T4 GPU. Phase 2’s SD 3.5 Large will support 2048x2048 for sharper outputs.
* **Cleanup**:
* print("\n🧹 Cleaning up GPU memory...")
* del pipe
* torch.cuda.empty\_cache()
* gc.collect()
* print("✅ All done!")
  + Frees GPU memory to manage Kaggle’s 16GB VRAM limit.
  + **Insight**: Essential for Kaggle’s constraints; Paperspace’s 24GB VRAM will reduce cleanup frequency.

**Cell 3: Architecture Selection for Base Image (Commented)**

# import os

# import base64

# import json

# from PIL import Image, ImageDraw

# from groq import Groq

...

* **Purpose**: Defines a commented-out section for selecting text zone layouts using "Llama 4 Maverick" (simulated via LLaMA 3).
* **Prompts**:
  + asymmetric\_prompt: Specifies 6 text zones with strict constraints (120px distance, 40px buffer, max sizes).
  + adaptive\_prompt: Allows 15px overlaps, 80px gaps, and slightly smaller zones.
* **Insight**: The commented code suggests experimentation with layout strategies. The executed code uses layout.json (from a prior run), indicating the asymmetric layout was chosen. Phase 2 can refine this with dynamic layout generation on Paperspace.
* **Output (from prior run)**: layout.json with 6 zones, used in Cell 4.
* **Phase 2 Relevance**: SD 3.5 Large’s higher resolution may require adjusted zone sizes and spacing.

**Cell 4: Poster Rendering**

def encode\_image\_base64(image\_path):

with open(image\_path, "rb") as img\_file:

return base64.b64encode(img\_file.read()).decode('utf-8')

...

def render\_poster():

...

* **Configuration**:
* API\_KEY = "gsk\_1ffxCAp2CjMJchP3wBlIWGdyb3FY3WVN9Z0RfD1BbpbxTJGfuqZY"
* TARGET\_SIZE = 1024
* base\_image\_path = "/kaggle/working/edtech\_1024.png"
* output\_path = "visualization\_chosen.png"
  + **Insight**: Hardcoded paths and API key repeat Cell 2’s security issue. Phase 2 should use environment variables.
* **Image Encoding**: Converts edtech\_1024.png to base64 for potential API use (not utilized in this run).
* **Font and Logo Paths**:
  + Uses Montserrat, Open Sans, Poppins, and Inter fonts; LinkedIn and X logos from /kaggle/input/data2.
  + **Output**: Confirms all files exist.
  + **Insight**: Font fallback to Montserrat Bold due to missing fonts (e.g., Lato, Merriweather) indicates path issues. Phase 2 must ensure all fonts are available on Paperspace.
* **Text Rendering**:
  + Functions like wrap\_text, get\_text\_color\_for\_zone, and draw\_zone\_content handle dynamic text wrapping, color assignment, and shadow effects.
  + **Output**: Successfully renders 6 zones (e.g., hero\_headline at 49px, benefit\_statement with 4 lines).
  + **Insight**: Font size adjustments (e.g., hero\_headline reduced from 65px to 49px) ensure fit within zones, but Montserrat fallback reduces visual variety. Phase 2 can use specified fonts with proper paths.
* **QR Code and Logos**:
  + Adds QR code at (900, 880) and logos at (30, 880).
  + **Output**: Successfully drawn with debug boxes in visualization\_maverick.png.
  + **Insight**: Hardcoded positions work for 1024x1024 but may need adjustment for 2048x2048 in Phase 2.
* **Output Files**:
  + final\_poster\_clean.png: Clean poster with text, QR code, and logos.
  + visualization\_maverick.png: Debug version with bounding boxes.
  + **Insight**: Dual outputs aid debugging; Phase 2’s Streamlit app can display both for user validation.

**Cell 5: Clearing Kaggle Directory**

import os

import shutil

...

for item in os.listdir(working\_dir):

item\_path = os.path.join(working\_dir, item)

if os.path.isfile(item\_path):

os.remove(item\_path)

print(f"🗑️ Removed file: {item}")

elif os.path.isdir(item\_path):

shutil.rmtree(item\_path)

print(f"🗑️ Removed directory: {item}")

* **Purpose**: Deletes all files and directories in /kaggle/working/ to manage Kaggle’s 20GB storage limit.
* **Output**: Removes final\_poster\_clean.png, visualization\_maverick.png, layout.json, edtech\_1024.png, and cache.
* **Insight**: Necessary for Kaggle but disrupts persistence. Paperspace’s 2TB storage eliminates this need.

**Insights and Limitations**

* **Successes**:
  + The pipeline generates a visually appealing poster with six text zones, QR code, and logos, meeting Gen Z aesthetic requirements.
  + SDXL produces a high-quality base image, enhanced by LLaMA’s prompt engineering.
  + Dynamic text wrapping and debug outputs ensure robustness and ease of troubleshooting.
* **Limitations** (aligned with *Project Deployment Report*):
  + **Token Limit**: SDXL’s 77-token cap restricts prompt complexity, addressed by SD 3.5 Large’s 250-token limit.
  + **Resolution**: 1024x1024 limits scalability; Phase 2’s 2048x2048 support enhances quality.
  + **Kaggle Constraints**: 16GB VRAM, 20GB storage, and 30-hour GPU limit hinder high-volume processing. Paperspace’s A5000 and 2TB storage resolve this.
  + **Font Issues**: Fallback to Montserrat Bold due to missing fonts reduces visual fidelity.
  + **Deployment**: Kaggle’s lack of persistent hosting prevents Streamlit app deployment, necessitating Paperspace.
* **Phase 2 Alignment**: The code’s modular structure (image generation, layout, rendering) supports transition to Paperspace with minimal changes. SD 3.5 Large and Streamlit integration will enhance output quality and enable live deployment.

**Recommendations for Phase 2**

Based on the *Project Deployment Report* and code analysis:

1. **Infrastructure**:
   * Deploy on Paperspace A5000 (24GB VRAM, ~$0.45/hour) with 30GB RAM and 50GB SSD.
   * Upload data2 (fonts, logos) and layout.json to Paperspace storage.
2. **Model Upgrade**:
   * Replace SDXL with SD 3.5 Large (pip install diffusers) for 250-token prompts and 2048x2048 resolution.
   * Update prompt engineering to leverage the increased token limit (e.g., "2048x2048 purple gradient with white overlays").
3. **Streamlit App**:
   * Develop a Streamlit script integrating SD 3.5 Large, PIL rendering, and user inputs for dynamic poster generation.
   * Expose port 8501 via Gradient Deployments for live hosting.
4. **Font Management**:
   * Ensure all fonts (Montserrat, Lato, Merriweather, etc.) are available on Paperspace to avoid fallbacks.
5. **Security**:
   * Replace hardcoded API keys with environment variables or Paperspace secrets.
6. **Testing**:
   * Test rendering with 2048x2048 images and extended prompts.
   * Validate QR code and logo placement for higher resolutions.
7. **Future Integration**:
   * Prepare for website integration via iframe or API, as outlined in the long-term plan.

**Conclusion**

Phase 1 successfully demonstrates a functional poster generation pipeline on Kaggle, producing high-quality outputs despite platform limitations. The code’s modular design and robust error handling provide a strong foundation for Phase 2. Transitioning to Paperspace with SD 3.5 Large and Streamlit will address current constraints, enabling higher-resolution outputs and live deployment. Immediate next steps include setting up a Paperspace instance, installing dependencies, and prototyping the Streamlit app to ensure seamless scalability.