# A Project Report on Image Generation Using Stable Diffusion and Control Net

## Thought Process

The primary goal was to develop an efficient image generation system that uses both text prompts and depth maps for added control over the output images. We used ControlNet with Stable Diffusion to allow better conditioning of the generated images. Created a pipeline for successful execution

Main Considerations:

- Efficiency: Using a pre-trained model saved locally to reduce the time and cost of downloading from external sources.
- Scalability: Writing the code in such a way that allows easy scaling for generating multiple images based on various prompts and conditions.
- Latency Measurement: Added a functionality to calculate and display the time taken for image generation, which is crucial for performance analysis.

## Visual Results

The generated images were saved in the generated_images/ folder(present in the github repo).
Each image corresponds to a prompt and depth map, allowing for both textual and visualization of the output.

Sample images results:

- Prompt: "A luxury bedroom,"
    - Generated Image: A well generated image of a bedroom, showing details like a bed and lights as seen when we google luxury bedroom on google .
    - Edge Map: Canny edge detection applied to the depth map provided an accurate boundary for the image.
- Aspect Ratio Analysis: Images generated with different aspect ratios showed that 1:1 ratio did not provide with the best quality as the image got short and elements were overlapped, while wider ratios like 16:9 led to some loss of image on the corner/edges of images but helped in identifying the features correctly.

# Analysis of Performance

## a. Areas where it will work well:

1. Image Conditioning: Using depth maps and edge detection with ControlNet provides highly detailed images. The additional conditioning ensures that the generated images closely follow the depth information, adding realism to the images.
2. Pre-trained Model Use: Loading the pre-trained model from a local directory significantly reduced the time required to initialize the pipeline.
3. Latency Monitoring: The addition of latency measurement helped assess how long each image took to generate, allowing us to monitor performance and look for optimizations. **On an average each image took around 35 minutes to generate.**
4. Edge Detection: Using Canny edge detection with depth maps adds an extra layer of control to the generated images, ensuring that key features are retained.

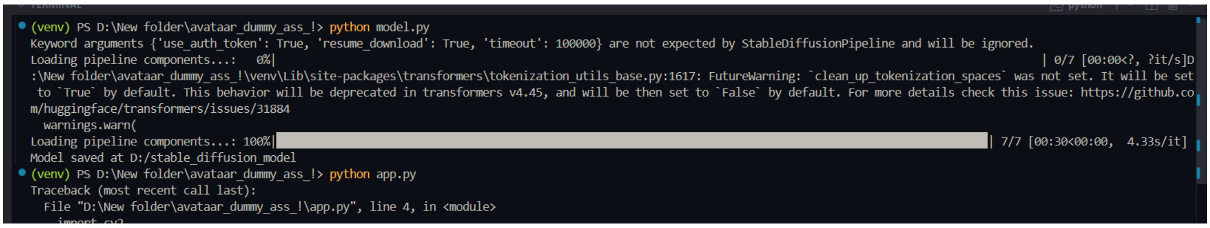
## b. Areas where it will fail:

1. Handling Large Depth Maps: For large depth maps, the process becomes slow, and in some cases, the images fail to generate. The current approach does not handle memory-intensive operations efficiently.
2. Aspect Ratio Distortions: Non-standard aspect ratios lead to distortion in some image areas. While we attempted to preserve image quality, there are visible artifacts in wide or narrow aspect ratio.
3. Reduced Latency: When the latency of an image is less , it will lead to less accurate response and sometimes no image generation will take place.

# Ideas for Improvement

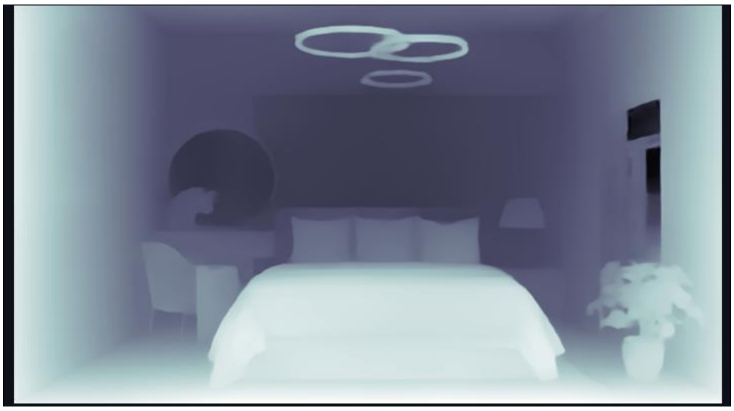
1. Memory Optimization: One idea is to resize depth maps dynamically before processing, reducing the load on the GPU.
2. Better Aspect Ratio Handling: Implement padding techniques to maintain image quality across all aspect ratios without distortion.
3. Edge Detection Enhancement: Explore using more advanced edge detection techniques (e.g., Sobel, Laplacian) that may work better with different types of depth maps, especially when the depth data is noisy or incomplete.
4. Batch Processing: Implement a batch processing pipeline to generate images more efficiently for multiple prompts, reducing the time taken by individual function calls
5. Latency Improvement: We can significantly improve the latency of the images upto 20-50% by using limiting the output length, caching frequently and saving the responses in the memory so that if the image is generated again it will be fater.

## Some Output Images




Downloading the model from hugging face and extracting pipeline components.


Model Training and images being generated



Aspect ratio (1:1)

Aspect Ratio (16:9)