Name: Sk Mamud Haque
B.Tech CSE AIML [Hons] [2021-2025]
Avataar Assignment

# Problem Statement: Dissecting image generation

**Dissecting image generation** involves understanding how deep learning models like **Stable Diffusion** and **GANs** create images from text or other inputs. These models use techniques like **transformers**, **latent spaces**, and **diffusion** processes to generate visually appealing outputs. The problem statement explores the various methods and challenges involved in producing high-quality, photo-realistic images based on textual descriptions, with the ability to control and guide the image generation process through various conditioning inputs, such as **depth maps**, **normal maps**, **or other visual cues**. Applications include art, gaming, product design, and scientific visualization.

**The problem is broken down into several parts:**

a. **Generating High-Quality Images Based on Text and Depth Information:** The goal is to create the highest-quality images possible using text prompts and depth maps. We'll use Stable Diffusion, a leading image generation model, and enhance its output with ControlNet. ControlNet allows us to incorporate additional information, like depth maps, to ensure more accurate and controlled results.
b. **Aspect Ratio Impact on Generation Quality:** To explore how image aspect ratios influence generation outcomes, we'll deviate from Stable Diffusion's standard 1:1 format. By testing wide and tall images (**using the specific file 2_nocrop.png**), we aim to determine if the model can consistently produce high-quality results across various aspect ratios.
c. **Latency and Optimization:** A critical part of the analysis focuses on the time it takes to generate images called generation latency. While image quality is paramount, the speed at which images are generated is also crucial for many practical applications. The goal is to analyze the current pipeline's latency, identify potential bottlenecks, and propose optimizations to reduce generation time without significantly compromising image quality.

## Methodology

A. **Understanding the Metadata:** This given metadata consist of two parts
   a. **Prompts:** Text descriptions that you will use to guide image generation.
   b. **Images:** Depth maps corresponding to the prompts, which act as guidance for the image generation process.
B. **Setup the environment**
   a. **Clone the required repositories**
      1. Clone the Stable Diffusion repository from [Stability-AI/stablediffusion](Stability-AI/stablediffusion).
      2. Clone the ControlNet repository from [lllyasviel/ControlNet](lllyasviel/ControlNet)
   b. **Install necessary packages:** We need to install the packages like torch, diffusers, transformers, PIL, numpy, and opencv-python.
      1. Commands:
         `pip install torch diffusers transformers PIL numpy opencv-python`
      2. For running across any distributed configuration : `pip install accelerate`

## C. Generating Images Based on Metadata

a. **Stable Diffusion Setup:** Load the Stable Diffusion checkpoint using this code

```
from diffusers import StableDiffusionPipeline
import torch
model_id = "runwayml/stable-diffusion-v1-5"
pipe = StableDiffusionPipeline.from_pretrained(model_id).to("cuda")
pipe.set_seed(12345) # Fixed seed
```

b. **Integrating Depth Maps with ControlNet**

```
from diffusers import StableDiffusionControlNetPipeline,
ControlNetModel
controlnet_model                                                    =
ControlNetModel.from_pretrained("controlnet-depthmodel").to("cuda")
controlnet_pipe = StableDiffusionControlNetPipeline.from_pretrained(
 "runwayml/stable-diffusion-v1-5", controlnet=controlnet_model
).to("cuda")
```

c. **Load the depth maps from the metadata folder and generate images**

```
from PIL import Image
depth_map = Image.open("path_to_depth_map.png")
prompt = "Your text prompt from metadata"
output = controlnet_pipe(prompt=prompt, depth_map=depth_map).images[0]
output.save("output_image.png")
```

d. **Combining with Other Forms of Conditioning:** we combine depth maps with normal maps, canny edge maps, etc.

```
import cv2
depth_map = cv2.imread('path_to_depth_map.png', cv2.IMREAD_UNCHANGED)
normals = cv2.Sobel(depth_map, cv2.CV_64F, 1, 0, ksize=5) # Example
normal extraction
```

## D. Generating Images of Different Aspect Ratios

a. **Test Aspect Ratio Generation:** To generate images of different aspect ratios, use the **Metadata/No crop/2_nocrop.png** depth map and experiment with different aspect ratios

```
output = controlnet_pipe(
 prompt="Your text prompt",
 depth_map=Image.open("Metadata/No crop/2_nocrop.png"),
 height=512, width=768 # Example aspect ratio
).images[0]
output.save("aspect_ratio_test.png")
```

## E. Generation Latency

a. **Measure Latency:** We can measure the time taken for image generation using Python's time module

```
import time
start_time = time.time()
output = controlnet_pipe(prompt="Your prompt",
depth_map=depth_map).images[0]
end_time = time.time()
latency = end_time - start_time
print(f"Generation Latency: {latency} seconds")
```

b. **Optimize for Latency**
   1. **Reduce Latency**

```
output = controlnet_pipe(prompt="Your prompt", depth_map=depth_map,
num_inference_steps=30).images[0]
```

## Generation latency

**Generation latency** refers to the time it takes for a model like **Stable Diffusion** (or any image generation model) to produce an image from a given input (e.g., text prompt, depth map). It includes all processing steps, from loading the model and preprocessing inputs to producing the final output image.

For this assignment, you're expected to measure the **total time** the pipeline takes to generate an image. This latency can vary based on several factors:

a. **Hardware**: GPU vs. CPU, the speed and number of cores, available memory.
b. **Model Size**: Larger models, like Stable Diffusion, take longer to load and generate images.
c. **Input Dimensions**: Images with larger resolutions or more complex guidance inputs (like depth maps, normal maps) will have increased latency.
d. **Optimization Techniques**: Efficient memory management, batching, and use of optimized frameworks can reduce latency.