Machine Learning Challenge

Applicant: Shayari Bhattacharjee, shayarib211196@gmail.com

Introduction

In this section, we deal with the open-source project: ControlNet Zhang et al. (2023), which can be considered as a advanced version of Stable diffusion which not only accepts text prompts but also has the capability to accept image as input. To understand the functioning of ControlNet, let us first dive into some basic concepts:

Stable Diffusion

Stable diffusion is one of the most popular AI generative methods for generating high-quality images from text prompts. This concept can also be used to modify images by providing text+image prompts. It was originally proposed by Rombach et al. (2021).

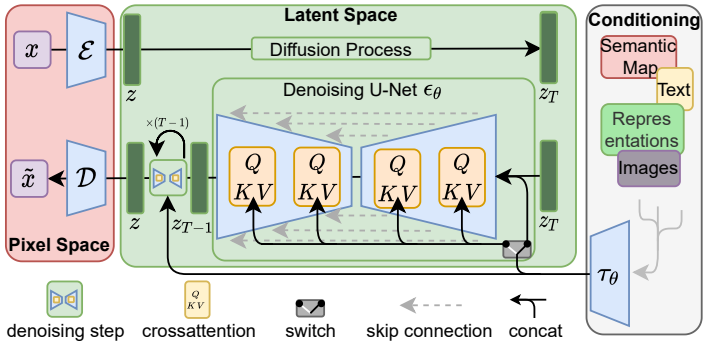

Figure 1: Stable Diffusion

The stable diffusion process consists of three subcomponents: The **Pixel Space**, where the original input image and the final generated image exists, The **Latent Space** where the diffusion and denoising process takes place and finally **Conditioning Space**, where prompts in the form of semantic maps, text, representations and images exists which are fed back to the system in the de-noising process.

How it works? The input image passed through the encoder $\mathcal{E}$ which converts into a latent representation z which then undergoes a diffusion process in which noise is added to produce z_T in multiple steps. For the denoising process, prompt input(τ_θ) is taken from the conditioning space and the noised latent representation(z_T) is denoised through U-Net architecture(ε_θ) with Cross-Attention Blocks (Q, K, V) that allow the model to incorporate external conditioning inputs, such as text or other images and skip connections which helps in preserving fine details by linking early layers to later layers. It also comprises of switch and concatenation layer to combine different feature representations efficiently. This process removes noise step-by-step in a guided manner to finally achieve latent representation z which passes through the decoder($\mathscr{D}$) to get final generated image.

ControlNet

ControlNet can be considered an advanced version of stable diffusion process which controls diffusion models by adding extra conditions. ControlNet achieves this control by copying the weights of the neural network blocks into "locked" and "trainable" copies. The locked copies do not participate in the training process and the original model remains unaltered. Whereas, the trainable copy gets altered. It also consists of special **1x1 convolutions(Zero convolutions)**, initialized with zeros, prevent any distortion of the model before training but causes distortion during training as $\frac{\delta_y}{\delta_x} \neq 0$ when $y = wx + b$, so it behaves as a normal convolutional layer during training.

Advantages of this architecture are that it is memory efficient due to fewer gradients, it uses the concept of fine-tuning while keeping the original model safe. This allows training on small-scale or even personal devices and is also friendly to merge/replacement of models/weights/blocks/layers. It also consists of "Guess Mode" where ControlNet encoder will try best to recognize the content of the input control map, like depth

map, edge map, scribbles, etc, even if you remove all prompts.

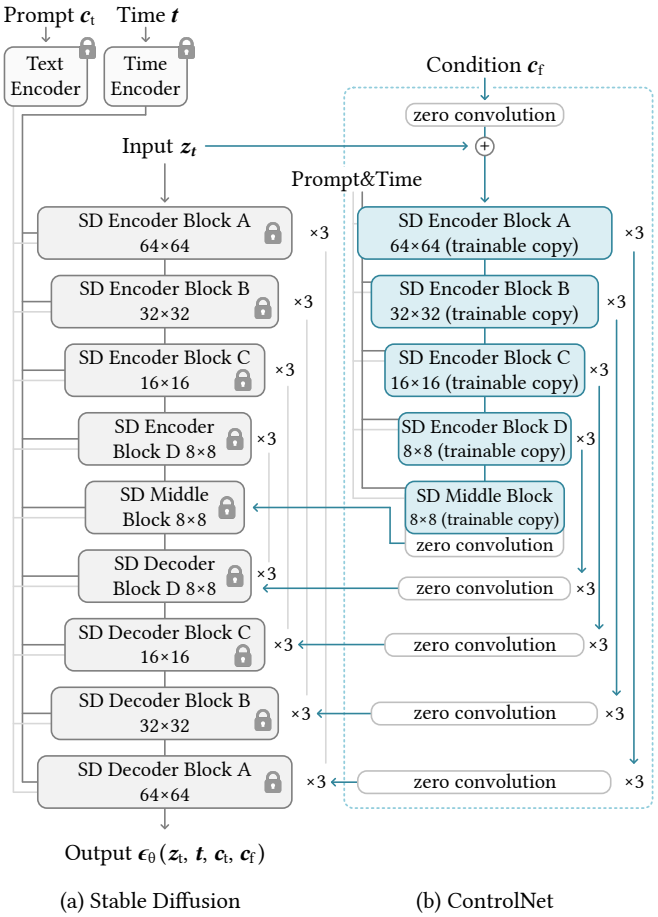

Figure 2: ControlNet

ControlNet can be used for multiple applications as follows:

- *Canny Edge Detection*: Generates images based on the edges detected in an input image. Good for creating stylized artwork or reproducing the structure of an object.
- *Hough lines*: Generates images based on straight lines detected in an image.
- *HED Boundary Detection*: Uses a "soft" boundary detector to preserve details, making it good for recoloring and stylizing images.
- *User Scribbles*: Allows you to guide image generation with hand-drawn scribbles.
- *Semantic Segmentation*: Uses semantic segmentation maps to control the objects and regions in the generated image.
- *Fake Scribbles*: Synthesizes scribbles automatically from an input image.
- *Human Pose*: Generates images with specific human poses, extracted from an input image or pose data.
- *Depth Maps*: Generates images consistent with a given depth map, controlling the 3D structure of the scene.
- *Normal Maps*: Generates images based on normal maps, which can be good for preserving geometric details.
- *Anime Line Drawing*: A specific ControlNet trained for generating anime-style line drawings.

Task 1: Use-case Analysis

Challenges

Some of the challenges that can be encountered are as follows:

- *Gradio Customization*: Currently the codebase uses gradio to create a UI for the tasks. However, some tasks require a better UI for better results as Gradio is considerably difficult to customize.
- *Dependencies*: ControlNet needs other detector models such as HED edge detection model, Midas depth estimation model, Openpose, SD2 depth model, etc, which on one hand can make it quite flexible but cause inferior results and dependencies mismatch leading to installation or runtime errors.
- *Security and copyright concerns*: Security issues of deploying GenAI methods and its copyright issues is quite a important concern Golda et al. (2024). It can also be used for misleading content creation such as DeepFakes.
- *Computational Resources*: Models used in ControlNet have high memory storage requirement (>5GB) and retraining/fine-tuning might require powerful expensive GPUs as DiffusionWrapper has 859.52M parameters with 512 input channels with shape (1,4,32,32)=4096 dimensions.
- *Model choice*: Choosing models for tasks such as semantic segmentation, pose detection, etc. and prompts could be challenging and time-consuming and will vary considering the use case.
- *Integration and Compatibility issues*: As ControlNet relies on multiple packages, there are high chances of package mismatch and missing dependencies. For GPU support, CUDA or CuDNN needs to set up correctly, requiring users to debug libtorch_cuda.so issues in case of runtime errors.
- *Parameter Selection*: ControlNet has several key parameters (e.g., conditioning strength, guidance scale, choosing prompts(User and default), ddim steps, seeds, thresholds and lastly, the user-defined image prompt) that significantly affect output quality can seem tedious to set-up.
- *Lack of generalization*: Pre-trained models might be susceptible to bias or overfitting to training data and might not generalize well for unseen test data.

Applications in Zeiss

Some of the application areas in Zeiss were ControlNet can be utillised are as follows:

- *Medical Imaging*: can be used to create synthetic datasets or to augment a dataset for better training. Canny edges, segmentation masks, and normal maps can be suitable for adding further details and different modalities.
- *Semiconductor Industry*: Similar to medical imaging, canny edges, segmentation masks, synthetic images and normal maps generated synthetic images with a different scope of product inspection which makes sure that the produced items maintain certain quality and standards.
- *Futuristic lens consultation*: Being pioneers in optical industry, Controlnet generated images can be used to train AI models to train recommendation systems for perfect spectacles fit without collecting samples which is tedious, expensive and time-consuming.
- *Robotics and Automation*: Pose estimation, depth estimation, and canny edges which are utilised by ControlNet architecture to generate images can be suitable for human-robot collaboration by tracking human activities and visual inspection of products by visualizing depth maps and canny edges to maintain quality standards.

- *Research and Development*:Using generative AI tools to generate images for training and fine-tuning can save resources compared to training from scratch.
- *Rapid Prototyping*: Fake/User Scribbles can be used for rapid prototyping by converting scribbles to high-quality images which can serve as visual prototypes for new products and development of Proof-of-concept(POCs).

Task 2: Process Planning

In this section, we focus on model training and feature development.

Overall Code Architecture

Available resources:

- Modular repository structure.
- Tutorials, models, gradio and README.md are present.
- Utilises pre-existing training models efficiently.
- Easier reproducibility using environment.yaml with instructios present in README.md.

Missing resources:

- Currently, models are split between annotator, cldm, ldm and models folders. And in some use-case like canny-edge detector, hed detector etc in annotator directory are missing. In general, directory structure is not well managed.
- Absence of Experiment tracking and CI/CD pipelines.
- Data and model version control is missing.
- Reproducibility concerns due to non-standardized environment setup.Missing docker container.
- README.md is not well organised and required refining.
- tutorial.train.py can be refactored into a proper training pipeline. No support for different dataset formats.

Scope of Improvement:

- All gradio notebooks can be moved to a single directory for better organisation. Tool and tutorial scripts can have their designated directories.
- The README.md can be refined by adding further details on training and highlighting the advantages of using ControlNet by sharing code snippets and evaluation results rather than only infrastructure image.
- Better directory structure with having seperate directories for training, gradio notebooks, tutorials.
- Implementation of experiment tracking(Tensorboard etc) and CI/CD pipeline(Gitlab action with testing, linting and deployment).
- Using libraries like hydra or pydantic settings to manage configuration which are currently hardcoded in YAML files in ControlNet/models/ directory. This allows for easy experiment configuration and reproducibility. Store configurations in YAML files.
- Setting up requirement.txt(for pip), environment.yaml(for conda) and docker containers.
- Use pytorch lightning and pytorch dataloaders to streamline the training process with support for various different dataset formats.

Git Development Workflow

A Git workflow is a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner. Some of the strategy that will be useful:

- *Branching Startegy*:
  - Main Branch: Stable, production-ready code for customers.
  - Development : For development of features. Can be converted into multiple features.
  - Release : For preparation for code release
  - Hotfix : For bug fixing.
- *CI/CD Pipelines and Pull Requests*: All the code changes(already reviewed) can be submitted via pull requests which triggers the CI/CD pipeline which automatically triggers testing,linting and deployment.This helps catch errors early and ensures code quality.
- *Commit Messages*: Developers can push code with concise and clear commit messages for better tracking and communication.

Training Infrastructure

- *Cloud-based Training*: Utilize cloud computing platforms (e.g., AWS, GCP, Azure) for training which has powerful GPUs and is scalable.
- *Local Training*: Create Virtual/Conda environment by using .yaml file or requirements.txt.
- *Containers*: Create a Dockerfile that defines the training environment (Python version, dependencies, CUDA version, etc.), you can also mount volume for datasets and previously trained checkpoints.

Maintaining Reproducibility

- Use virtual environments (e.g., venv, conda) to manage dependencies and ensure consistent training environments for local setup. Docker container can also be used for consistent training across different systems.
- Use version control(Eg.Git) to keep track of various code changes and keeping older versions and use Data Version Control(DVC) for tracking changes in the dataset.
- Experiment tracking tools like Tensorboard, Weights & Bias and MLFlow can also be used to visualize the training behaviour and identify changes in the training pattern.
- Maintain similar training parameters by keeping .config files and maintaining similar seed values, same network architectures.

Handling Retraining Requests

- *Automated Retraining Pipeline*: If the performance degrades or there is new dataset, automated retraining pipeline is initialited to fetch the new training data, train the model,evaluate and deploy if performance is suitable.
- *Queueing System through API*: Implement a queueing system (e.g., Celery, Redis Queue) or an API to receive and manage retraining requests. This allows for asynchronous processing and prevents the API from being overwhelmed.
- *Versioning and Rollback*:If the retrained model performance is not satisfactory, the pipeline rollbacks to the last trained model whose performance was satisfactory.

Managing Issues like Input Data or Concept Drift

- *Input Data Issues*: Use data augmentation and noise reduction techniques to increase the diversity of the training data and make the model more robust to variations in the input data. Data validation techniques can also be helpful to maintain data format and type.
- *Concept Drift*: Statistical tests (e.g., Kolmogorov-Smirnov test, Chi-squared test) can be used to monitor/detect significant changes in the data distribution and if the performance degrades trigger(sign of ""Concept Drift") a retraining request.

  Active learning can also be useful to tackling concept drift to selectively label the most informative data points and improve the model's ability to adapt to concept drift.

Python Best Practices

- Using PEP8 coding style and Object Oriented Programming(OOPs) concepts to reduce coding redundancy.
- Regular code reviews to ensure code quality and less bugs with clear documentation.
- Using virtual environments and containers for uniform environment setup.
- Maintaining CI/CD pipelines to code,lint, test and deploy in a systematic manner.

Task 3: Deployment

In this task, we design a REST API to generate synthetic images using the ControlNet framework through REST API calls. More implementation details can be found in the Github repository (https://github.com/shayari21/ControlNet-FastAPI).

REST technology is popular because it consumes less bandwidth and is simpler and more adaptable, making it ideal for internet applications. It is primarily used to retrieve or send information from a web service, with all communication occurring via HTTP requests. A client sends a request to the server through a web URL using HTTP methods such as GET, POST, PUT, or DELETE. The server then responds with a resource, which could be in various formats like HTML, XML, images, or JSON. The functioning of REST API can be explained through the Figure.3.

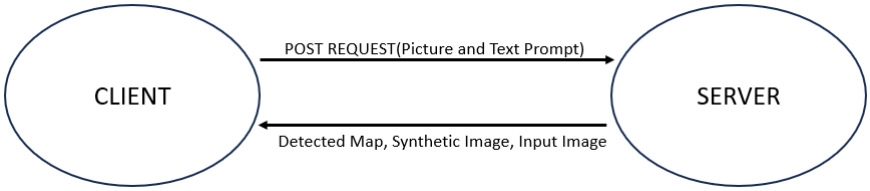

Figure 3: REST API

FastAPI is a modern web framework that is relatively fast and used for building APIs with Python 3.7+ based on standard Python-type hints. FastAPI also assists us in automatically producing documentation for our web service so that other developers can quickly understand how to use it. This documentation simplifies testing web service to understand what data it requires and what it offers. FastAPI has many features like it offers significant speed for development and also reduces human errors in the code. It is easy to learn and is completely production-ready.

In this project, we use FastAPI to develop web application for generating synthetic images by sending POST requests with input image and text prompt to the Server which hosts the ControlNet framework.

Steps of Deployment

Step- 1 Environment Setup

After cloning the original Controlnet repository, we use the environment.yaml file initially provided in the repository to configure the environment using the following command in the terminal:

```
$ conda env create -f environment.yaml
$ conda activate controlNet
```

Then we download the canny edge detection model(control_sd15_canny.pth) from the https://huggingface.co/lllyasviel/ControlNet/tree/main/models and place it in the models/ subdirectory and place the *awesomedemo.py* file in the main ControlNet repository. Then we run the demo using the command:

```
$ python awesomedemo.py
```

and we receive the output:

To enable FastAPI development, we need to add further packages

```
$ pip install FastAPI uvicorn pillow
```

so we update the environment.yaml and requirement.txt based on the new environment

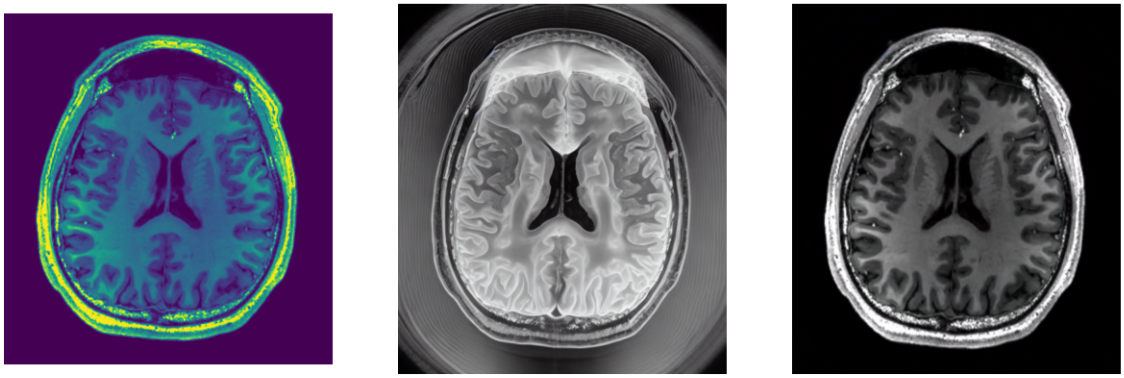

Figure 4: Caption

```
$ #for conda .yaml file
$ conda env export > environment_FastAPI.yaml
$ #for pip
$ pip freeze > requirements.txt
```

so we can use the new *environment_FastAPI.yaml* or we can continue in the same environment with the new installed packages.

Step- 2 FastAPI local Implementation

We first create a new subdirectory in the controlnet parent directory using the following command:

```
$ mkdir app
$ cd app
```

we add some files and the subdirectory in \app folder with the following purpose:

- **app\main.py**: This is the python file which initiates the FastAPI and the ControlNet model. It accepts the user-defined input image and the text prompt and calls the method *generate_synthetic_image* in model.py with the user defined input. It then receives the output from the model and creates a buffer which is then utilized to share the output to the API gateway.
- **app\model.py**: This python script accepts the user-defined input from the main.py file and processes it based on the imported model, controlnet architecture, utility function(from utils.py) and generates output using DDIM architecture. The final output which is a combined output of the canny edge detection map, generated image and the input image.
- **app\config.py** This contains various config parameters which were originally defined in the controlnet directory.
- **app\utils.py** This script contains image processing functions which are imported by models.py for image processing tasks if required.

After creating this subdirectory, we use the following command to run the FastAPI(from inside the app/ subdirectory):

```
$ python main.py
```

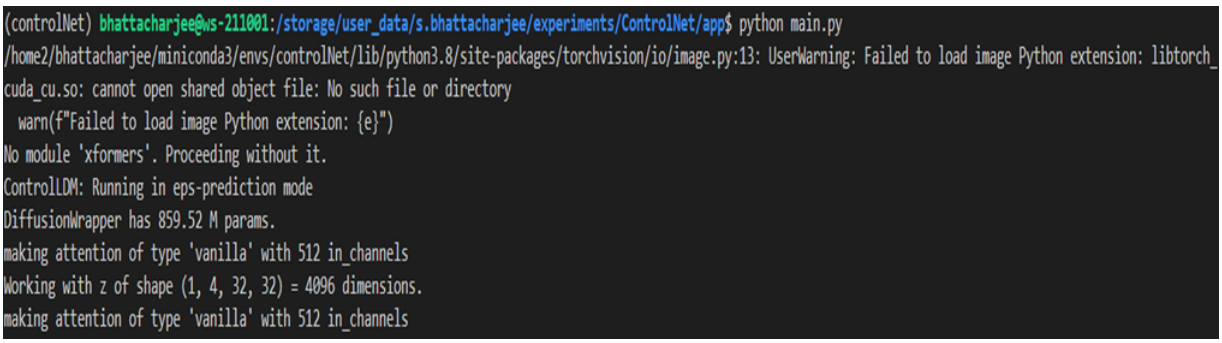

Figure 5: Running main.py to run FastAPI

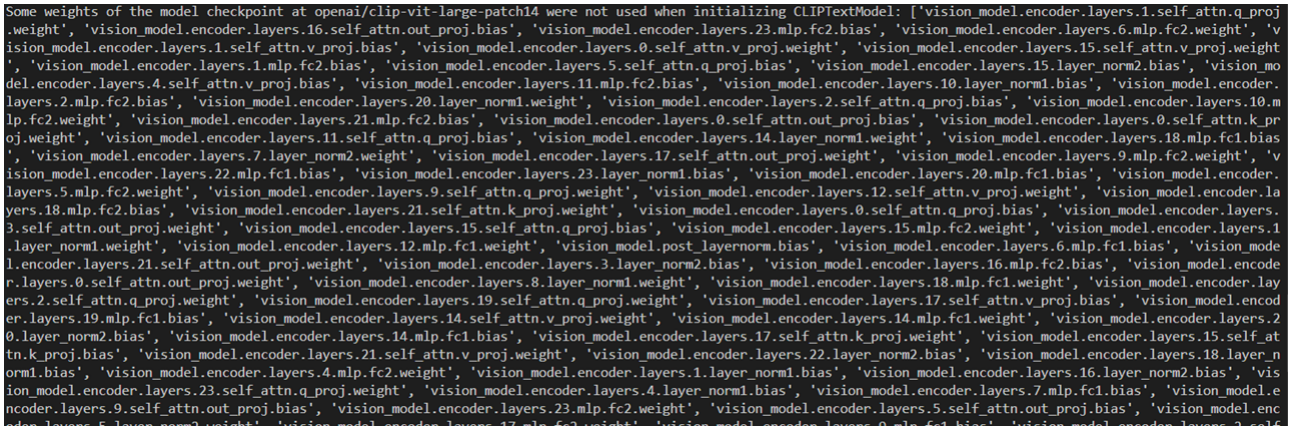

Figure 6: Dependencies being loaded

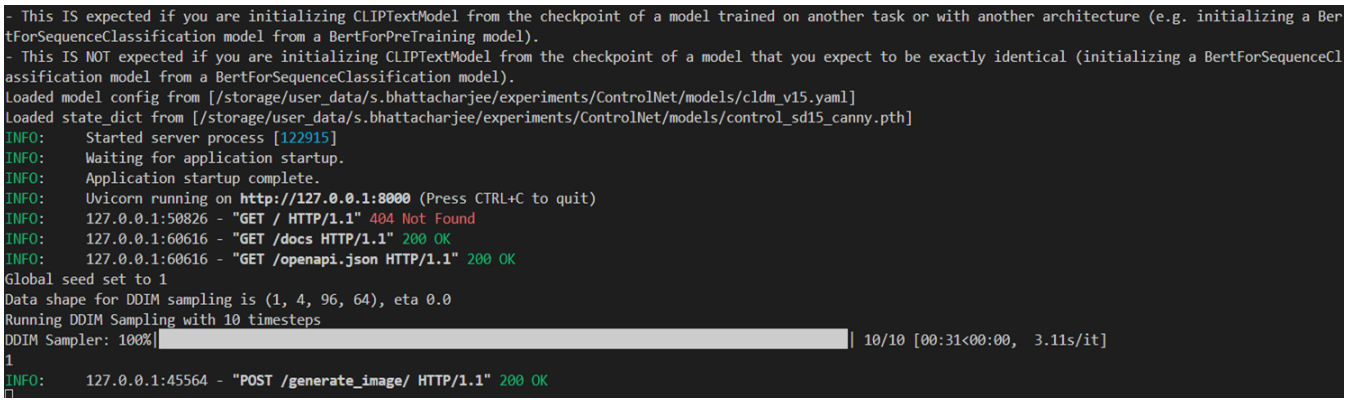

Figure 7: FastAPI is launched

and observe the following terminal response

When we run main.py, all the dependencies are loaded and finally the FastAPI is launched which can be accessed by the address http:\\127.0.0.1:8000\docs in the browser

After adding the bird image with the text prompt of "bird" we press execute to receive the following output We receive the canny edge map, the final generated image from the DDIM which runs for 10 steps and the original input image. We can exit the API by pressing Ctrl+C in the terminal to receive the following output.

Step- 3 FastAPI Docker Implementation

For Docker implementation, we first create a Dockerfile in the Controlnet repository, which has the following contents:

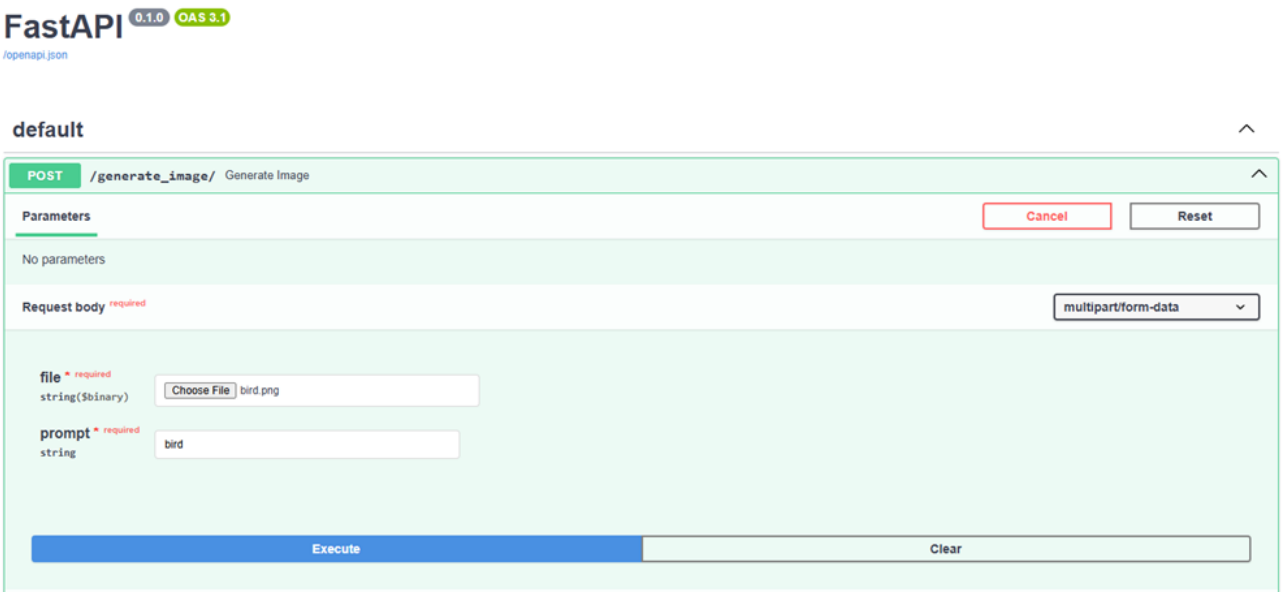

Figure 8: Receiving output from FastAPI

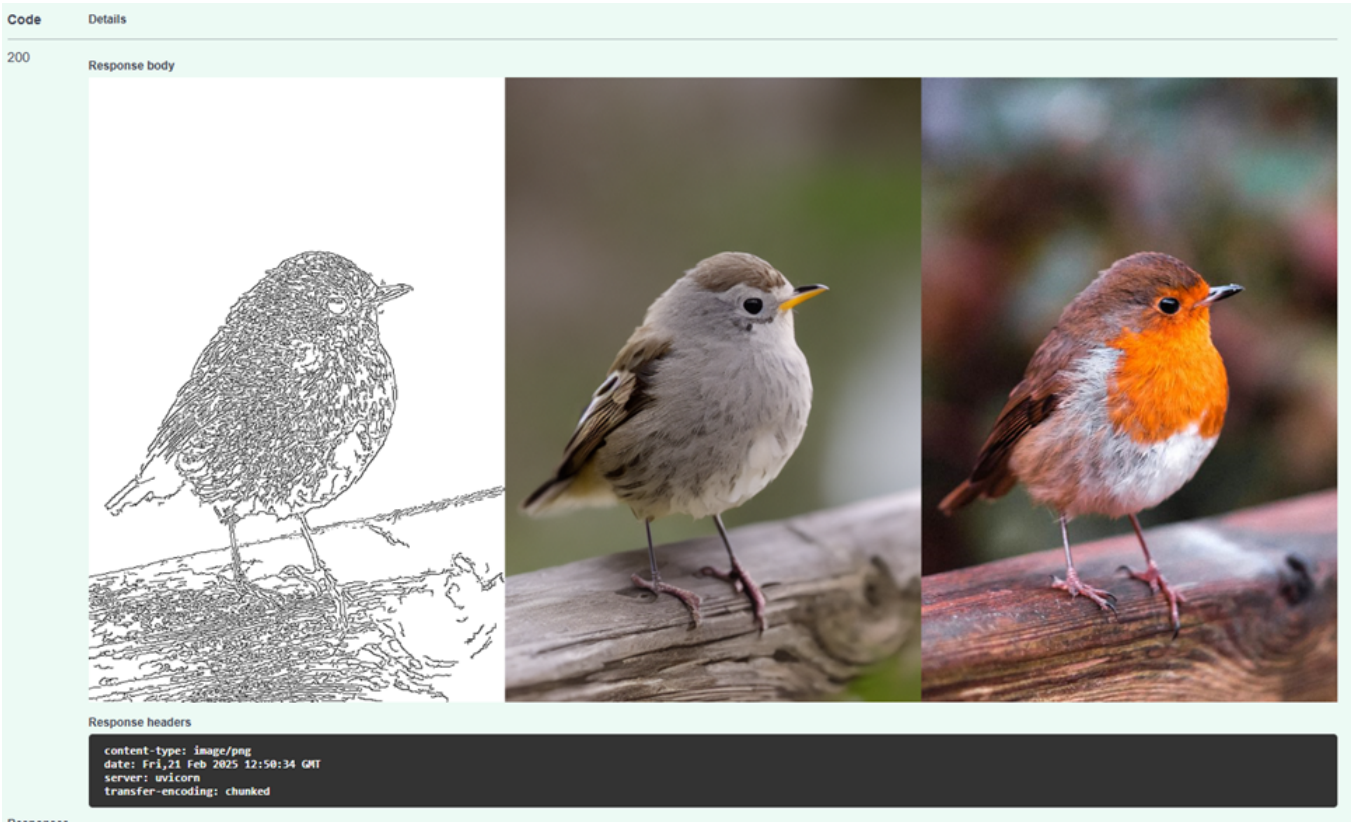

Figure 9: Receiving Output for "Bird" on Locally Hosted FastAPI

Figure 10: Closing Locally Hosted FastAPI

```
# Use an official CUDA-enabled base image
FROM nvidia/cuda:11.8.0-runtime-ubuntu22.04
```

```
# Install system dependencies (including OpenCV missing dependencies)
RUN apt-get update && apt-get install -y \
    wget \
    libsm6 \
    libxext6 \
    libxrender-dev \
    libglib2.0-0 \
    libgl1-mesa-glx \
    && rm -rf /var/lib/apt/lists/*

# Install Miniconda
RUN apt-get update && apt-get install -y wget && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O miniconda.sh && \
    bash miniconda.sh -b -p /opt/conda && \
    rm miniconda.sh

# Set Conda environment variables
ENV PATH="/opt/conda/bin:$PATH"

# Set the working directory inside the container
WORKDIR /workspace

# Copy the Conda environment file and files
COPY environment_FastAPI.yaml /workspace/environment_FastAPI.yaml
COPY . /workspace

# Create the Conda environment
RUN conda env create -f /workspace/environment_FastAPI.yaml && conda clean --all -y

# Check if Conda environment exists
RUN conda info --envs

# Activate Conda environment when the container starts
SHELL ["conda", "run", "--no-capture-output", "-n", "controlNet", "/bin/bash", "-c"]

# Set the working directory to app/
WORKDIR /workspace/app

# Expose FastAPI's default port
EXPOSE 8000

# Run FastAPI using Conda
CMD ["conda", "run", "--no-capture-output", "-n", "controlNet", "uvicorn",
"main:app", "--host", "127.0.0.1", "--port", "8000"]
```

To create the Dockerfile, we first use the nvidia/cuda:11.8.0-runtime-ubuntu22.04 base image and install the missing system dependencies and miniconda to create a conda environment inside the docker container. We setup the workspace by creating a copy of files in the controlNet repository to the docker workspace. Currently, we also copy the model.pth file present in the model\directory but in case of multiple models, it is a better approach to mount the volume for faster processing. We then activate the conda environment inside the docker container and change the directory to the /workspace/app directory where the FastAPI files are situated and then we lauch the API using the conda command

```
CMD ["conda", "run", "--no-capture-output", "-n", "controlNet", "uvicorn", "main:app",
"--host", "127.0.0.1", "--port", "8000"]
```

Followed by creation of dockerfile, we run it to create the docker image using the following command from ControlNet-FastAPI directory:

```
$  docker build -t controlnet-FastAPI .
```

and receive the following output after successfully creating the docker image, we run the container(with gpu support) using the command

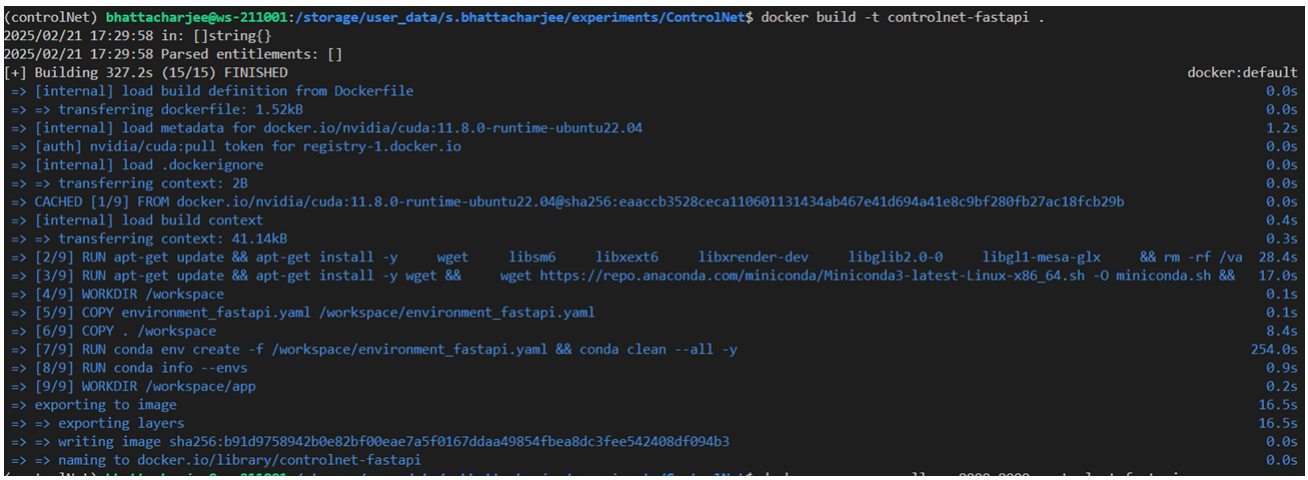

Figure 11: Running Dockerfile

```
$  docker run -gpus all -p 8000:8000 controlnet-FastAPI
```

and receive the following response

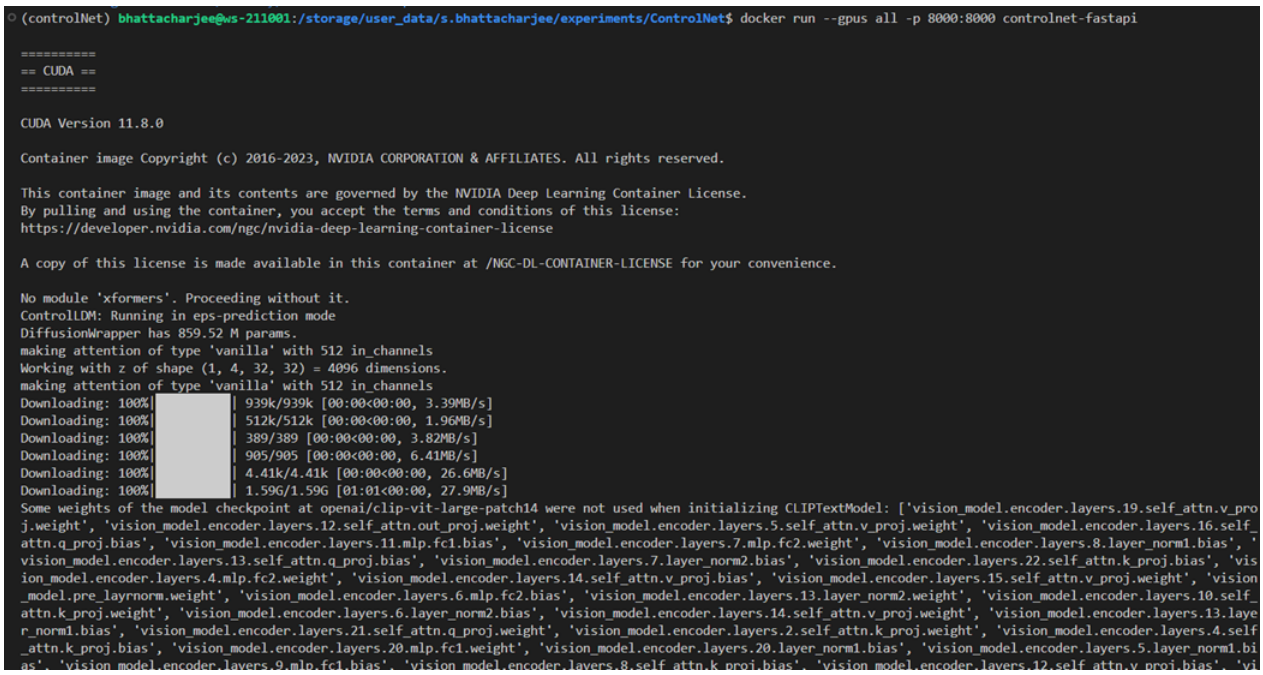

Figure 12: Starting Docker Container

After starting the container, we can observe that the API has been deployed when we observe a response such as

We can access the API through http:\\127.0.0.1:8000\docs in the browser or through CURL command such as :

```
$ curl -X 'POST' \
  'http://127.0.0.1:8000/generate_image/' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@bird.png;type=image/png' \
  -F 'prompt=mri brain scan'
```

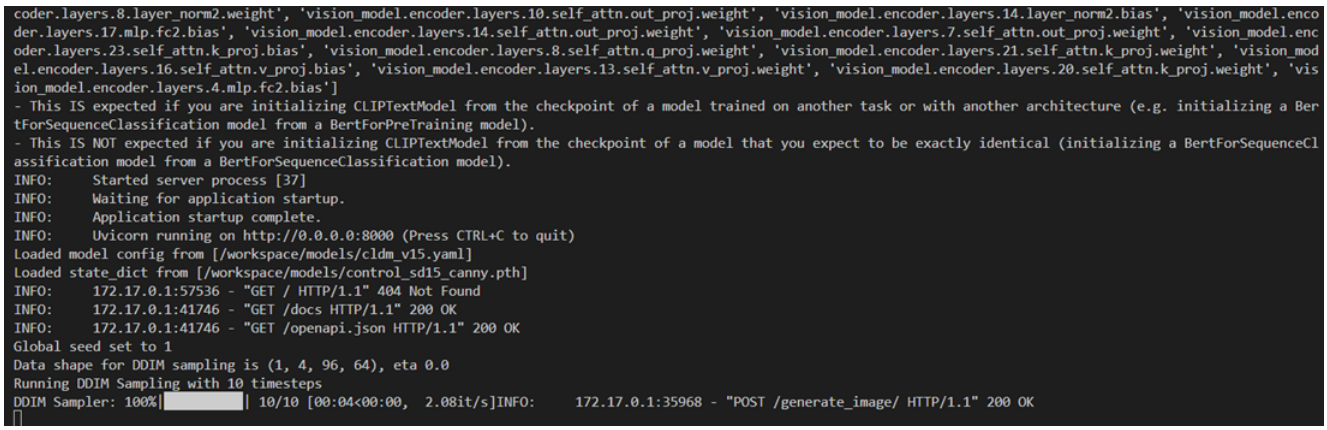

Figure 13: FastAPI deployed

and the response on the browser looks as follows: After adding the bird image with the text prompt of "bird" we press execute to receive the following output

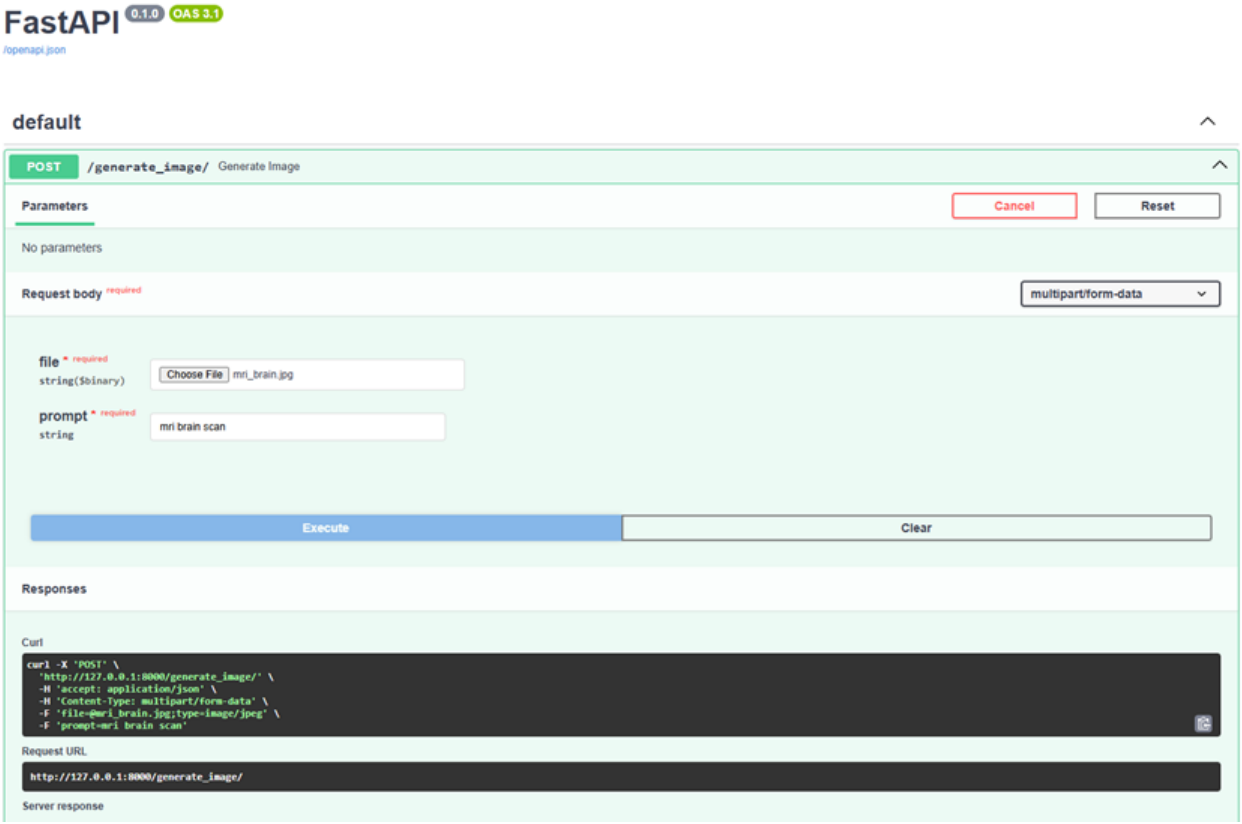

Figure 14: Providing Input to Docker-Deployed FastAPI

Improvements Made in the Codebase

- Modularized code for better readability and adapted code based on the new structure.
- Added docker support.
- Implemented FastAPI for better user interaction.
- created new environment_FastAPI.yaml and requirements.txt for better reproducibility.

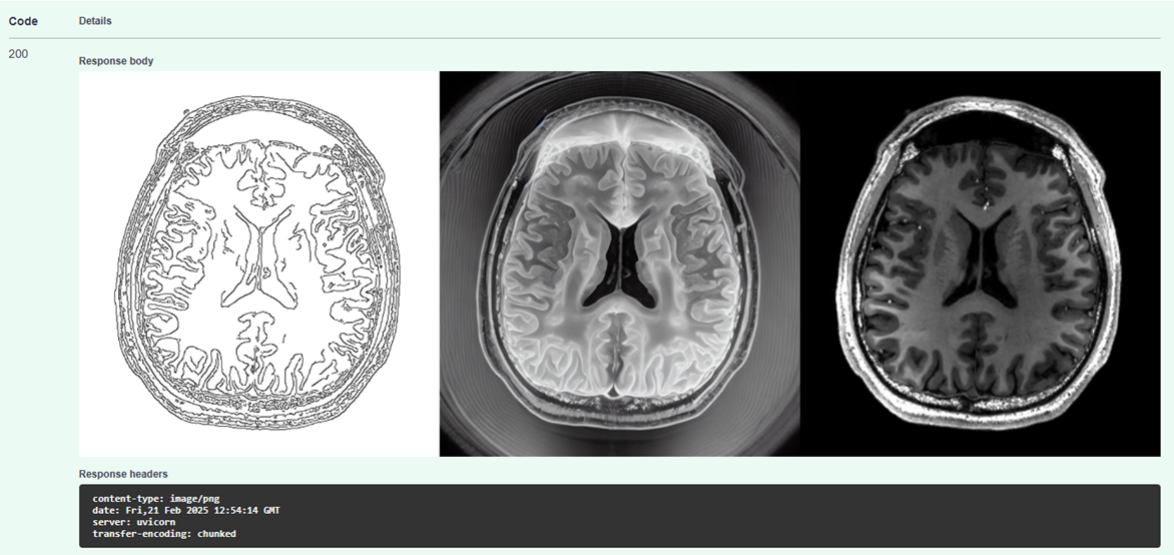

Figure 15: Receiving Output for "mri brain scan" on Docker-Deloyed FastAPI

Future Scope of Improvement

- Currently, the model only uses canny edge detection for image generation, it can be extended to include other modes of image generation which involves segmentation maps, HEC maps, pose detection which are originally present in the ControlNet directory.
- The prompt of the FastAPI can be improvement by adding one more field for a text prompt which would indicate the mode of image generation.
- The UI of the deployed FastAPI is currently the one which is default, it can be improved for better user experience.

References

Golda, A., Mekonen, K., Pandey, A., Singh, A., Hassija, V., Chamola, V., and Sikdar, B. (2024). Privacy and security concerns in generative ai: A comprehensive survey. *IEEE Access*.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2021). High-resolution image synthesis with latent diffusion models.

Zhang, L., Rao, A., and Agrawala, M. (2023). Adding conditional control to text-to-image diffusion models.