Fine Tuning Stable Diffusion Model With LoRA

Yuanhe Guo #1, Yiling Cao #2

Abstract

We want the stable diffusion model to learn to generate images in a specific style. And our work is a downstream task of a diffusion model, which can do a specific mission. The difficulty lies in how to fine-tune a model with a large number of pre-trained parameters. We adopted the same LoRA method for fine-tuning the large language model, analyzed the structure of the stable diffusion model, and added new parameters to the cross-attention layer in the decoder for training. The LoRA model we trained can allow specific prompt text to guide the model to generate pictures of a certain style. After using scribble data for training, we successfully used the stable diffusion model to generate pictures stably in the style of ordinary people's hand-painted scribbles. And the size of the LoRA model is only 3.3 MB, which is very easy to share and use.

1 Introduction

Text-driven image generation AI has become a hot spot again since 2022, with the emergence of Dall-E2[1], stable diffusion[2], midjourny[3] and so much more. Other than using GAN, these projects open their arms to diffusion models. As is explained in this paper[4] by researchers in stability AI, they combine CLIP[5] with guided diffusion and create stunning results. Most states of the art diffusion models are integrated into the Hugging Face diffusers pipeline[6], and there's an open-source web UI for image generation[7].

For pre-trained diffusion models like stable-diffusion-v1.5[2], although stable-diffusion-v1.5 has a stunning ability to generate a wide range of images, we find it hard to guide it towards a specific image style. In our project, we want to fine-tune a diffusion model to generate a specific style of images.

2 Data

2.1 Dataset

We use the quickdraw dataset from google creative lab[8], a dataset of over 50,000,000 hand-drawn images labelled into 345 categories. The dataset is pre-processed into various datatypes, and we choose NumPy .npy format of the simplified drawings that have been rendered into a 28x28 grayscale bitmap. We randomly select 100 categories and split them into two subsets of 70% and 30% data respectively. Then in each category, we randomly select 20 example images for training the LoRA model. Figure1 shows eight randomly picked examples of image-category pairs in the quick draw dataset.

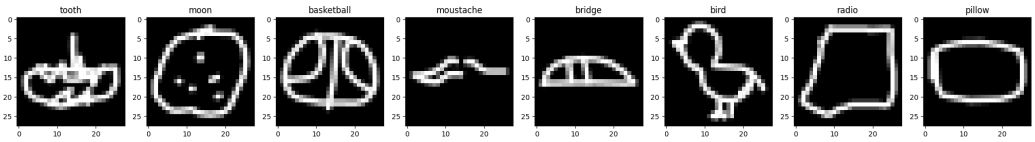

Figure 1: Examples of image-category pairs in the dataset

2.2 Image Processing

The base model we choose, stable-diffusion-1.5, was trained on RGB images with a resolution of 512x512. We first map our 28x28x1 bitmap data into 28x28x1 RGB images. Then we resize images to 512x512x3 using bicubic interpolation. Now the images consist of white strokes on black backgrounds, but we want our trained model to generate scribbles that have black strokes and white backgrounds. Thus we invert colours of all images. To enhance the dataset, we apply random horizontal flips to all images. Finally we convert image data into tensors and normalize them with $\mu = 0.5, \sigma = 0.5$. Figure2 shows processed results of example images in Figure1.

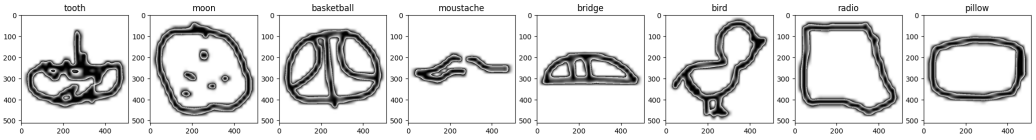

Figure 2: Examples of processed images in the dataset

2.3 Caption Processing

In the loaded dataset, each image has one word showing the category it belongs to. For our project, we want the model to learn the style of human-drawn scribbles. Therefore, we enhance the label by adding "a scribble of " before category names. Then we apply them to the pre-trained text tokenizer in Stable Diffusion v1-5 Model [9]. We will discuss the function and usage of text tokenizers in the following paper.

3 Solution

We fine-tune the model with the method called Low-Rank Adaptation of Large Language Models (LoRA), a memory-efficient way of fine-tuning large models with a small number of trainable parameters. This is introduced for fine-tuning large language models, but the methodology also works for fine-tuning diffusion models. In this section, we will show how we implement this method to generate human scribble-style images by first introducing the structure of the stable diffusion model, and then where and how we add LoRA weights to the base diffusion model.

3.1 Stable Diffusion Structure

3.1.1 Noise scheduler

The schedule functions, denoted Schedulers in the library take in the output of a trained model, a sample on which the diffusion process is iterating, and a timestep to return a denoised sample. That's why schedulers may also be called Samplers in other diffusion model implementations[10].

(1) Schedulers define the methodology for iteratively adding noise to an image or updating a sample based on model outputs.

   (a) adding noise in different manners represents the algorithmic processes to train a diffusion model by adding noise to images.

   (b) for inference, the scheduler defines how to update a sample based on an output from a pre-trained model.

(2) Schedulers are often defined by a noise schedule and an update rule to solve the differential equation solution.

3.1.2 Text encoder

Contrastive Language-Image A pre-training approach or model based on contrastive text-image pairings is called pre-training. A multimodal model called CLIP[11] is based on contrastive learning. In contrast to specific contrastive learning techniques in CV like moco and simclr, CLIP's training data consists of a picture and its associated text description. The model is intended to learn the matching connection between text-image pairs using contrastive learning. The text encoder and image encoder are two models that CLIP offers, as may be seen in the image below. The text transformer model, which is often used in NLP, may be used with the text encoder to extract text features and the image encoder to obtain picture features.

(1) Contrastive Pre-training
This performs contrastive learning on the extracted text features and image features. The CLIP model[12] will forecast the similarity of N^2 potential text-image pairings for a training batch including N text-image pairs, incorporating N text features and N image features in pairs, where the similarity is determined directly. The matrix displayed in the following graphic represents the cosine similarity of text features and image features. There are N total positive samples, text and image that truly belong together as a pair (diagonal elements in the matrix), and $N^2 - N$ total negative samples. Accordingly, the training objective of CLIP is to maximize the number of positive samples while reducing the similarity of the negative samples.

Algorithm 1 Contrastive Pre-training

I_f = image_encoder(I) ▷ Extract image features
T_f = text_encoder(T) ▷ Extract text features

I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

logits = np.dot(I_e, T_e.T) * np.exp(t) ▷ Compute scaled cosine similarity: [n, n]

labels = np.arrange(n)
$loss_i$ = cross_entropy_loss(logits, labels, axis=0) ▷ Symmetric contrastive learning loss
$loss_t$ = cross_entropy_loss(logits, labels, axis=1)
loss = $(loss_i + loss_t)/2$

(2) Create dataset classifier from label text
Construct the description text of each category according to the classification label of the task, and then send these texts to the Text Encoder to obtain the corresponding text features. If the number of categories is N, then N text features will be obtained.

(3) Use for zero-shot prediction
Send the image to be predicted to the Image Encoder to obtain image features, and after calculating the scaled cosine similarity with N text features (consistent with the training process), choose the category corresponding to the text with the highest similarity as the image classification prediction result. These similarities can also be considered as logits. After being sent to softmax, the predicted probability of each category can be obtained.

3.1.3 VAE

(1) Introduction
Variational auto-encoder (VAE)[13] is an important class of generative model. VAE uses the Encoder to efficiently encode the input (we use pictures as the input here), and then the Decoder uses the code to restore the picture. Ideally, the restored output picture should be very similar to the original picture. It can be roughly divided into two parts, Encoder and Decoder. For

the input picture, the Encoder will extract the code: a mean vector and a deviation vector, and then use this code (two vectors) as the input of the Decoder, and finally output a picture similar to the original picture.

(2) Formula deduction
Formula definition: $max(L) = \sum_x logP(x)$ while P(x)=$\int_z^x P(z)P(x|z)dz$
Objective function: $max(L) = \sum_x logP(x) = \sum_x log \int_z^x P(z)P(x|z)dz$

(3) Training process: adjust Encoder/Decoder to increase L_b

$$logP(x) = L_b + KL(q(z|x)||P(z|x))$$

(4) Loss function of actual training

$$max_{loss} = -loss_1 + loss_2 = -KL(q(z|x)||P(z)) + \int q(z|x)logP(x|z)dz$$
$$\simeq \sum_{i=1}^{3}(exp(\sigma_i) - (1+\sigma_i) + (m_i)^2) + 1/L \sum_{l=1}^{L} logP(x^{(i)}|z^{(i,l)})$$

3.1.4 Unet

(1) Unet noise predictor with text
Let's look at a diffusion Unet, whose inputs and outputs would look like the form showing in Figure3[14]:

(a) The Unet is a series of layers that work on transforming the latent array.

(b) Each layer operates on the output of the previous layer.

(c) Some outputs are fed (via residual connections) into the processing later in the network.

(d) The timestep is transformed into a time step embedding vector, and that's what gets used in the layers.

(e) Attention layer is added between the ResNet blocks. Note that the resnet block doesn't directly look at the text. But the attention layers merge those text representations in the latent. And now the next ResNet can utilize that incorporated text information in its processing. (Attention layer will be further explained in the following paragraph.)

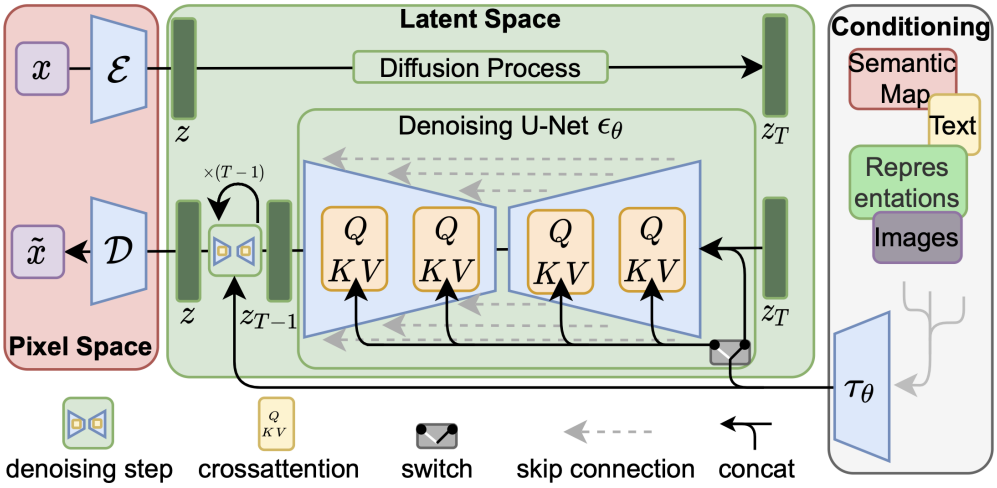

Figure 3: Neural Network of Unet

(2) Attention layer

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt[2]{d_k}})V$$

$Q_{M*d}$is query, and K_{N*d} is key. Among them, the query can be regarded as M vectors with dimension d (expressed by sequence vectors with length M), and key can be regarded as N vectors with dimension d (expressed with sequence vectors with length N expression) spliced together.

(a) Parameters in the Attention mechanism[15]

i. Map q, k and v to the linear transformation matrices $W^Q(d_{model}*d_q)$, $W^K(d_{model}*d_k)$, $W^V(d_{model}*d_v)$ of Q, K and V respectively.

ii. Map the output expression O to the linear transformation matrix $W^O(d_v*d_{model})$ of the final output o.

(b) Cross-attention

The q of cross-attention[15] represents the current sequence, k and v are the same input, corresponding to the encoded sequence, that is, the output result of the last layer of the encoder (for each layer of the decoder side, k and v remain unchanged).

And each layer of the linear mapping parameter matrix is independent, so Q, K, and V after mapping are different. The goal of model parameter optimization is to map q, k, and v to a new high-dimensional space, so that each layer Q, K, and V capture the relationship between q, k, and v at different levels of abstraction. Generally speaking, the underlying layer captures more lexical-level relationships, while the higher-level layer captures more semantic-level relationships.

(c) The role of Attention

Only self-attention exists in the Encoder part, while self-attention and cross-attention[15] exist in the Decoder part.

Self-attention: The query, key, and value of the self-attention in the encoder all correspond to the source sequence (that is, A and B are the same sequences), and the query, key, and value of the self-attention in the decoder corresponding to the target end sequence.

Cross-attention: The cross-attention query in the decoder corresponds to the target sequence, and the key and value correspond to the source sequence (the cross-attention in each layer uses the final output of the encoder).

(d) Multi-head Attention

As it is shown in Figure4[15], attention maps query and key to the same high-dimensional space to calculate similarity, and the corresponding multi-head attention maps query and key to different subspaces α of high-dimensional space (α_1,α_2,...,α_h) to calculate similarity.

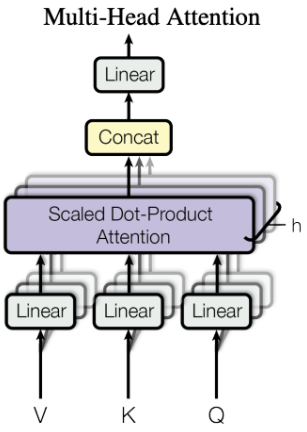

Figure 4: Multi-Head Attention consists of several attention layers running in parallel.

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to $d_q, d_k and d_v$ dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure4.
Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h) * W^O$$
$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^K)$$

(e) Applications of Attention in our Model
Our model mainly makes three separate applications of multi-head attention:

i. In "encoder-decoder attention" layers, the memory keys and values are derived from the encoder output, while the queries are derived from the preceding decoder layer. As a result, the decoder's locations may all pay attention to every position in the input sequence. In sequence-to-sequence models, this mirrors the conventional encoder-decoder attention processes.

ii. Self-attention layers are present in the encoder. All of the keys, values, and queries in a self-attention layer originate from the same source, in this example, the encoder's output from the previous layer. Each place in the encoder has access to every position in the layer below it.

iii. The decoder's self-attention layers work similarly, allowing each position to pay attention to all positions up to and including it. To maintain the auto-regressive characteristic of the decoder, we must prohibit leftward information flow. By masking away all values in the softmax's input that correspond to illicit connections, we implement this within scaled dot-product attention. See Figure4.

3.2 Finetuning with LoRA

3.2.1 Update Matrices

Our method fine-tunes the parameters in the Unet in the decoder. As is depicted by Figure 3, vectors for text and image meet in the cross-attention layer. By updating parameters in the cross-attention layer, the model could be trained to generate images with certain features given specific words. We followed the training method by Hu&Shen[16]. For weight matrix $W_0 \in R^{d\times k}$ in each cross attention layer, we represent it by decomposing the update into two low-rank matrices, where $B \in R^{d\times r}$, $A \in R^{r\times k}$, rank $r \ll min(d, k)$.

$$W_0 + \Delta W = W_0 + BA$$

For $h = W_0 x$, now the new forward pass returns:

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

The loss function used during LoRA training is the same as that for training the noise predictor for a text-to-image diffusion model. We calculate the mean squared error between the predicted noise and the sampled noise by the noise scheduler.

3.2.2 Hyperparameters

During training, we fix the original weight W_0, and only update matrices A and B. We initialize A with a random Gaussian, and set B to all 0. For scheduler and optimizer, we use the cosine

scheduler from DDPM[17] and AdamW, similar to the recommended setup on hugging face [18]. We sampled 2000 images from the dataset, and ran 70 epochs with batch size at 8 and gradient accumulation step at 4. The number of total optimization steps is 4410. We used a single Nvidia RTX8000 GPU and spent over six hours on training. The LoRA model checkpoint takes only 3.3 MB of storage, which is significantly smaller than the 4.27GB size of stable-diffusion-v1.5.

4 Results and Discussion

4.1 LoRA Model Inference

We save our trained LoRA weights to a file. During image generation, we add the weights on top of the base model weights. When merging the LoRA weights with the frozen pre-trained weights, we can adjust the scale of LoRA weights. Setting the scale to 0 not using LoRA, while setting the scale to 1 means using LoRA only. In Figure 5, we plot the results generated from different seeds, different LoRA scales and the same prompt "a scribble of apple", using stable-diffusion-v1.5 as the base model.

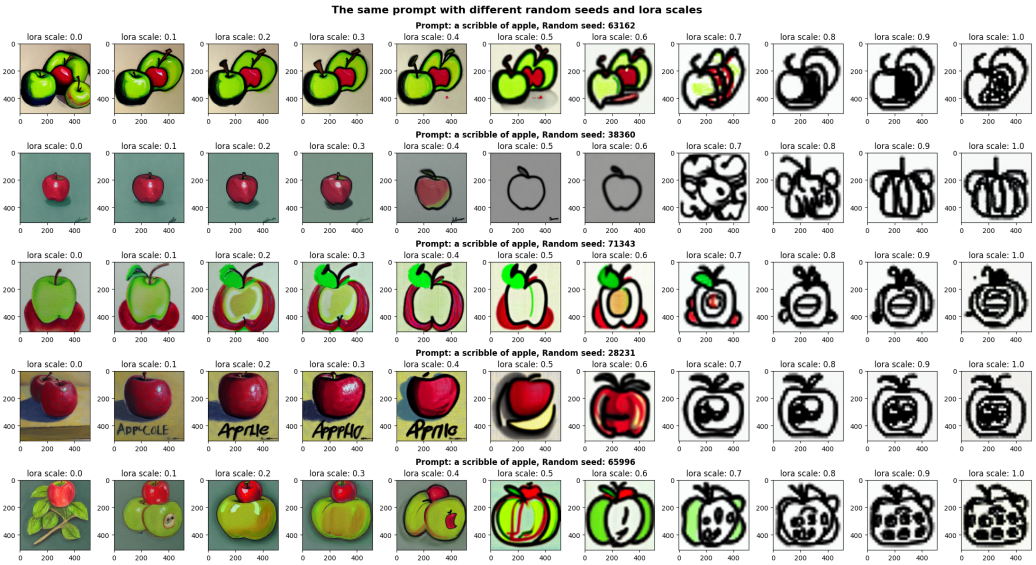

Figure 5: Generated images of the prompt "a scribble of apple"

When the LoRA scale is set to 0, the generated images are black low resolution strokes drawn on pure white background, which matches the style of images we used for training. We find the result from the LoRA scale equal to 0.5 and 0.6 visually pleasing.

4.2 Model Extensibility

4.2.1 Generate with Simple Concepts

During training, we only selected 100 out of 345 categories. To test the model's extensibility, we first compare the generation results of trained prompts and untrained prompts. Figure 6 shows images generated with categories in the training set, with prompts in the format of "a scribble of " + category name. Figure 7 shows images generated with categories outside of the training set, also with prompts in the format of "a scribble of " + category name.

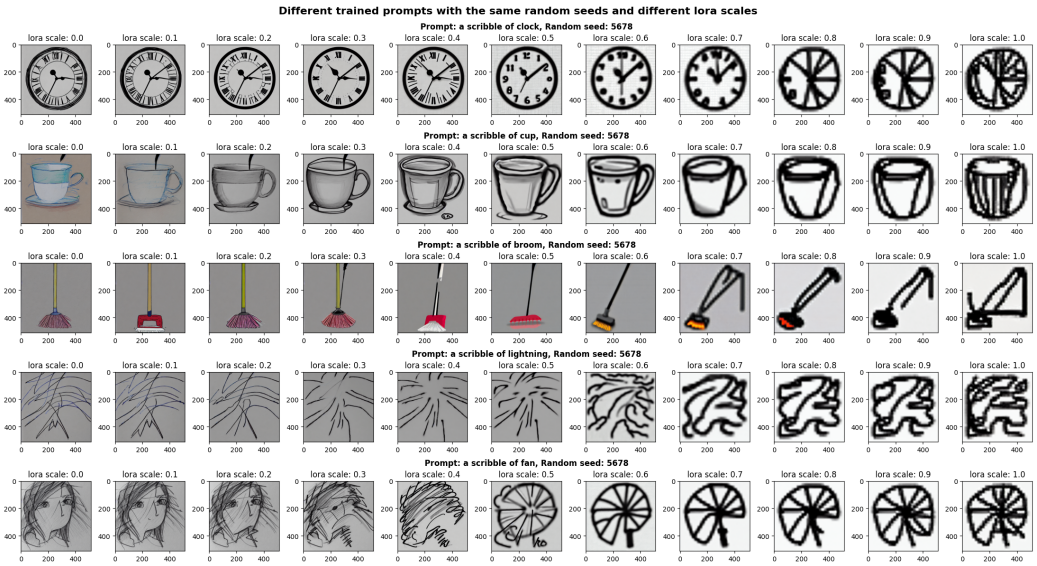

Figure 6: Generated images with categories in the training set, with prompts in the format of "a scribble of " + category name

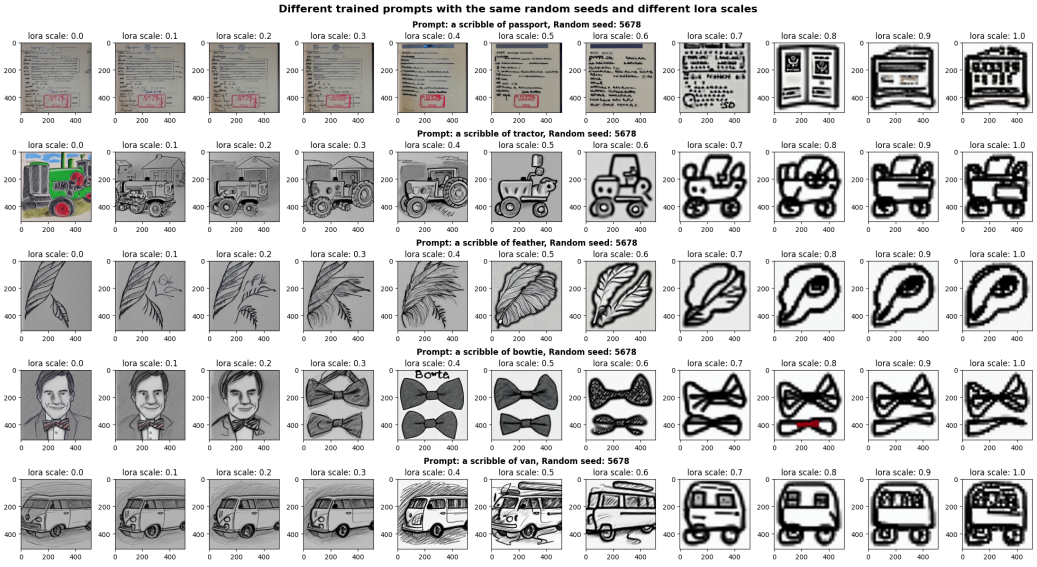

Figure 7: Generated images with categories outside of the training set, with prompts in the format of "a scribble of " + category name

Notice that in Figure6, for prompt "a scribble of fan", the base stable-diffusion-v1.5 model misinterprets the prompt, and draws a human face. The LoRA weight, on the other hand, guided the model back to generating images of a fan. But for "a scribble of lightning", neither the base model nor the LoRA generates proper images.

4.2.2 Generate with Complex Concepts

We test the model with longer and more complex prompts, containing multiple concepts. For example, "Cabin in the woods with the forest fire in the background and smoke". Figure8 shows images generated from complex prompts with "a scribble of " inserted in the front, in order to test

how the prompt "a scribble of " would affect the style of the image. Figure9 shows images generated from complex prompts without "a scribble of". The generation seed for these two figures is the same.

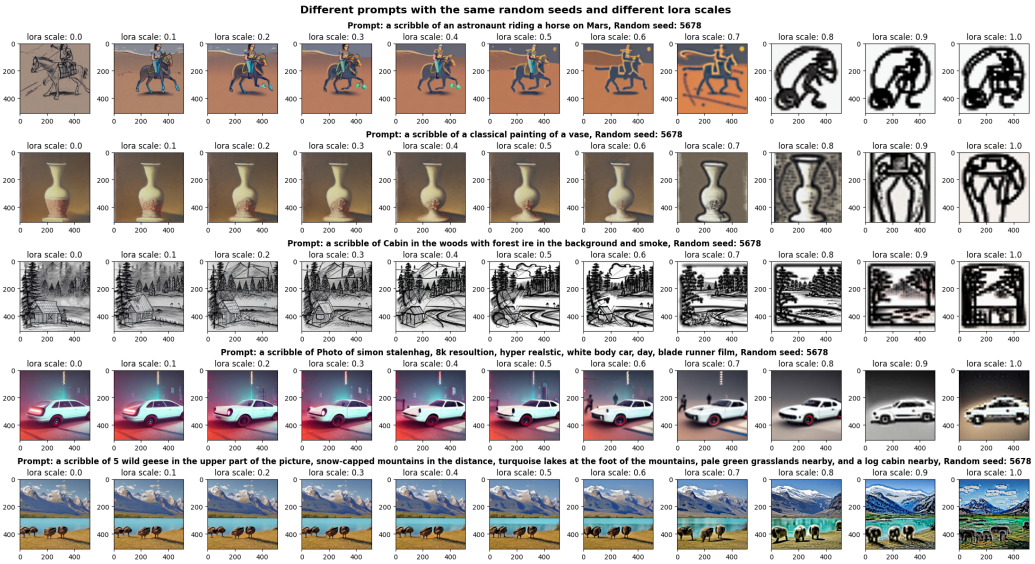

Figure 8: Image generated from complex prompts with "a scribble of " in the front

Figure 9: Image generated from complex prompts without "a scribble of "

Taking a look at Figure8 only, even when the LoRA scale is set to 1, some images are still coloured, and look different from human scribbling. The effect of our extra LoRA weights becomes visually noticeable only after the scale is over 0.7. The effect of our LoRA weight on image generation could be noticed by comparing images generated from the same prompt in Figure8 with Figure9. Without prompt "a scribble of", we can only notice the impact of LoRA weight when the scale is over 0.9. Also, through comparison, we can tell that the image style with LoRA weight is related to the image style generated with the base model only. For prompts "a scribble of Cabin in the woods with the forest fire in the background and smoke" and "Cabin in the woods with the forest fire in

the background and smoke", the stable-diffusion-v1.5 model generates a hand-drawn style for the former prompt, and a photo-realistic style for the latter prompt. This leads to a scribble-like result after adding LoRA weights for the former prompt.

4.3 Benchmark Evaluation

4.3.1 CLIP score

To get a quantitative evaluation for our text-to-image model, we adopt CLIP score [19] as one of the benchmarks. It measures the semantic similarity of image-caption pairs. The higher clip score implies higher compatibility [20]. In our case, we used the CLIP score function in the pre-trained OpenAI/clip-vit-base-patch16 model [11], and calculate clip scores based on ten randomly selected prompts. In figure 10, we plot the CLIP scores of different LoRA scales. The blue curve shows average CLIP scores for prompts included in the training set. The orange curve shows average CLIP scores for prompts in the format of "a scribble of " + category, while all categories are outside of the training set. The generator reed is fixed for all images. For reference, we randomly select images and captions (also in the format of "a scribble of " + category) from our ground truth dataset and calculate the CLIP score.

When LoRA scale becomes higher, the CLIP score for both trained and untrained prompts gets lower, and CLIP scores for trained prompts are lower than ones for untrained prompts in general. The reason probably lies in that the CLIP score is not primarily trained one-hand drawn scribbles by ordinary people. Even images drawn by humans get a low CLIP score. However, we cannot conclude that the lower CLIP score shows that our LoRA model generated images close to the style of the quick draw dataset, because poorly compatible image-label pairs also result in low CLIP scores. Therefore, in our case, using the CLIP score cannot properly evaluate our model.

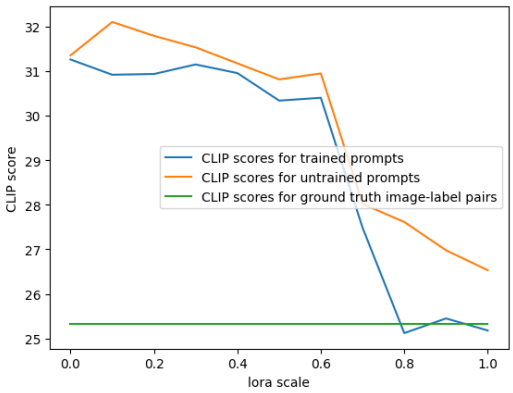

Figure 10: CLIP scores of different LoRA scales

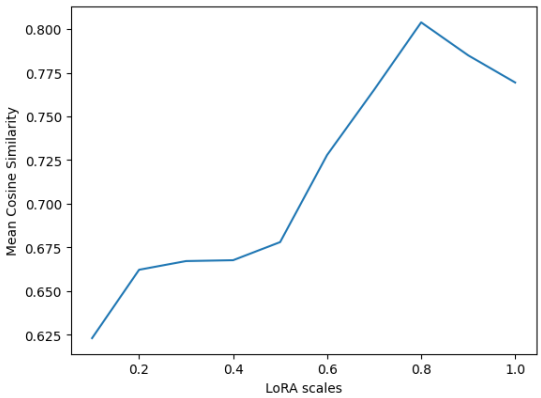

Figure 11: Similarity of different LoRA scales

4.3.2 Cosine Similarity

To better evaluate our model performance, we also calculated the cosine similarity between generated images and ground truth images. We cannot compute image cosine similarities directly because the dimension of each image is 512x512x3. We need to encode images into vectors. We use an image processor and pre-trained image encoder in the OpenAI/clip-vit-large-patch14 checkpoint [12], the same as that used during the training process of stable-diffusion-v1.5 model. We randomly pick ground truth images, and generate them with prompts in the format of "a scribble of " + category, using stable-diffusion-v1.5 and our LoRA model. After encoding both generated and ground truth images, we get feature vectors and calculate their cosine similarity. Figure11 shows cosine similarity scores for different LoRA weights. For each LoRA scale, we generate twenty different images and take the mean value of cosine similarity scores. The cosine similarity grows as the LoRA scale gets larger, and reaches the peak when the LoRA scale equals to 0.8. Then cosine similarity drops at the

LoRA scale of 0.9 and 1. Setting the LoRA scale at 0.8 reaches the balance of both guiding towards the style learned from the training set and utilizing the semantic interpretation ability of the large pre-trained stable diffusion model.

4.4 Discussion

In our project, we successfully fine-tuned our model to generate images of a specific style. We generated quite a few visually satisfying images using the stable-diffusion-v1.5 model with our trained LoRA model added on top. Also, our model successfully guides generation results closer to ground truth images.

For further research, we can focus on the selection of hyperparameters that can make training more efficient:

(1) In our project, we use the cosine noise scheduler during the training process. Other noise schedulers could be tested.

(2) During the training process, we saved checkpoints every 500 iterations. We could compare the model performance of each checkpoint, and find out the optimal total training steps.

(3) Given the relatively small number of trainable parameters in the LoRA model, it doesn't require a large amount of training data. We could compare the result with different training data sizes, and see how quickly the model converges.

(4) Although we train the LoRA model on stable-diffusion-v1.5, we can apply it to two other base models with the same structure. We could test the performance of the LoRA model paired with other base models.

References

[1] "DALL·E 2 — openai.com," https://openai.com/product/dall-e-2.

[2] E. Mostaque, "Stable Diffusion Public Release — Stability AI — stability.ai," https://stability.ai/blog/stable-diffusion-public-release.

[3] "Midjourney — midjourney.com," https://www.midjourney.com/.

[4] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," 2022.

[5] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021.

[6] "Diffusers — huggingface.co," https://huggingface.co/docs/diffusers/main/en/index.

[7] "GitHub - AUTOMATIC1111/stable-diffusion-webui: Stable Diffusion web UI — github.com," https://github.com/AUTOMATIC1111/stable-diffusion-webui.

[8] "GitHub - googlecreativelab/quickdraw-dataset: Documentation on how to access and use the Quick, Draw! Dataset. — github.com," https://github.com/googlecreativelab/quickdraw-dataset.

[9] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 10 684–10 695.

[10] [Online]. Available: https://huggingface.co/docs/diffusers/v0.9.0/en/api/schedulers

[11] "openai/clip-vit-base-patch16 · Hugging Face." [Online]. Available: https://huggingface.co/openai/clip-vit-base-patch16

[12] "openai/clip-vit-large-patch14 · Hugging Face." [Online]. Available: https://huggingface.co/openai/clip-vit-large-patch14

[13] C. Doersch, "Tutorial on variational autoencoders," 2021.

[14] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[16] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.

[17] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," 2021.

[18] "Low-Rank Adaptation of Large Language Models (LoRA)." [Online]. Available: https://huggingface.co/docs/diffusers/main/en/training/lora

[19] J. Hessel, A. Holtzman, M. Forbes, R. L. Bras, and Y. Choi, "Clipscore: A reference-free evaluation metric for image captioning," 2022.

[20] "Evaluating Diffusion Models." [Online]. Available: https://huggingface.co/docs/diffusers/main/en/conceptual/evaluation