BABEŞ BOLYAI UNIVERSITY, CLUJ NAPOCA, ROMÂNIA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Indoor Room Generation using Stable Diffusion

– MIRPR report –

Team members
Rastoaca Nicolae Sergiu, IR, 236, email
Savu Daria Maria, IR, 236, email
Stan Adrian Valentin, IR, 236, email

2024-2025

Abstract

Text of abstract. Short info about:

- project's main idea and relevance/importance,
- inteligent methods used for solving,
- data involved in the numerical experiments;
- conclude by the the results obtained.

Please add a graphical abstract of your work.

Contents

Chapter 1

Introduction

1.1 What? Why? How?

As AI technology develops more and more every year, the task of acquiring large datasets in order for those AI models to achieve their full potential becomes more and more important. A difficult task nowadays is indoor room semantic segmentation and depth estimation- the ability to correctly identify objects in a room and estimate the spacial coordinates relative to each other. For that task to be completed by AI models, large and varied datasets are needed. Obtaining those datasets is costly and challenging thus it is more efficient to have other AI models generating the datasets. There are models used for indoor room generation such as Interior design control network (IDCN) but the input is limited to text only. It would be way easier and more efficent for users to be able to also give a sketch as an input in addition to the textual input. This is what we aim to do: take the input in both image and text form and fine tune ControlNet to generate images that adhere to the given input specifications. The benefit of this method is that it can also be used by interior designers when working with a client in order to save time and make sure their ideas align with the client's expectations.

1.2 Paper structure and original contribution(s)

Chapter 2

Scientific Problem

2.1 Problem definition

New AI models require large datasets to reach their full potential. The problem we are addressing is the generation of realistic synthetic images depicting scenes from indoor environments to improve semantic segmentation and depth estimation tasks in these settings. The importance of this problem lies in the potential to reduce reliance on real data, which is difficult and costly to obtain in large quantities. Additionally, the ability to introduce variations in these generated data provides AI models with datasets that allow them to learn from multiple scenarios.

While there are models such as IDCN which use Stable Diffusion with Conditional Control to generate indoor images [2], the model is limited to certain styles by the dataset used. For the images to be used in training of other AI models, they have to be varied and contain different styles.
This is why we aim to fine tune ControlNet on a large dataset (NYU Depth Dataset V2) and take input in both text and image form. In this way we obtain detailed images while also staying true to the segmentation and depth of the desired design.

Input: A paragraph which describes an interior design scene and/or a PNG or JPG file which represents a sketch of the desired output.
Output: A list of RGB images which fulfill the textual descriprion and the given sketch.

Chapter 3

State of the art/Related work

One of the solutions previously used for image generating are the Generative Adversarial Networks (GAN). They use a discriminator network as an objective for the generator. The generator creates synthetic data from random variables, and the discriminator distinguishes between synthetic and real data. The generator's goal is to produce data that the discriminator cannot differentiate from real-world data, while the discriminator strives to minimize its classification error.[3]. GANs efficiently generate high-resolution images with good quality but are difficult to optimize and struggle to capture the full data distribution.[4]

Some other options are Variational autoencoders (VAE) and autoregressive models (ARM). VAE which provide an efficient synthesis of high resolution images the but sample quality is not as good as that of GANs, while ARMs achieve a good performance in density estimation, but are limited to low resolution images due to their computationally demanding architectures and sequential sampling process.[4]

More recently, Diffiuson Models have been introduced which work with a forward and backword diffusion process. During the forward diffusion process, Gaussian noise is added gradualy over time. In the reverse diffusion process, a neural network is trained to predict the noise added at each step and then the data has the noise removed over a series of time steps. The model is trained to minimize the difference between the predicted noise and the actual noise added during the forward process.[1]. The problem with diffusion models comes from optimizing and evaluating them in the pixel space because of low inference speed and very high training costs.[4]

Latent Diffusion Models act similarly to Diffusion Models, but by operating in a latent space, they solve the high cost problem of diffusion models and allow for high resolution image generation [4].Stable

Diffusion, an implementation of a Latent Diffusion Model, has an architecture of 25 blocks in total, including 12 encoder and decoder blocks, with each encoder and decoder having down-sampling and up-sampling layers.

ControlNet improves Stable Diffusion (an implementation of a Latent Diffusion model) by adding conditional controls to its architecture. [5] It creates a trainable copy of the 12 encoder blocks and the middle block, which bridges the encoder and decoder, of the Stable Diffusion model. Then it adds trainable layers alongside the copied ones(the locked blocks). Zero convolution layers connect the trainable components to the locked blocks, enabling conditional control while retaining the pretrained model's capabilities.[5]

Interior design control network (IDCN) is the result of a method which connects a small neural network to a pre-trained diffusion model. The first part of the method is the locked copy, where the parameters of the pre-trained diffusion model are directly used and the second part is the trainable copy, which inherits the parameters of the locked copy and supports continued training to learn new knowledge. The locked copy and the trainable copy are connected through a unique convolutional layer called zero convolution to generate the final results.[2]

Chapter 4

Investigated approach

4.1 Training on the small dataset

The first step in the developing the interior design image generating model is experimenting and better understanding ControlNet. This is why we trained it on a smaller dataset, with a lower number of epochs. To train the model you need a set of: control images (the sources), text prompts and target images. From the text input and control image we want to obtain the target image. The dataset used is one with different colored circles. The model is trained to generate images of circles where the text prompts indicates the colors of the circle and the background and the control image indicates the position of the circle. The full dataset consists of 50000 inputs (control image, target image and prompt) which is quite large as well. Because of computational limitations at this stage, we decided to only use 1000 of these inputs.
The initial shape of the images is : (512, 512, 3)

A prompt example: brown circle with salmon background

Control image: The edges as shown in Fig 4.1

Ground truth/target image: A colored circle positioned on the colored background as shown in Fig 4.2 TODO: **It would be nice to detail more the inputs. In addition to the prompt, did you use some conditions (the edges or the segmentation mask)?**

After loading the dataset, we need to copy the weights of a stable diffusion model (we used SD2 `https://huggingface.co/stabilityai/stable-diffusion-2-1-base/tree/main`) and add a control net to that architecture. The next step is to actually train the model. We tried different sets of parameters to try and optimize the training time while not losing too much performance. Due to the size of the model and large dataset the training has to be done on GPUs. We tried using one batch due to low VRAM but it took to long to run. We used the default batch size of 4, but lowered

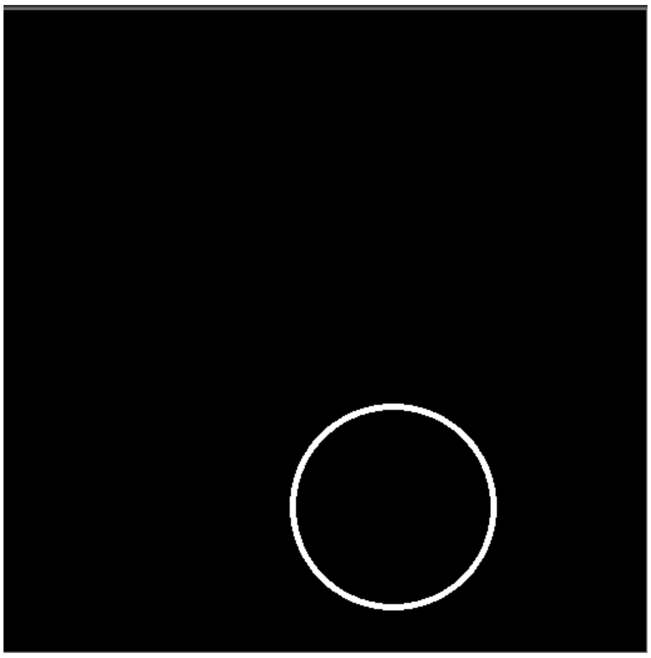

Figure 4.1: Circle edge used as control image

the numerical precision in training from 32bit float to 16bit float. Considering the small dataset, we decided to run on 10 epochs to avoid overfitting. We measured the performance using FID (Frechet Inception Distance). To be able to compute FID, we took the generated images and the target images and separated them in two folders. We resized them to (299,299,3) and using the inception_v3 model, computed the FID, obtaining a value of 3.664 which is very good considering the small dataset and number of epochs.

Generated image in the first epoch of training: Fig 4.3

Generated circle image in the last epoch of training: Fig 4.4

Table 4.1: Results and params for small data

Results			
Image shape	**Learning Rate**	**Epochs**	**FID**
(512, 512, 3)	1e-5	10	3.664

4.2 Training on the NYU Depth V2 dataset

The NYU Depth V2 dataset consists of 1449 images of indoor rooms which have the ground truth image, the semantic segmentation and the depth estimation images. The ground truth image are RBG images with real life rooms while the depth estimation and sematic segmentation are grey scale

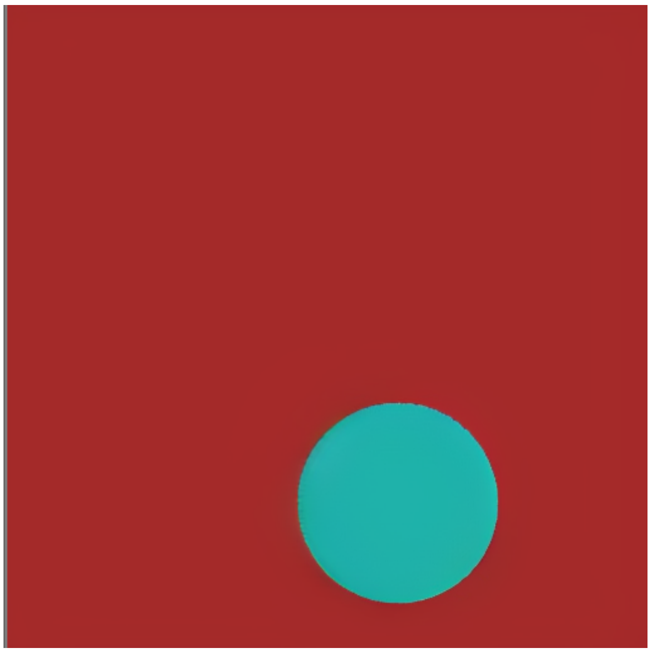

Figure 4.2: Ground truth image for circles dataset

images with the extracted features from the original image. In our training we only use the semantic segmentation and original image - view Fig 4.7 We decided to split the dataset into training and test samples as follows:

1. First 100 sets are used for testing
2. The other 1349 sets are used for training

TODO: **please add a short description of the dataset used in these scenarios (the "main dataset") - a quantitative analysis of the data could be useful**

We have 2 approaches:

4.2.1 Fine tuning the semantic segmentation model from control net

We took the semantic segmentation pre-trained model and used the dataset to fine tune it. The pre-trained model uses sd1.5. The segmentation mask of each image was used as a control, the real image was the ground truth and since the dataset did not have any textual description for the images we used a generic prompt: "An indoors design". We saved the weights to a new file and then loaded the model and tested on a sample of 100 images from the initial dataset. In the testing process the ground truth control and text prompt remained the same as during training. With the predicted images we computed the FID.The summary for this training can be viewd in Fig 4.2

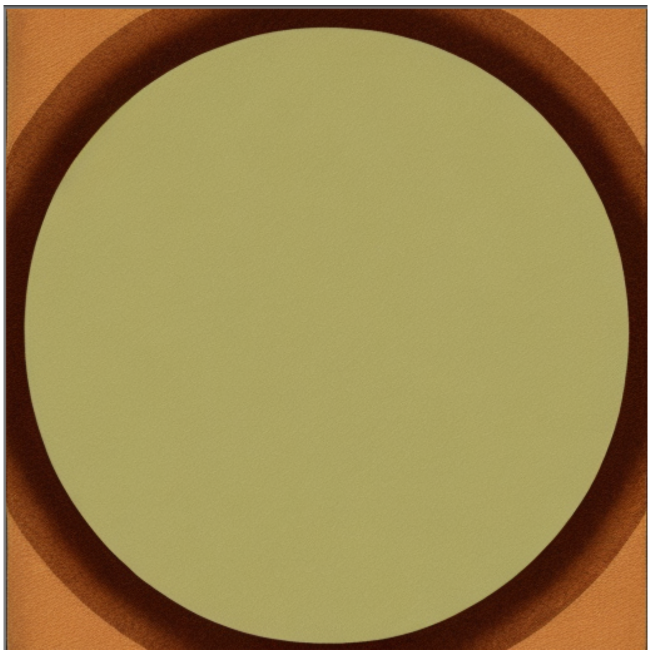

Figure 4.3: Circle generated in the first epoch of training

TODO: **please give more details about the training setup and the dataset used for this fine-tuning stage. Did you use the segmentation mask as a condition here? What about the prompt used?**

Table 4.2: Results and parameters for semantic segmentation pre-trained model

Results					
Image Shape	**Learning Rate**	**Epochs**	**Batch Size**	**FID**	**Loss**
(256, 256, 3)	1e-5	30	4	5.211	Refer to Figure 4.8
(256, 256, 3)	1e-4	15	32	4.050	Refer to Figure 4.9
(256, 256, 3)	1e-5	12	64	2.743	Refer to Figure 4.10
(256, 256, 3)	1e-3	15	64	15.371	Refer to Figure 4.11
(128, 128, 3)	1e-5	10	128	1.402	Refer to Figure 4.12

4.2.2 Fine tuning the canny edge model from control net

We extracted the edges from the real life images in the dataset. In training, the canny image became the control, the ground truth was the original image and the prompt was generic: "An interior design".

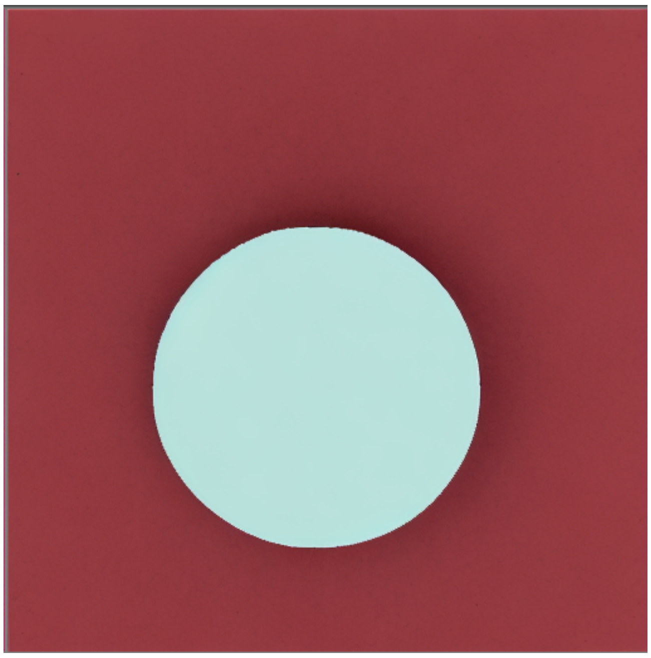
Figure 4.4: Circle generated in the last epoch of training

The inital weights were those loaded from the canny pre-trained model available on control net and the updated weights were save to a new file. We used the new weights for testing where we selected 100 images from the dataset and used the same prompt control and ground truth as in training. View example for ground truth and test images in Fig 4.17

TODO: **please give more details about the training setup and the dataset used for this fine-tuning stage. Did you use the segmentation mask as a condition here? What about the prompt used?**

Table 4.3: Results and parameters for canny edges pre-trained model

Results					
Image Shape	**Learning Rate**	**Epochs**	**Batch Size**	**FID**	**Loss**
(256, 256, 3)	1e-5	30	4	12.897	Refer to Figure 4.18
(256, 256, 3)	1e-5	12	32	7.967	Refer to Figure 4.19

TODO: **I recommend adding some details (table/plot with the loss, other performance measures, training time, etc) about the training process. Some visual results could help, also.**

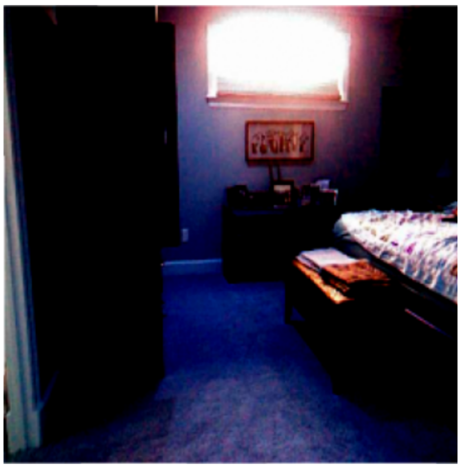

Figure 4.5: Ground Truth RGB Image

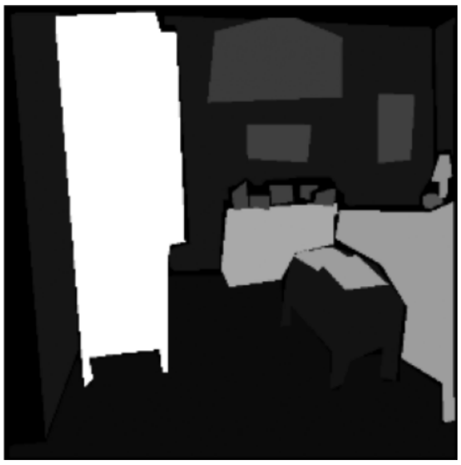

Figure 4.6: Semantic Segmentation Mask

Figure 4.7: Examples from the NYU Depth V2 dataset: (a) Ground Truth RGB Image, (b) Corresponding Semantic Segmentation Mask.

4.3 App's description and the main functionalities

We implemented a web application that takes two inputs from the user: a paragraph describing what the desired images should look like and a PNG or JPG file representing the sketch that the generated images should follow.

After the Generate Images button is pressed, the AI generates a few RGB images and they are rendered in the app.

4.4 App's design

- use cases

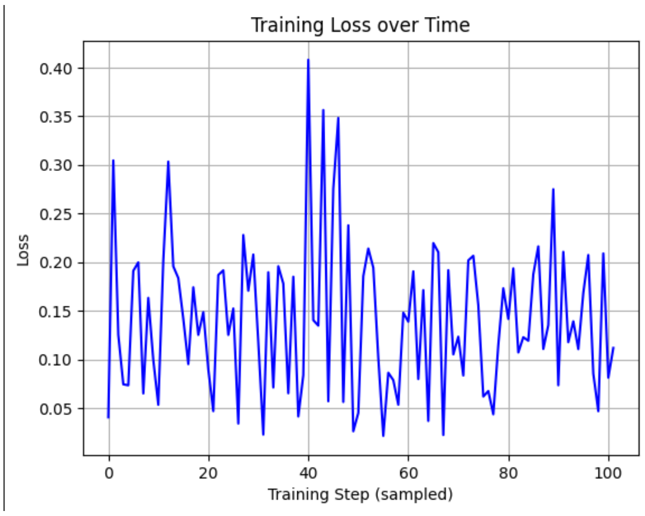

Figure 4.8: Loss plot for the semantic segmentation model- batch size 4 and number of epochs 30

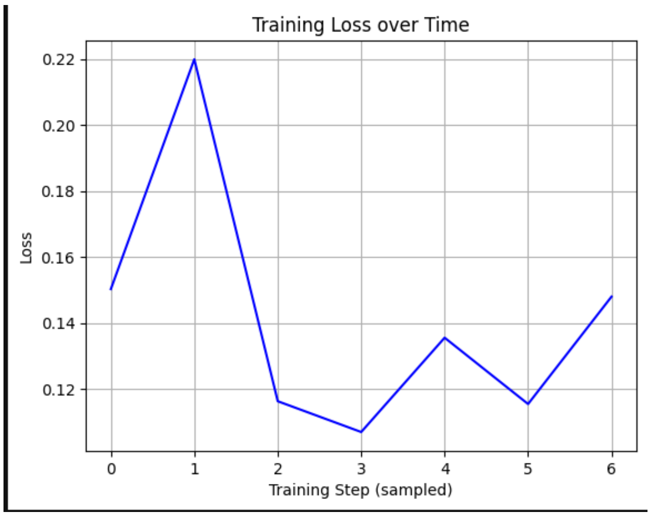

Figure 4.9: Loss plot for the semantic segmentation model- batch size 32 and number of epochs 15

- diagrams (class diagram, sequence diagram, database structure)

4.5 Implementation

4.5.1 Frontend Implementation

Technology Used: React.js **Purpose:**

- Provides a user-friendly interface for interacting with the application.
- Collects user inputs (text description and/or sketch image) and displays the generated images.

Key Features:

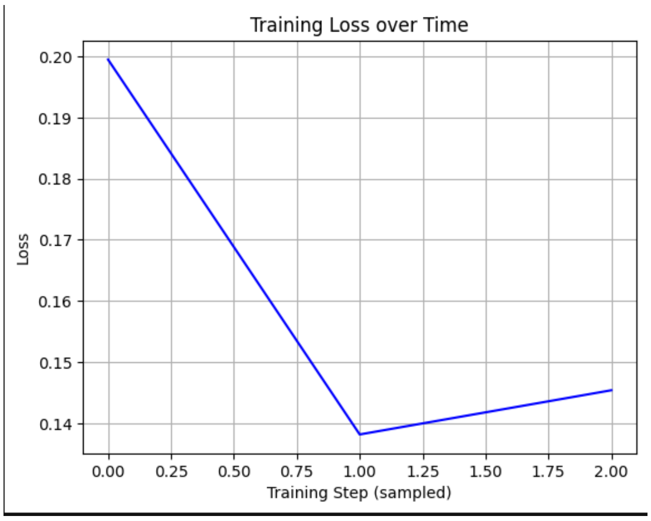

Figure 4.10: Loss plot for the semantic segmentation model- batch size 64 and number of epochs 12

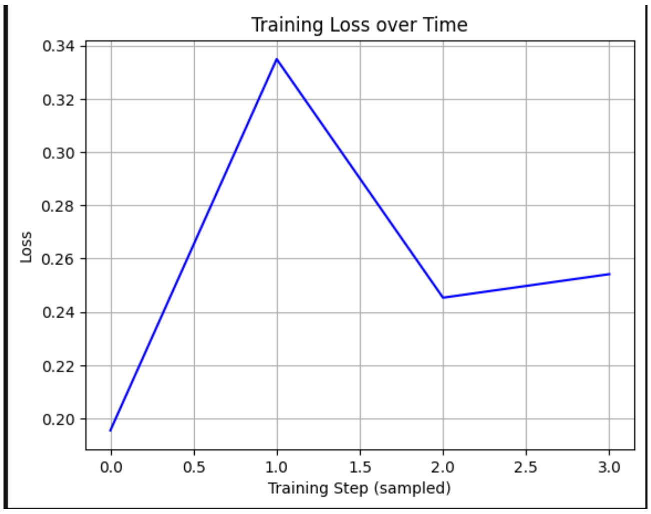

Figure 4.11: Loss plot for the semantic segmentation model- batch size 64 and number of epochs 12- with larger learning rate

- File Upload: Allows users to upload sketches in PNG or JPG format.
- Text Input Field: Users can describe the desired scene for generation.
- Image Rendering: Displays the output RGB images generated by the AI model.
- Dynamic Interface: React's state management ensures seamless updates without refreshing the page.

Notable Libraries and Tools:

- Axios: For communicating with the backend API.
- Material-UI / Bootstrap: For styling and UI components (if applicable).

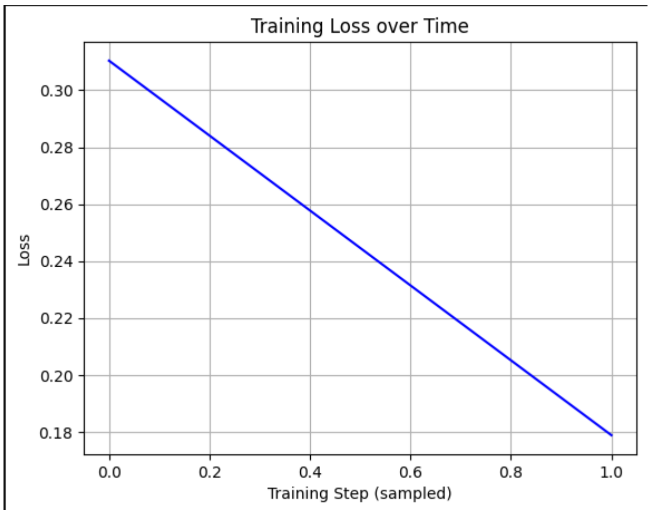

Figure 4.12: Loss plot for the semantic segmentation model- batch size 128 and number of epochs 10

- React Router: Enables navigation between pages (if the app has multiple pages).

Special Settings:

- The app is configured to handle image uploads using an `<input type="file">` element.
- API requests are sent to the backend with JSON payloads for the text description and multipart data for the image.

4.5.2 Backend Implementation

Technology Used: Python **Framework:** Flask **Purpose:**

- Handles the processing of inputs and generates images using AI models.
- Serves as the bridge between the frontend and the AI model.

Key Features:

- API Endpoints:
  - /process: Accepts POST requests with a text description and/or uploaded image.
  - Returns a generated RGB image as a JSON response
- AI Model Integration:
  - Utilizes a fine-tuned ControlNet model to generate high-quality synthetic images.
  - Loads the model weights during initialization for efficient inference.

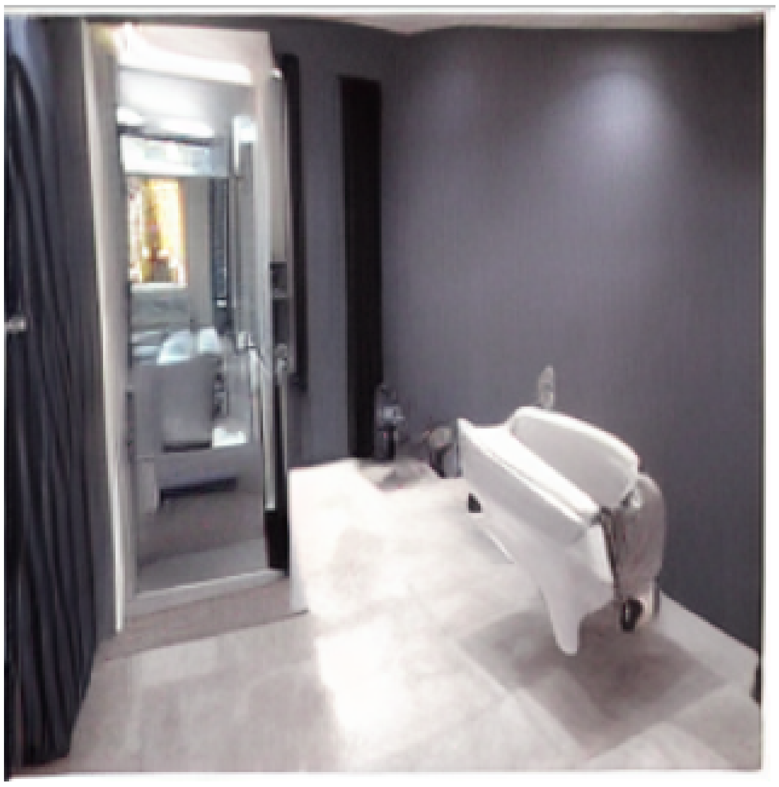

Figure 4.13: Generated image from the testing session with the 64 batch size semantic segmentation model

- Input Preprocessing:
  - Handles the uploaded sketch by extracting and validating the image format.
  - Processes the text input to match the format required by the AI model.
- Output Formatting:
  - Converts the AI-generated images into a suitable format: base64-encoded strings for frontend rendering.

Notable Libraries and Tools:

- Torch: For loading and running the AI model.
- Pillow: For image preprocessing and validation.
- NumPy: For efficient data handling during image processing.

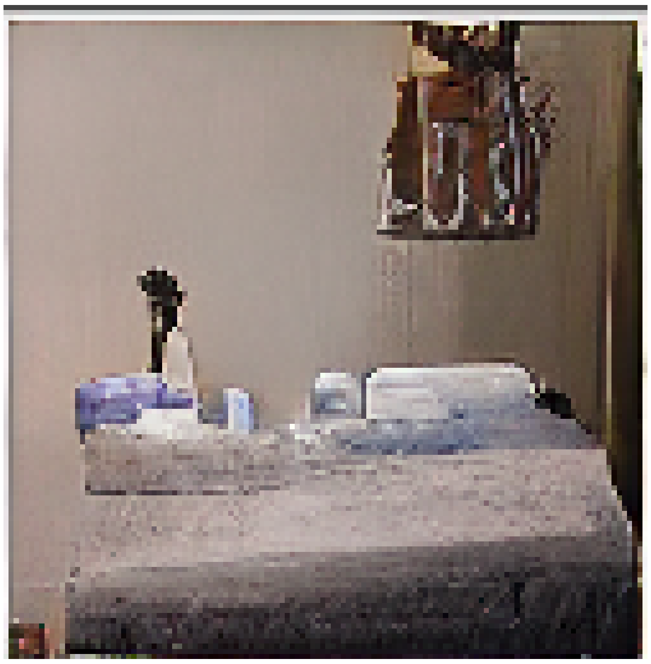

Figure 4.14: Generated image from the testing session with the 128 batch size semantic segmentation model

4.6 Numerical validation

The numerical validation in our application consists of measuring the FID (Frechet Inception Distance) for each of our trained models on a specific dataset for testing. The dataset used for testing consists of 100 images from the 1449 of the total images in the NYU Depth V2 Dataset that have been specifically kept for testing. The results for this metric can be viewd in Table 4.2 for the semantic segmentation training and in Table 4.3 for the canny edges training.

4.6.1 How FID Works

Feature Extraction: Both real and generated images are passed through a pre-trained network, commonly the Inception v3 model, to extract feature representations. These features capture high-level details of the images.

Statistical Comparison:

- The extracted features from each dataset are modeled as multivariate Gaussian distributions.
- The FID measures the distance between these two distributions using the FrÃ©chet distance formula:

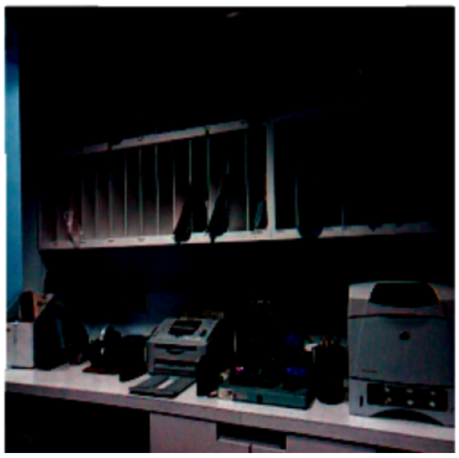

Figure 4.15: Ground Truth RGB Image

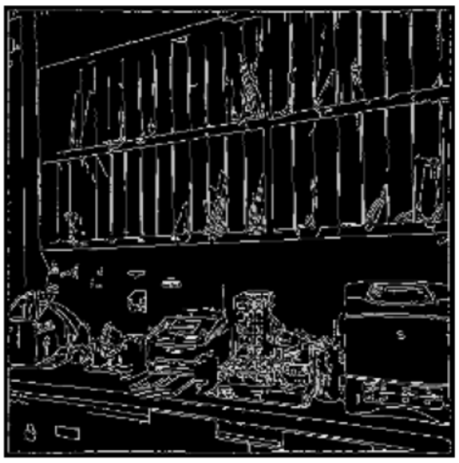

Figure 4.16: Canny Edges Mask

Figure 4.17: Examples from the NYU Depth V2 dataset: (a) Ground Truth RGB Image, (b) Corresponding Canny Edges Mask.

$$\mathrm{FID} = \|\mu_r - \mu_g\|^2 + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r\Sigma_g\right)^{1/2}\right)$$

where:

- μ_r, μ_g: Mean vectors of the real and generated feature distributions.
- Σ_r, Σ_g: Covariance matrices of the real and generated feature distributions.
- Tr: Trace of a matrix.

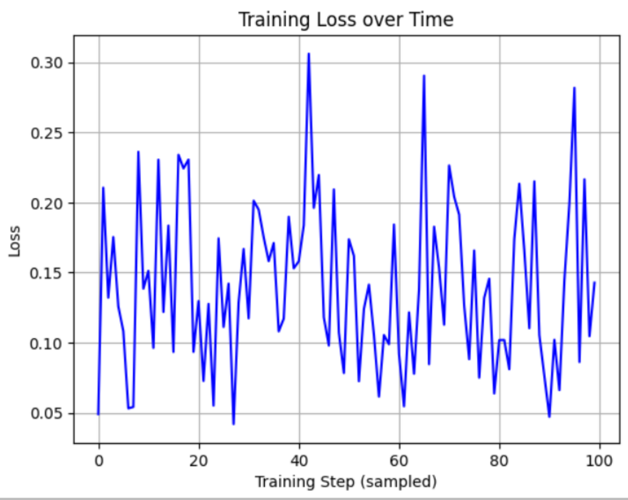

Figure 4.18: Loss plot for the canny edge model training- batch size 4

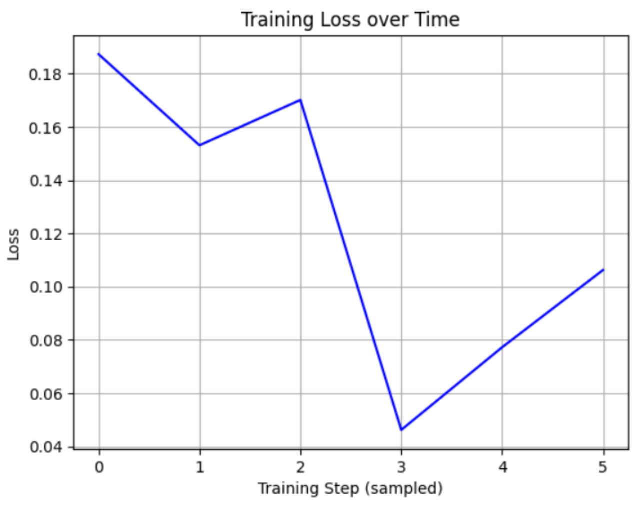

Figure 4.19: Loss plot for the canny edge model training- batch size 32

4.6.2 Interpreting FID:

- **Lower FID values** indicate greater similarity between the real and generated images, suggesting better performance of the generative model.
- **Higher FID values** suggest more significant differences.

4.6.3 Methodology

Because generated images are generally harder to evaluate we used two criteria:

- The loss function plot- a loss function that consistently decreases indicates a model which has improved

- The numerical value of the FID computed on the testing samples- a lower computed value indicates a better similarity between the generated image and the ground truth one

Through this experiment we wanted to evaluate how well a stable diffusion model with added control blocks could generate realistic images of interior design scenes. The dataset we used consisted of real life images with various scenes of interior rooms with images having various lightning levels and being taken from different angles. It is a varied and challenging dataset. To find the best results we tested different values for the parameters and observed the following:

4.6.3.1 Sematic segmentation vs Canny edges

We compared two different types of images used as controls for training and testing: the semantic segmentation mask and the canny edges of the original image. As noted in the values shown in Table 4.2 and Table 4.3 the semantic segmentation models obtained better FID values and had better loss functions. Thus we decided to further experiment with the values of the hyperparameters for the semantic segmentation training.

4.6.3.2 Influence of hyperparameters

1. Learning rate- we did not change the learning rate much, but we did experiment with quite a large one: 1e-3 which generated an extremely unstable model

2. Number of epochs- initially we had a very high number of epochs-30 and it was clear that the model was overfit. We then decreased it and experimented with values between 10 and 15.

3. Image Resolution- we kept the image resolution reasonably low- 256 * 256 for most training simulations to find a balance between performance and being usable in a real-life application. We did a training simulation with a lower resolution of 128*128 and alltough the results were the best both in terms of loss and FID, the small resolution makes it harder to integrate in an application. However, the smaller resolution considerably reduces the training and generating times.

4. Batch size- we initially trained with a very small batch size due to video memory restrictions and obtained a zigzag pattern in the loss function. As we gradually increased the batchsize and lowered the number of epochs we observed an improvement both in the loss function and FID values. Moreover, larger batch sizes meant quicker training, but we were limited by the video memory limit.

4.6.4 Results

Throughout our testing process we had two promising models in terms of results:

- The model trained with image resolution of 256 * 256, batch size 64, learning rate of 1e-5 and 12 epochs which obtained an FID of 2.743
- The model trained with image resolution if 128 * 128, batch size of 128, learning rate of 1e-5 and 10 epochs which obtained an FID of 1.402

We then had to decide between a better performance but smaller resolution and a better resolution and slightly worse performance. Considering the fact we will be using the model in an application, it would be better for the user to have images at a higher resolution thus we decided to use the slightly worse performing model with a higher image resolution.

4.6.5 Discussion

We have successfully proven that we can use stable diffusion models with control blocks to generate realistic images of interior rooms. The advantages of this method lies in the quality of the images and their resemblence to real life ones, while the disadvantages are made up of the considerable resouces needed to train and test the models.

Chapter 5

Conclusion and future work

All in all, the experiment was successful and it has room to be developed further. The usage of stable diffusion models with added control blocks can be expanded in various domains including the generation of training images for other purposes. The NYU Depth V2 Dataset is quite small but with a larger dataset and more perfromant GPUs the results obtained could be impressive.

Bibliography

[1] Md Manjurul Ahsan, Shivakumar Raman, Yingtao Liu, and Zahed Siddique. A comprehensive survey on diffusion models and their applications. *arXiv preprint*, 2024.

[2] Junming Chen, Xiaodong Zheng, Zichun Shao, Mengchao Ruan, Huiting Li, Dong Zheng, and Yanyan Liang. Creative interior design matching the indoor structure generated through diffusion model with an improved control network. *Journal of Design and Technology*, 2024.

[3] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in Neural Information Processing Systems*, 30:6626–6637, 2017.

[4] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *arXiv preprint arXiv:2112.10752*, 2022.

[5] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. *arXiv preprint arXiv:2309.02248*, 2023.