MSML650 Project Final Report

Jiayuan Shen
MSML
UID: 118464947

Jihai Luo
DATA
UID: 120387807

Ke Xu
MSML
UID: 120161216

Yan Liu
DATA
UID: 120424595

Yuqin Feng
MSML
UID: 120196935

***Abstract*—The primary goal of our project is to accurately generate the image in terms of the input text prompt and deploy the trained model within the SageMaker platform provided by Amazon Web Service(AWS), allowing users to invoke the endpoint utilizing the service at any time, facilitating further applications such as image-to-image generation, image modified service, and other multimedia applications. Our project leverages vigorous concept transfer learning, adapting a pre-trained generative model, the Stable Diffusion, renowned for its efficacy in image generation tasks. This adaptation involves adjusting the model's hyperparameters to focus on generation metrics, utilizing a specific inference pipeline tailored to this task. Initial results demonstrate that the adapted Stable Diffusion model can accurately generate the image based on various text prompts. The model performs well, with the generated images' backgrounds and lighting conditions exhibiting robust coherence, generalizability, and reliability. However, the generation speed varies by the scene's complexity and the hardware's performance, suggesting areas for further optimization and improvement. The project confirms that diffusion architecture can generate a coherent image where the input is the Gaussian random noise combined with the stable diffusion model, a method to control the output images based on text prompts. Furthermore, AWS illustrates a platform to deploy the model, which helps us become familiar with the inference process and deployment approaches and bypass actual GPU limitations by paying a few bills. While the model delivers high-accuracy results, future work will enhance processing speed and update the model architecture to an industrial version to generate more colorful and pertinent images; we could involve further hyperparameter tuning or the integration of more streamlined neural network architectures.**

I. Problem Statements

In recent years, the rapid advancement of artificial intelligence (AI) and computer vision (CV) has significantly transformed the landscape of human-computer interaction with images. As one of the most prominent applications of these technologies, image generation tools, providing a different perspective of perceiving arts, have become integral tools in various domains, including gaming design, e-commerce, healthcare, and education. Despite their widespread adoption, existing image generation tools technologies still face substantial challenges in understanding prompts, generating coherent images, and scaling effectively to handle diverse queries from a global user base.

This project aims to develop and implement a sophisticated text-to-image denoise diffusion model based on the Amazon Web Services (AWS) platform. The diffusion model derives from the Variational Autoencoder probabilistic model combined with Bayesian statistics and Markov chain Monte Carlo, enabling a more stable training process, providing more accurate and contextually relevant responses, which is its original architecture does not have a port for textual information and since the input supposed to be the Gaussian noise, which leads that the style of the output images should be verisimilar within the training dataset without explicit control panel.[5]However, it adds and denoises the entire image, which induces tremendous training resources and diverges from the original aspects that generate the image based on the text prompt. Therefore, this project aims to explore efficient methods for integrating textual input into diffusion models, construct the intuition behind the mathematical formulation, understand the training and inference processes, merge the stable diffusion model instead of the original denoise diffusion probabilistic model, and evaluate the model's capability to generate coherent images based on an input text.[3].

II. Literature review

A. Review of Related Work

Historically, image generation has transformatively evolved from simple geometric models to more sophisticated machine-learning approaches. Early methods included the PixelRNN, which effectively generated the image pixel by pixel by modeling the traditional conditional distribution of each pixel given previous ones to maximize the likelihood computation. However, it lacked robustness against complex facial appearance variations and was extremely slow due to the nature of sequence generation. Furthermore, the family of autoencoders was another method that used the variational lower bound combined with the reverse KL divergence and reparameterization trick to approximate the true distribution with another distribution to construct the loss function for training the image generation model, which ensures that the generated image is verisimilar with the training sets. With the advent of deep learning, particularly the transformer architecture, the diffusion family significantly dominated the performance of image generation systems. Researchers have developed numerous transformer-based concerning the residual connection architectures, such as the stable, which can be attributed as an innovative adventure that merges various attention mechanisms, residual connections, latent space, and the Contrastive Language-Image Pretraining (CLIP) as a multimodal for better comprehending the text prompt for image generation, which is also the architecture we used for this project.

B. Identification of Gaps

Current advancements in face detection predominantly utilize multimodal, which is a combination of the classifier model, symmetric transformer architecture neural network, latent encoder, latent decoder, and a denoise diffusion sampler; while effective, it often require complex, multi-stage processing pipelines that can introduce inefficiencies due to the nature of training two separate model. Recent research has begun to explore the incorporation of classifier-free guidance in image generation; these implementations are widespread and typically fully leverage the transformer's capabilities for global context integration and end-to-end training. However, these techniques still demand high-performance computing resources, significantly burdening hardware requirements and limiting their scalability for real-time applications.

Our project aims to address this gap by adapting the stable diffusion model, which fundamentally alters the traditional generation paradigm through its encoder-decoder structure. By employing classifier-free guidance and the AWS SageMaker online platform, we uploaded the pre-trained stable diffusion model parameter within the S3 buckets, attached the entire generation process to the AWS ECR service, and leveraged the real-time inference of the SageMaker to create an end-to-end connection facilitate the user for invoke the model. By adopting this particular method, we not only bypass the requirement of the GPU but also have a mature stable diffusion model.

III. Related Work

This section will briefly introduce the mathematical intuition behind the stable diffusion model, the model architecture, and the related section we used for AWS services.

A. Mathematical Intuition

Generally speaking, the core stable diffusion model adds high-dimensional Gaussian noise to the latent space(an abstract image representation) through the forward diffusion process and gradually removes the noise through reverse diffusion to achieve the same image as input during the training process. Through the training process, we also add the time dimension and expect the model to understand the noise we add at particular timesteps so that the model will learn how to remove the noise as expected. [3], [10], [7]

B. Model Architecture

1) Encoder: The stable diffusion encoder draws on the idea of the variational autoencoder. Instead of adding the noise directly to the original shape of the noise, it adds the noise to the latent space to reduce the speed for training and inferencing and also mitigate the need for computational resources. By gradually reducing the spatial resolution of the input image (downsampling) and increasing the number of channels, the encoder extracts features at multiple levels. For this project, since we load the pre-trained parameters, we set the shape of the latent space as [Batch_Size, 4, 64, 64] for consistency to encompass the original image and retain the essential information of each image.[4], [11] Furthermore, unlike the traditional autoencoder, which merely uses the convolution and the pooling layer, the stable diffusion encoder also adopts self-attention, residual connection, and group normalization to reserve more space so that the model can learn more during the training stage.

Residual Blocks

- Capture local features through multiple convolutional layers.
- Retain input information, prevent feature loss, and improve training stability.

Attention Blocks

- Extract global context information to capture long-range dependencies in the input.
- Focusing on key perception of the input.

Code Implementation: https://colab.research.google.com/drive/1p3KdVfeTi5ckpqSFKW6kcibV7TXOujRw?usp=sharing[4], [11]

2) Contrastive Language Image Pre-training(CLIP): A common technique is to represent words as high-dimensional vector embeddings to allow the model to understand a given text prompt. This approach allows the model to generate coherent responses by interpreting the text in a continuous, semantic space; therefore, the stable diffusion draws on a similar idea adopting the CLIP model, which OpenAI invented with Ang Li and his co-authors at FAIR who in 2016 demonstrated using natural language supervision to enable zero-shot transfer to several existing computer vision classification datasets, such as the canonical ImageNet dataset.[6]

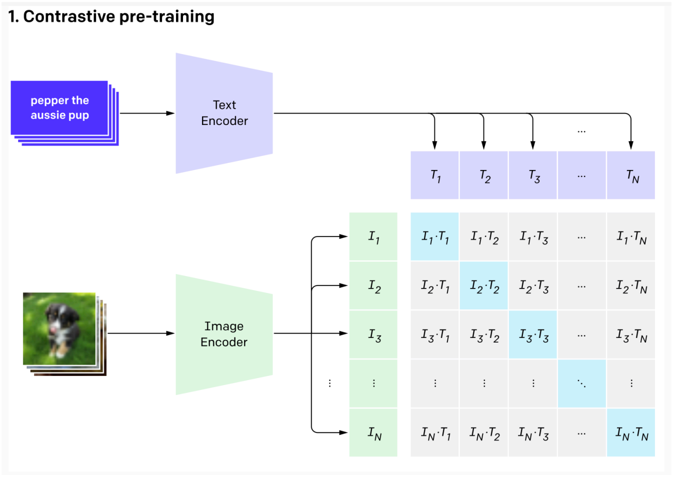
1. Contrastive pre-training
pepper the aussie pup
Text Encoder
Image Encoder
T_1 T_2 T_3 ... T_N
I_1 I_2 I_3 ... I_N
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ ... $I_1 \cdot T_N$
$I_2 \cdot T_1$ $I_2 \cdot T_2$ $I_2 \cdot T_3$ ... $I_2 \cdot T_N$
$I_3 \cdot T_1$ $I_3 \cdot T_2$ $I_3 \cdot T_3$ ... $I_3 \cdot T_N$
$I_N \cdot T_1$ $I_N \cdot T_2$ $I_N \cdot T_3$ ... $I_N \cdot T_N$

Fig. 1. High-Level Overview of the CLIP[6]

The virtual representation of the CLIP model is illustrated as follows 1: The idea behind CLIP is to train both a text encoder and an image encoder so that each input (whether text or image) is transformed into a corresponding high-dimensional vector.[8] Once trained, the similarity between a text prompt and an image is determined by their dot product(Cosine Similarity); a high dot product value indicates that the textual description matches the image content. In this manner, the loss function of the CLIP model encourages maximizing diagonal elements and minimizing other elements to learn to comprehend text prompts and effectively link them to related visual representations.[8]

Code Implementation: https://colab.research.google.com/drive/1PBRZSrtee7GQXa-pg4VVsqw24tLKGoFt?usp=sharing[4], [11]

3) Decoder: It is the counterpart to the encoder in the Variational Autoencoder. Its job is to take the latent representation (low-dimensional compressed features) through the symmetric nature upsampling method to reconstruct the original high-dimensional input, such as an image.

Code Implementation: https://colab.research.google.com/drive/1eMRn93f9KuJm19BKoy_jzGJAW2OTBXti?usp=sharing[4], [11]

4) UNET: In stable diffusion, the primary component of the noise prediction part has been achieved by a UNET-like architecture. Unlike the original UNET, the stable diffusion is also augmented with cross-attention, which allows the model to incorporate additional conditioning signals (e.g., text embeddings in the CLIP). These attention layers are interleaved within the U-Net' s encoder and decoder blocks. They guide the noise prediction process so that the resulting image aligns not only with a prior distribution learned from training data but also with specific user-provided prompts. Furthermore, with long and short skip connections, we can ensure that even within the multi-scale encoder-decoder(Here, the encoder and decoder are not the components we introduced previously; it is that due to the symmetry of the UNET architecture, we convent the left part as encoder and the right part as decoder) network, the large amount of information will be reserved even the scope the model components are significant.

Code Implementation: https://colab.research.google.com/drive/1yYt575OUqLomm3UDFTsJhoD40E79P3Wh?usp=sharing[4], [11]

Comprehensive List of the Code Reference:

- **Major Parts:** Umar Jamil's video, Hugging Face Library, DDPM paper and Stable Diffusion paper. [4], [2], [3], [10]
- **Minor Elements:** Umar Jamil's video, Hugging Face Library, and a book. [4], [2], [9]
- **Model Parameters:** Hugging Face Libraries, Github Repositories. [2], [1]

Figure 2 shows the High-Level overview of the Stable Diffusion.

C. Amazon Web Services (AWS)

To successfully employ the stable diffusion, we leverage the advantages of the AWS cloud computing services, including SageMaker, Jupyter Lab, S3 buckets, and Amazon Elastic Container Registry(ECR), to deploy our model for real-time inference purposes successfully.

1) SageMaker: SageMaker provides the computation resources for this project, allowing stable diffusion with enough GPU memory and CUDA computation. We choose the ml.g4dn.2xlarge instances for inference tasks, which offer 1 NVIDIA T4 GPU with 16GB that satisfies the minimum requirements for executing the inference pipeline. It costs 0.94 per hour, which is affordable for testing the code and experimenting. Furthermore, SageMaker also provides the endpoint of the model so that we can invoke it through the regular application-level request, and the model will respond to it as a JSON file. Several post-processing tools, such as decode base64, can display the images if the model can be deployed to the endpoint.

2) Jupyter Lab: Jupyter Lab is a subsection within SageMaker that provides a platform to execute the Python code for creating, configuring, and deploying the endpoint, similar to the Colab environment but with less flexibility. Additionally, Jupyter Lab also established the connections between endpoint, ECR, and S3 buckets so that our model can run sequentially without extra manual interference during the inference stage.

3) S3 buckets: S3 buckets have been used to store the model artifacts, including the external parameters and essential supporting files like the tokenizer. During our investigation, S3 buckets will also be used to receive the output if we use the asynchronous inference service. Still, since we are using real-time inference, we eliminate a step to increase the robustness and reliability of the process.

4) Amazon Elastic Container Registry(ECR): ECR is an Amazon version of docker, which contains all the necessary CUDA accelerated runtime, start point, inference pipelines, external libraries, and the server configuration that SageMaker requires.

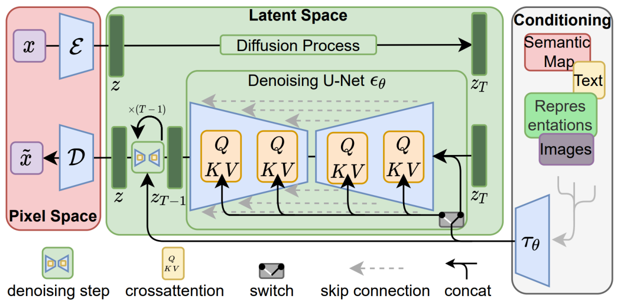

Fig. 2. High-Level Overview of the Stable Diffusion [10]

IV. Solution and its Significance

Although we have already introduced the tools we used, it is still far from achievement. We refine and revise the entire process to fulfill the live demo performance by slightly modifying the model, reconstructing the pre-trained parameter loading process, and redesigning the docker image structure.

A. Model Refine

Since we are using the pre-trained parameters, it is infeasible to change the model's structure to protect the external files' integral and prevent unexpected loading issues. First, since the SageMaker real-time deployment required a 60s response, which means that for each output image, we had to acquire the result within 60s, we slightly changed how the original author coded it. For example, due to the timely manner in which the author introduced some out-of-date methods by inheritance of the nn.Sequence module while constructing the primary model class, we refine it using orthodox approaches that directly use nn.Module. Also, while we transform the log variance to the standard deviation, the original author adopts the regular exponential method; we, on the other hand, utilize the log property product 0.5 within the exponential part to avoid overflow issues shortcut an extra processing speed. Secondly, during the UNET architecture construction, the author's logic in identifying the layer can be partly attributed to redundancy; thus, we made a significant emendation for more clarity and readability.

Additionally, since the attention mechanism has been ubiquitously adopted in our project, we initially attempted to involve the flash attention method to accelerate the processing speed; nevertheless, we realized that the T4 GPU does not support the flash attention method; therefore, we still convert it into an efficient way to calculate the attention score by merging the three QKV matrix rather than three separate linear layers. Finally, to fulfill the requirements, we abolish the traditional method of loading parameters through the state_dict() instead of encompassing the entire model into a .pt file. Another advantage of this initiative is that we could reduce the number of libraries while constructing the docker image. This remarkably mitigates the pressure of managing the module dependencies and file path adjustment.

B. Docker Image and ECR

According to the AWS official example about deploying a prebuild sentiment analysis SpaCy model using SageMaker, we find out that their tutorial misses several essential materials, which the example dockerfile they provide contains factual mistakes, and we cannot reproduce their results through our environment. Therefore, we first fix the error in the dockerfile and run the container in the local environment. After a few rounds of experiments, it turns out that the main error concentrates on the supporting architecture of the computational instances, some of which only support x_86 architects; thus, we used an x_86 laptop to construct the docker image and complete the reproduction eventually. However, the situation becomes more tricky when we attempt to deploy our own model. The first problem we solved was the size of the docker image; in order to bypass and utilize the CUDA environment, we must upload the appropriate runtimes that support CUDA acceleration, leading to reinstalling the entire Python and the requirement libraries within the docker image. Furthermore, we try to solve the architectural dependency by installing Ubuntu inside the docker images, but due to the size of our model, the processing time significantly exceeds our expectations; therefore, we stick with the original plan of using a x_86 laptop while building the docker image.

On the other hand, multiprocessing is another major problem we encounter; the default start point uses the multiprocessing for loading the model parameters, but for a large number of model parameters, we entail redesigning the multiprocessing by setting the memory block and locks to resolve the conflict loading. Nevertheless, due to the restricted time, we use one CPU to load the model artifacts.

V. Novelty and High-level approach

While diffusion models for text-to-image generation have been explored by researchers and companies with products like DALL·E 2 and Midjourney, our project can still hold novelty in several ways, incorporating a consolidated understanding of the holistic procedure for inference the diffusion-related model by undertaking the entire pipeline during the project development; showcases a fully integrated, production-ready pipeline on AWS including, containerizing the model with Docker, pushing it to ECR, and setting up a scalable inference endpoint via SageMaker. Secondly, instead of fine-tuning, importing external pre-trained stable diffusion parameters is directly integrated into the AWS environment, which simplifies the deployment process, demonstrating how to efficiently leverage pre-trained weights on a cloud computing platform, potentially reducing both development time and computation costs. Thirdly, the inference of a CV task-related model demands tremendous high-performance computing resources to accelerate the entire process, particularly within the limitation of the 60s response. We investigate several pertinent strategies to use limited resources effectively and introduce novel model optimization and efficiency approaches. Finally, demonstrating the effective use of AWS services for machine learning tasks provides valuable examples of cloud integration.

A. Mathematical Ingestion

- **Variational Lower Bound**: Understanding the underlying method of ELOB is necessary to comprehend the purpose of model design.
- **Forward and Reverse Diffusion**: Comprehending the probability theory that involves these two processes.

B. Model Implementation

- **Model Construction**: Construct a denoising diffusion probabilistic model (DDPM) tailored for text-to-image generation. Focusing on pure image generation at early stage.
- **Integrate Text**: Implement mechanisms to condition the diffusion process on textual input, such as concatenating text embeddings or using cross-attention layers.

C. AWS Platform Setup

- **Computation Resources**: Using AWS SageMaker instance with GPU acceleration or Google Colab Pro for model inference.
- **Storage**: Leveraging S3 for storage the weight parameters.

D. *Evaluation*

- **Post-processing**: Convert the logits back to the images and transform it to a specific format for evaluation and demonstration purposes.
- **Quantitative Metrics**: Assess the model using metrics like Frechet Inception Distance (FID) to evaluate the quality of generated images.

VI. Evaluation

We use FID(Fréchet Inception Distance) to evaluate the quality of our picture. FID is used to measure the quality and diversity of generated images by comparing the statistical characteristics of generated images with real images and is one of the most commonly used evaluation metrics for image generation quality.

In contrast, CLIP measures the semantic alignment between generated images and their associated text prompts, with higher scores reflecting better correspondence between the two. While FID emphasizes the fidelity of the generated content to real-world distributions, CLIP highlights how well the images match the intended descriptions, showcasing the trade-off between visual quality and text-image consistency in generative models. Lower FID scores indicate better realism and diversity. However, CLIP Scores measure the semantic alignment between generated images and text prompts, with higher scores indicating better text-image consistency.

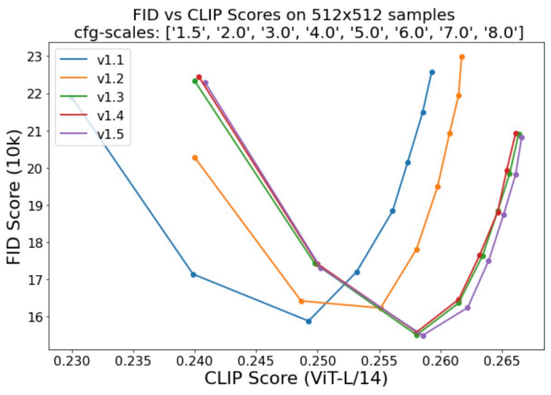

Fig. 3. Evaluation [1]

Figure 3 shows the the difference bwteen FID and CLIP.

According to this plot, a clear trade-off between FID and CLIP Scores. As the CLIP Score improves (indicating stronger text-to-image alignment), the FID Score tends to increase, reflecting a loss in image quality or diversity. The configuration scale (cfg-scale) also plays a crucial role in determining performance. Lower scales favor realism and diversity, resulting in lower FID Scores, while higher scales prioritize text alignment, leading to higher CLIP Scores but potentially sacrificing image quality.

Figure 4 shows the model output by giving the following prompt:

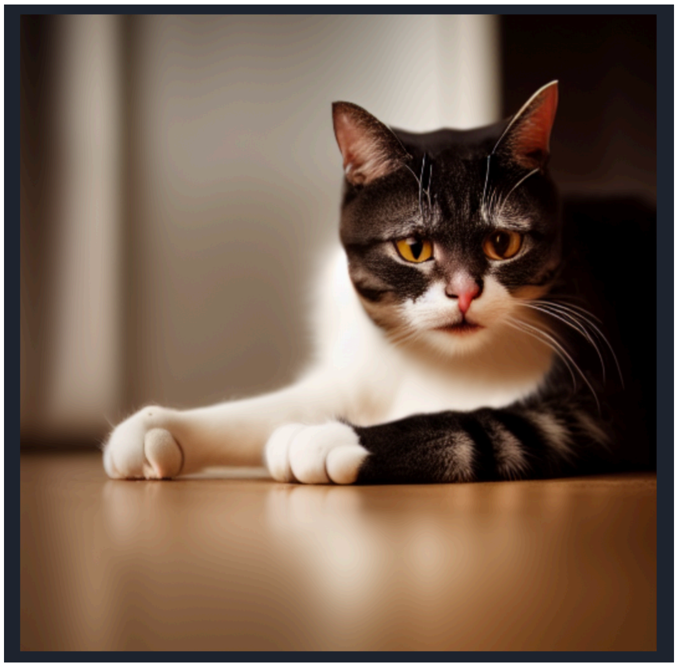

Fig. 4. Actual Output

- **Prompt**: A cat stretching on the floor, highly detailed, ultra sharp, cinematic, 100mm lens, 8k resolution.
- **Uncond_prompt**: ”.
- **Strength**: 0.5
- **cfg_scale**: 7.5
- **num_inference_steps**: 50

VII. Challange and Future Improvement

A. *Challenges*

Even though the generated process does not raise any error and produces the appropriate images, the final outcome we want to present to the user is still far ahead compared to the perfect result. While we tested different text prompts and expected a pertinent result, the model kept sticking around with less colorful images than the Midjourney, which might suggest that the generative method, pipeline function, and the DDPM process still entail further attention in finding out whether or not the model architecture entails update or the model parameters should be fine-tunning toward the forward and reverse diffusion inclined adjustments incorporating implement the correct way of adding the noise into the latent space, the method of calculating the predicted mean and variance. The classifier-free guidance method heavily relies on the accuracy of the CLIP model in which both the stable diffusion and the CLIP parameter we used are 2022 versions; we still need to pursue innovative papers and approaches in order to generate more colorful images.

During the period between the two process reports, we have conquered the challenges illustrated in the previous paragraph and encountered new difficulties that impede development. First, even though we encapsulated and streamlined the model, the entire docker image is still more extensive than expected, preventing future scalability. Secondly, despite the official documents providing a sample for teaching us how to design and implement the docker images, we followed the example precisely but still got tedious errors in the console, where most of the log messages were vague and hard to understand, even unable to reproduce the example and get the result. Furthermore, we also face the challenges of security configuration, in which SageMaker automatically creates some of the roles, and we do not entirely comprehend how those policies have been managed. Moreover, since the team is familiar with the backend, the lack of front-end knowledge also retard our steps. To leverage the SageMaker endpoint of model deployment, we should merge the GET and POST methods in our customer inference code but stuck with the integration of the two parts. Likewise, in the docker file, another essential component of the docker image, the distinction between other recourse and the official document, needed to be clarified about keeping in the correct trajectory. Finally, some of the log messages also suggested that we change a performance instance. However, since we could not diagnose which segment got wrong, we followed a conservative approach.

B. Future Plan

In the future, we will attempt to implement some advanced methods that increase the inference efficiency, like replacing regular multi-head attention with qv cache to accelerate the inference procedure, changing the variance schedule according to the new paper, which states that the previous variance schedular setting does not wholly convert the SNR ratio to zero. However, some novel approaches could solve this dilemma to make the generated image more creative. Finally, we must enhance our model security issues while using the AWS platform to access and present the model through the regular website by setting the appropriate Elastic IP address and the VPC.

VIII. Contribution

Jiayuan Shen

- Literature review the architecture of the diffusion model.
- Implement the basic architecture using PyTorch.
- Integrate text into the existing mechanisms.

Jihai Luo

- Collect or seek suitable pre-trained parameters of stable diffusion, CLIP, and tokenizer.
- Write the CLIP (Contrastive Language-Image Pre-training), UNET, and the DDPM (Denoise Diffusion Probabilistic Model) variance schedular of the stable diffusion
- Write dockerfiles and upload the files into docker.

Ke Xu

- Create a new ECR repository in AWS CLI and push an image to ECR.
- Write nginx.conf, predictor, serve, wsgi.py files using for uploading files into docker.
- Configure the end-point, client, and boto3 using Jupyter Lab.

Yan Liu

- Write the encoder and decoder sections of the Stable Diffusion code.
- Test the model with various input texts and analyze the quality of generated images.
- Maintaining the compatibility between Python and AWS.

Yuqin Feng

- Deploy the trained model for inference using AWS services.
- Develop an API or web interface for users to input text and receive generated images.
- Manage data storage using AWS services like S3

IX. Implementation Tools

- **Python and Relative Libraries**: Programming language for model architecture construction, data pre-processing, data post-processing.
- **AWS**: A platform that contains the relative parameters and training procedures.
- **Hugging Face**: An external application for loading the dataset and evaluation metrics.
- **ALL MATERIALS LINK: https://drive.google.com/drive/folders/14hW8dijEC6YQno4LHnm5thXR3h65wDOE?usp=sharing**

X. Reference

[1] Stability AI. Stable diffusion 1.5. https://huggingface.co/stable-diffusion-v1-5/stable-diffusion-v1-5, 2022. Accessed on 25 October 2024.

[2] Stability AI. Stable diffusion 2.1. https://huggingface.co/stabilityai/stable-diffusion-2-1, 2023. Accessed on 25 October 2024.

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[4] Umar Jamil. Coding stable diffusion from scratch in pytorch. https://www.youtube.com/watch?v=ZBKpAp_6TGI, 2023. Online; accessed 25 October 2024.

[5] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[6] Ang Li, Allan Jabri, Armand Joulin, and Laurens van der Maaten. Learning visual n-grams from web data, 2017.

[7] Calvin Luo. Understanding diffusion models: A unified perspective, 2022.

[8] OpenAI. Clip: Connecting text and images. https://openai.com/index/clip/, 2020. Accessed: [Insert the date you accessed the site here].

[9] Sebastian Raschka. *Build a Large Language Model (From Scratch)*. Manning Publications, 2024.

[10] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[11] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, Steven Liu, William Berman, Yiyi Xu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models.