

COMP7065 Innovative Laboratory

Fine-tuning Stable Diffusion with LoRA

After completion of this lab, students will be able to

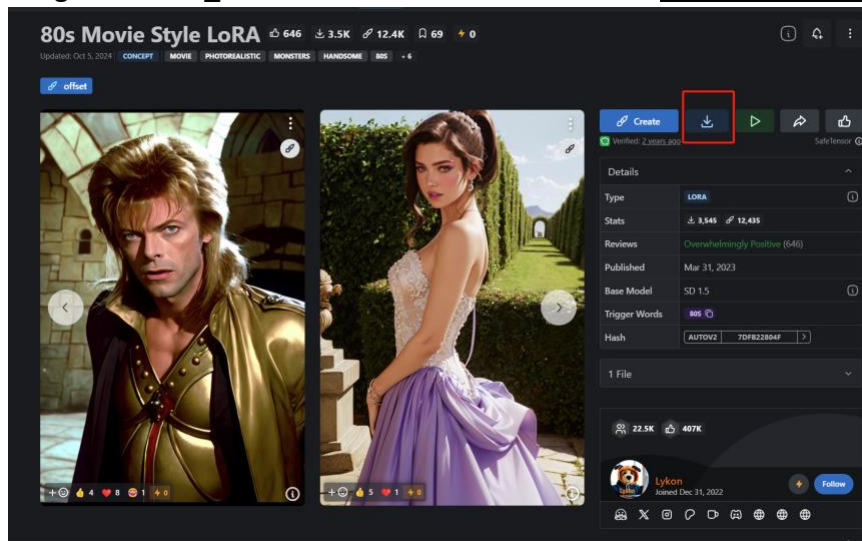
1. Apply existing LoRAs to facilitate their own AI art design;
2. Describe how the LoRA fine-tuning custom node works;
3. Perform LoRA fine-tuning on stable diffusion models within ComfyUI;
4. Construct their own dataset for LoRA fine-tuning

Table of Contents

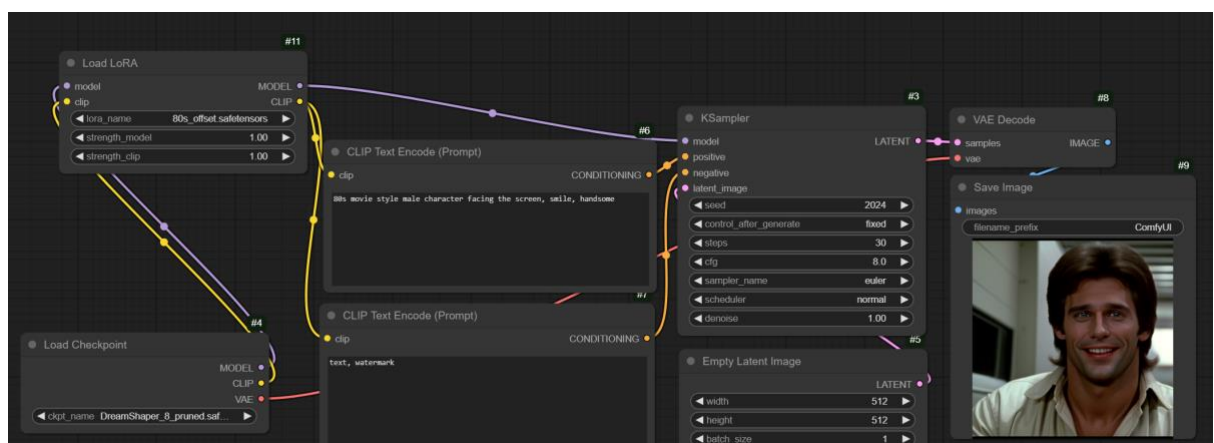
1. Apply existing LoRA: 'Load LoRA' node.....	2
2. Create your own LoRA: 'SimpleLoRA' node	5
3. Exercise	15

1. Apply existing LoRA: 'Load LoRA' node

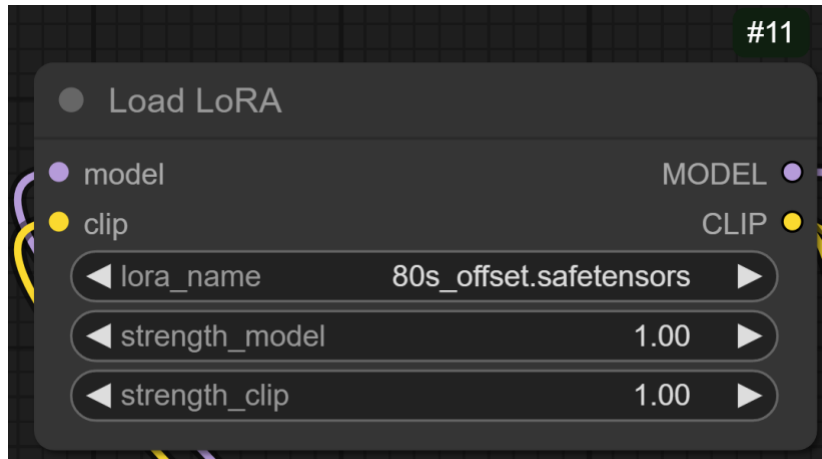
LoRA stands for **Low Rank Adapter**. It is a parameter-efficient fine-tuning method for neural network models. To be more straightforward, LoRAs are small-sized 'add on' of model weights that allow you to efficiently customize the base stable diffusion model. For example, you can use LoRAs to change the image style, specify a character you like, etc. In this part, you will learn how to apply an existing LoRA to a stable diffusion model, and understand its effect in an intuitive manner. We will use the **'80s Movie Style LoRA'** in this showcase. Please download the LoRA from the [release website](#) on CivitAI, and place its weight file '80s_offset.safetensors' into the folder **'ComfyUI/models/loras'**.



In ComfyUI, the **'Load LoRA'** node provides us with a convenient way to utilize an existing LoRA. Below is showcase workflow that loads the **'80s Movie Style LoRA'** and inserts it into the pruned Dreamshaper 8 model. As presented, the man in the output image looks just like a character from an old film.

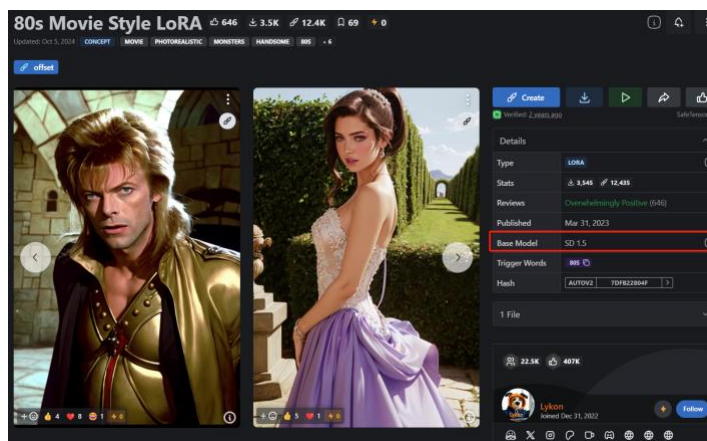


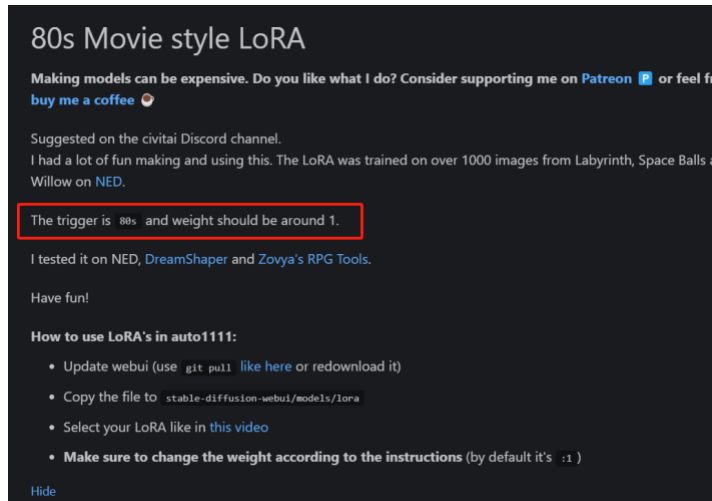
Now let's investigate the settings of 'Load LoRA' node. The 'lora_name' specifies which LoRA should be loaded. The 'strength_model' and 'strength_clip' adjust the weight of LoRA applied to UNet and CLIP respectively. The range of two values is usually between 0 and 1. In the workflow, however, you are also allowed to set the two strengths to negative values so as to remove the effect from LoRA.



Supplements: You should make sure that **the LoRA matches the base model**. For example, a LoRA for stable diffusion 1.5 models cannot be applied to a stable diffusion XL model. Additionally, LoRAs are usually trained with some trigger words. To make LoRA effective, you may need to **input these trigger words into the positive textual prompts**.

In general, you can find the information of trigger words and the expected base model from the release website of the LoRA. As shown below, the '80s Movie Style LoRA' matches the '**stable diffusion 1.5**' (SD1.5) model, and the trigger word is '**80s**'. Sometimes you may also find the values of 'strength_model' and 'strength_clip' recommended by the LoRA's author, but you are always encouraged to try them by yourselves and find the most suitable ones.





It's also noteworthy that **the base model in the showcase doesn't have to be exactly the original Stable Diffusion 1.5**. Other base models like Dreamshaper or DarkSushi are also supported, as long as they maintain the same network architecture with **Stable Diffusion 1.5**. Below is an example generated by [DarkSushi 2.5D](#) together with the '80's Movie Style LoRA'.



To make sure that the LoRA indeed had an effect on the output, you can remove the 'Load LoRA' node, run the generation process again while keeping other settings unchanged. Below is an example generated without the LoRA. It shows that without LoRA the style of the figure greatly differs from the one given in showcase, and thereby we can conclude that the 80's Movie Style LoRA is effective enough.



2. Create your own LoRA: 'SimpleLoRA' node

Despite that there are thousands of LoRAs available in the AI art community (e.g. [CivitAI](#)), sometimes you may still be unable to find a LoRA that suits your art style. In this case you may need to LoRA fine-tune the base model by yourself. We will explore how to perform LoRA fine-tuning completely inside ComfyUI without any programming. This can be achieved by utilizing the **SimpleLoRA node**.

Preparations: In this showcase, you will learn how to **fine-tune the pruned Dreamshaper 8 model with a Huggingface dataset**. We will use the **'pokemon-blip-captions-en-zh'** dataset as the training set, and perform LoRA fine-tuning with the **SimpleLoRA** node.

Dataset: You can preview the 'pokemon-blip-captions-en-zh' dataset on HuggingFace through this link: <https://huggingface.co/datasets/svjack/pokemon-blip-captions-en-zh>. In the showcase, **the Pokemon dataset will be downloaded automatically**, so it's not necessary for you to manually download it from HuggingFace.

The image shows the HuggingFace dataset card for 'pokemon-blip-captions-en-zh' by user svjack. The card includes metadata such as 'Text-to-Image' tasks, 'Image' and 'Text' modalities, 'parquet' format, and 'English' and 'Chinese' languages. It also shows a license of 'cc-by-nc-sa-4.0' and 42 likes.

Below the card is the 'Dataset Viewer' interface, which displays a table of dataset rows. The table has three columns: 'image', 'en_text', and 'zh_text'. The first row shows a drawing of a green Pokemon with red eyes, with the English caption 'a drawing of a green pokemon with red eyes' and the Chinese caption '红眼睛的绿色小精灵的图画'.

image	en_text	zh_text
	a drawing of a green pokemon with red eyes	红眼睛的绿色小精灵的图画
	a green and yellow toy with a red nose	黄绿相间的红鼻子玩具
	a red and white ball with an angry look on its face	一个红白相间的球，脸上带着愤怒的表情
	a cartoon ball with a smile on its face	笑容满面的卡通球

Dataset Card for Pokémon BLIP captions with English and Chinese.

SimpleLoRA: To enable LoRA fine-tuning in ComfyUI, we will need to utilize the custom node 'SimpleLoRA'. You can access this node through this link on Github: https://github.com/sifengshang/ComfyUI_SimpleLoRA

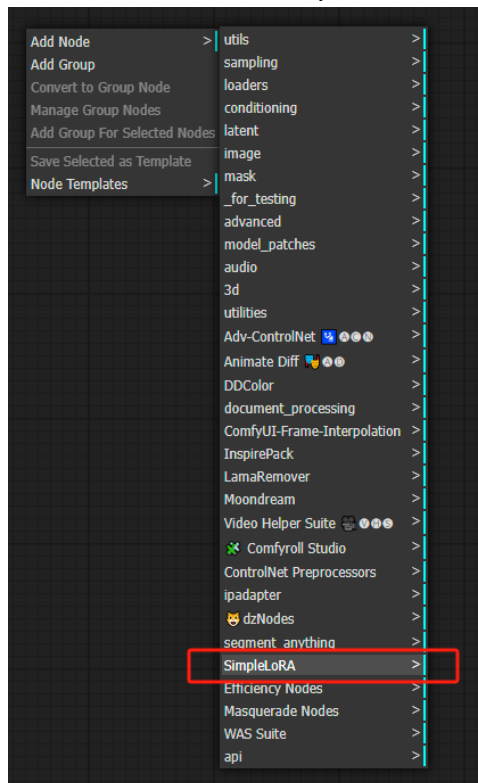
On the main website, I've explained how to install the node to your ComfyUI. You should first download the source codes of SimpleLoRA from the Github website, and place it into the folder for custom nodes: '**ComfyUI\custom_nodes**'.

__pycache__	2025/2/2 16:09
canvas_tab	2025/2/2 16:59
cg-training-tools-main	2025/2/16 22:34
ComfyUI_Comfyroll_CustomNodes	2025/2/16 22:41
comfyui_controlnet_aux	2025/2/10 20:36
comfyui_ipadapter_plus	2025/2/2 16:15
comfyui_layerstyle	2025/2/11 18:05
comfyui_segment_anything	2025/2/23 19:56
ComfyUI_SimpleLoRA	2025/3/2 11:10
comfyui-advanced-controlnet	2025/2/11 18:05
comfyui-animatediff-evolved	2025/2/11 18:05
ComfyUI-DDColor	2025/2/23 11:16
ComfyUI-Documents	2025/2/11 18:05

Next, **please close your ComfyU window and terminal**. You need to enter the folder 'ComfyUI_SimpleLoRA', and double click the batch file '**install_requirements.bat**'. It will automatically download and configure the Python dependencies of the node.

__pycache__	2025/3/2 11:10
__init__.py	2025/3/2 10:49
install_requirements.bat	2025/3/2 10:50
lora_core.py	2025/3/2 15:52
requirements.txt	2025/3/2 10:50
SimpleLoRA.py	2025/3/2 10:49

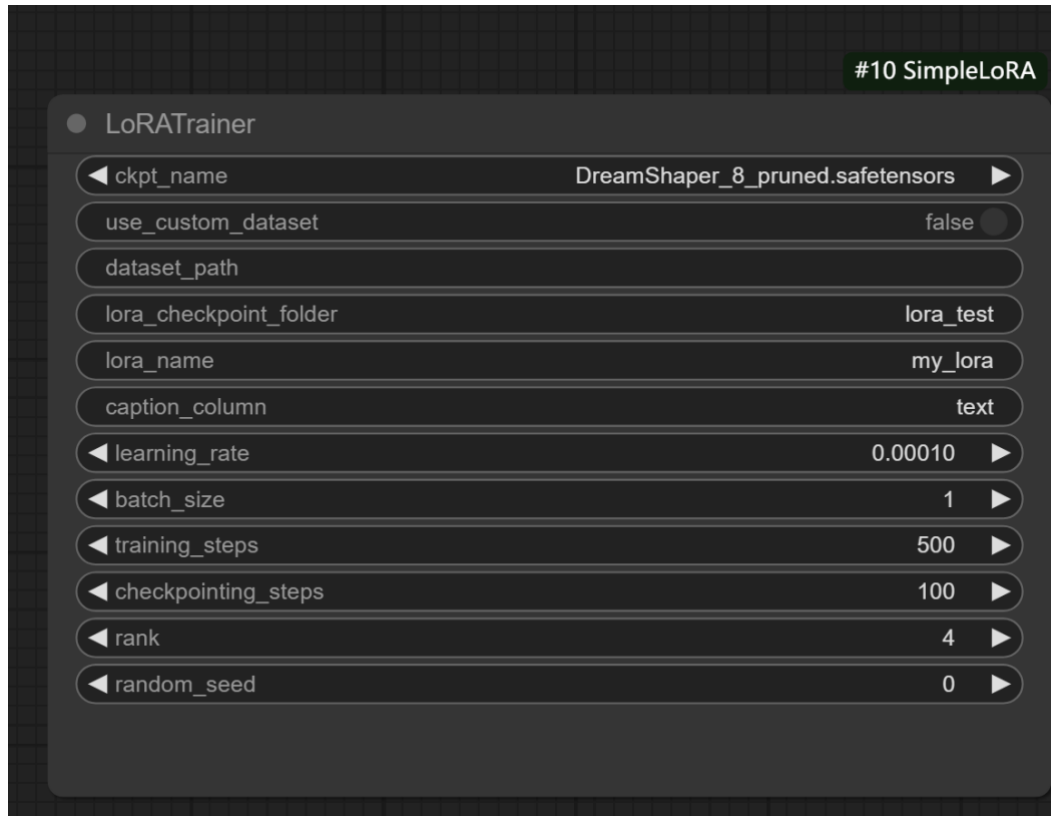
To check whether the installation is successful, you can restart ComfyUI, right click and select 'Add Node'. If you can see a selection of 'SimpleLoRA', then the custom node has been indeed successfully installed.



I would be very grateful if you can give me a star on Github 😊.

Fine-tune Dreamshaper 8 with SimpleLoRA

To utilize the node for LoRA fine-tuning, you should first **load a completely empty workflow** in ComfyUI, then **find the 'SimpleLoRA' node and add it to the workflow**. The node can be found by: **'Add Node'-'>'SimpleLoRA'-'>'LoRATrainer'** in turn. In other words, the workflow for LoRA fine-tuning with SimpleLoRA should **contain only a single node (LoRATrainer) without any connections**, which should look like this:



This node enables you to LoRA fine-tune the base stable diffusion model without any coding! All you need to do is to configure the settings of the node properly, and then run the workflow by clicking the 'Queue' button.

The following part is a showcase teaching you **how to LoRA fine-tune the pruned Dreamshaper 8 on 'pokemon-blip-captions-en-zh' dataset through the SimpleLoRA node**. After applying the trained LoRA, the pokemon image generated by the base model will become more cartoon style. From this showcase you can **learn the meaning of each setting in SimpleLoRA node** and how to configure them reasonably based on your need.

❖ 'ckpt_name'

This setting indicates the **'checkpoint name'** of the base model to be fine-tuned. It allows you to select a model from the folder 'ComfyUI/models/checkpoints'. In this showcase our base model is the pruned Dreamshaper 8, **so we can just set the 'ckpt_name' to 'Dreamshaper_8_pruned.safetensors'**.

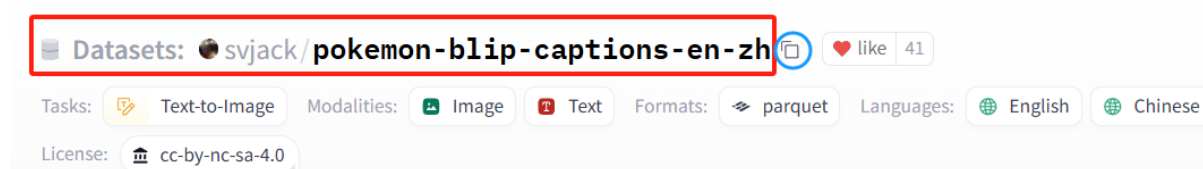
❖ 'use_custom_dataset' and 'dataset_path'

The two settings are combined together to indicate which dataset will be used for LoRA fine-tuning. To be more specific, when set **'use_custom_dataset' to True**, the node will **load a custom dataset constructed by yourself**, and **the folder path of the dataset will be specified by the setting 'dataset_path'**. You will learn more details about how to build a custom dataset in the exercise. On the other hand, when setting **'use_custom_dataset' to False**, the node will automatically download a HuggingFace dataset and use it for fine-tuning. **The name of the dataset is specified by 'dataset_path'**.

In this showcase we will use a HuggingFace dataset 'pokemon-blip-captions-en-zh' as the training set. So we should **set the 'use_custom_dataset' to False**. As for the **'dataset_path'**, you can refer to the release website of 'pokemon-blip-captions-en-zh' dataset on HuggingFace:

<https://huggingface.co/datasets/svjack/pokemon-blip-captions-en-zh>

On the website you can see **the name of dataset is 'svjack/pokemon-blip-captions-en-zh', which is exactly what we should put for the 'dataset_path'**. You can also directly copy the dataset name to clipboard by clicking the copy icon (**circled in blue**) and then paste it to **'dataset_path'**.



Supplements: In general, any **text-to-image dataset** on HuggingFace can be supported.

❖ 'lora_checkpoint_folder' and 'lora_name'

During the LoRA fine-tuning process, the node will save the trained LoRA weights periodically into the folder: 'ComfyUI/models/lora_logs/[lora_checkpoint_folder]'. For example, if we set the 'checkpoint_dirname' to 'lora_test', then the trained LoRA weights will be saved into the folder: 'ComfyUI/models/lora_logs/lora_test'. Note that, this folder will be automatically created by the node during training so you don't have to create it by yourself manually. And the filename of the LoRA weights in '.safetensors' format will be '[lora_name].safetensors'. For example, when setting 'lora_name' to 'my_lora', the filename will be 'my_lora.safetensors'.

In this showcase, **we can set the 'lora_checkpoint_folder' to 'lora_showcase', and 'lora_name' to 'pokemon_lora'**.

As a result, a folder in 'ComfyUI/models/lora_logs/lora_showcase' will be created, and the trained lora weights will be named as 'pokemon_lora.safetensors'.

❖ 'caption_column'






Before explaining the meaning of this setting, we should understand some basics about the dataset. We've learnt from the lecture that to train a neural network, one must prepare many training data organised as a dataset. In our showcase, **you may imagine the whole dataset as a big Excel table**, as shown in the illustration below. In this table, item in each row consists of a single **image** and at least one **caption**. The **caption** is exactly the positive textual prompts. And the base model is trained to generate exactly the content in the **image** when given the **caption**. Sometimes the training data may have multiple caption columns. As shown in the illustration below, for each image there are both English captions '**en_text**' and Chinese captions '**cn_text**'. In this case **we need to specify which caption column is used for training by specifying the 'caption_column' in the node**.

It should be noted that there are different ways to set 'caption_column'. **When using a custom dataset (i.e., when setting custom_dataset to True), the key of 'caption_column' is always 'text'**. For an existing HuggingFace dataset, you may need to enter its release website and take a look at the columns in the **Dataset Viewer**. In our showcase, the Dataset Viewer of 'pokemon-blip-captions-en-zh' on the [release website](#) shows that there are two columns for captions. One is '**en_text**' and the other is the '**cn_text**'. Since we use English captions for fine-tuning, so we set '**caption_column**' to '**en_text**'.

Dataset Viewer Auto-converted to Parquet </> API Embed Full Screen Viewer

Split (1)
train · 833 rows

Search this dataset SQL Console

image image · width (px)	en_text string · lengths	zh_text string · lengths
	a drawing of a green pokemon with red eyes	红眼睛的绿色小精灵的图画
	a green and yellow toy with a red nose	黄绿相间的红鼻子玩具
	a red and white ball with an angry look on its face	一个红白相间的球，脸上带着愤怒的表情
	a cartoon ball with a smile on it's face	笑容满面的卡通球
		

< Previous 1 2 3 ... 9 Next >

❖ 'learning_rate'

This parameter controls exactly the learning rate during training. A larger learning rate speeds up the training, but it may also cause the loss decrease to be unstable or even lead to diverge. On the other hand, if the learning rate is too small, it may lead to underfitting even after many training steps. In this showcase, **it is recommended to set the learning rate to its default value '0.0001'.**

❖ 'batch_size'

When setting 'batch_size' to 1, only one pair of (image, caption) will be fed to the model for each training step. This may slow down the training process, but will significantly save the cost of GPU memory. If you have enough GPU resources, you can set the 'batch_size' larger than 1, as long as the 'Cuda: out of memory' error is not triggered. In the Lab we will use a single NVIDIA RTX 3050Ti with only 8GB GPU memory for fine-tuning, so **it is recommended to set the 'batch_size' to 1.**

❖ 'training_steps' and 'checkpointing_steps'

The 'training_steps' denotes the total iterations of the training process, and the LoRA weights will be saved every 'checkpointing_steps'. For example, if we set the 'training_steps' to 2000 and 'checkpointing_steps' to 500, this means that the whole training steps will be 2000 and the LoRA weights will be saved every 500 steps.

It should be noted that LoRA trained with more 'training_steps' may be effective, but the training time will also increase proportionally. And if the training steps are too few, the the LoRA weights may be underfitting, resulting in bad quality. However, we cannot set the training steps to a very large value either, because the LoRA weights may suffer from overfitting, still resulting in bad output effect. In this showcase, in order to obtained a trained LoRA quickly, **we can set the 'training_steps' to 1000 and 'checkpoint_steps' to 200.** But you are also encouraged to set them by yourself and see how the quality of LoRA will be affected.

❖ 'rank'

This is a hyperparamter spcifically for the LoRA fine-tuning method. It indicates how many ranks are there for the adapters in each layer. In general, higher ranks may improve the quality of LoRA, but will cost more GPU memory. In our showcase, **it is recommended to set the 'rank' to 4.**

❖ 'random_seed'

Similar to the random seed in the Ksampler, it is an interger number that controls the randomness during training. **We can always set it to the default value 0.**

Below is an illustration showing correct settings of LoRATrainer node in our Pokemon showcase. You may refer to it for a quick check.

The image shows a dark-themed user interface for a LoRA Trainer. At the top right, there is a green badge with the text "#10 SimpleLoRA". Below this, the title "LoRA Trainer" is displayed with a small circular icon to its left. The main area of the interface consists of a list of parameters, each in a horizontal bar with a left arrow, the parameter name, the current value, and a right arrow. The parameters and their values are: ckpt_name (DreamShaper_8_pruned.safetensors), use_custom_dataset (false, with a toggle switch), dataset_path (svjack/pokemon-blip-captions-e), lora_checkpoint_folder (lora_showcase), lora_name (pokemon_lora), caption_column (en_text), learning_rate (0.00010), batch_size (1), training_steps (1000), checkpointing_steps (200), rank (4), and random_seed (0).

Parameter	Value
ckpt_name	DreamShaper_8_pruned.safetensors
use_custom_dataset	false
dataset_path	svjack/pokemon-blip-captions-e
lora_checkpoint_folder	lora_showcase
lora_name	pokemon_lora
caption_column	en_text
learning_rate	0.00010
batch_size	1
training_steps	1000
checkpointing_steps	200
rank	4
random_seed	0

After properly setting the LoRATrainer node, you may press the 'Queue' button to start the training workflow. The training process may take around 15 minutes. And you can **supervise the training progress through the terminal window of ComfyUI.**

```

C:\Windows\system32\cmd.exe
former_layers_per_block', 'time_embedding_dim', 'time_embedding_type', 'resnet_out_scale_factor', 'reverse_transformer
layers_per_block', 'resnet_skip_time_act', 'cross_attention_norm', 'time_embedding_act_fn', 'encoder_hid_dim_type', 'ad
dition_embed_type', 'only_cross_attention', 'dropout', 'mid_block_type', 'projection_class_embeddings_input_dim', 'class
embeddings_concat', 'mid_block_only_cross_attention', 'addition_embed_type_num_heads', 'use_linear_projection', 'upcast
attention', 'time_cond_proj_dim') was not found in config. Values will be initialized to default values.
Loading pipeline components...: 83% [REDACTED] | 5/6 [00:03<00:00, 1.23it/s] [
scaling_factor', 'use_post_quant_conv', 'shift_factor', 'mid_block_add_attention', 'latents_mean', 'force_upcast', 'use
quant_conv', 'latents_std') was not found in config. Values will be initialized to default values.
Loading pipeline components...: 100% [REDACTED] | 6/6 [00:03<00:00, 1.57it/s]
You have disabled the safety checker for <class 'diffusers.pipelines.stable_diffusion.pipeline_stable_diffusion.StableDi
ffusionPipeline'> by passing 'safety_checker=None'. Ensure that you abide to the conditions of the Stable Diffusion lice
nse and do not expose unfiltered results in services or applications open to the public. Both the diffusers team and Huga
ing Face strongly recommend to keep the safety filter enabled in all public facing circumstances, disabling it only for
use-cases that involve analyzing network behavior or auditing its results. For more information, please have a look at
https://github.com/huggingface/diffusers/pull/254 .
#####
#
# Will try to download/use the cached dataset from HuggingFace: svjack/pokemon-blip-captions-en-zh
#
#####
02/16/2025 18:55:37 - INFO - _main - ***** Running training *****
02/16/2025 18:55:37 - INFO - _main - Num examples = 833
02/16/2025 18:55:37 - INFO - _main - Num Epochs = 5
02/16/2025 18:55:37 - INFO - _main - Instantaneous batch size per device = 1
02/16/2025 18:55:37 - INFO - _main - Total train batch size (w. parallel, distributed & accumulation) = 4
02/16/2025 18:55:37 - INFO - _main - Gradient Accumulation steps = 4
02/16/2025 18:55:37 - INFO - _main - Total optimization steps = 1000
Steps: 5% [REDACTED] | 51/1000 [00:56<16:39, 1.05s/it, lr=9.94e-5, step_loss=0.0103]

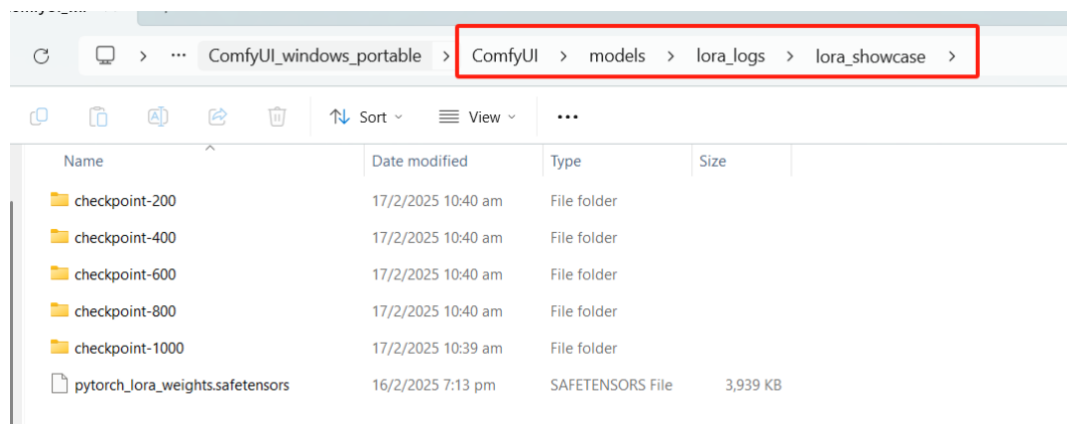
```

Credit to SHANG Sifeng

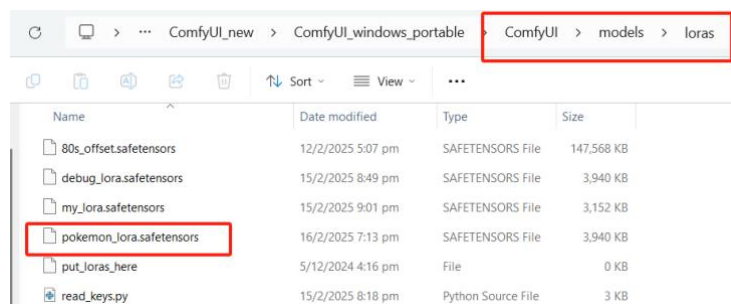
If the training is successful, you may see a message says **“Training is finished”** prompted in the terminal window.

```
C:\Windows\system32\cmd.exe
ComfyUI\models\lora_logs\lora_showcase\checkpoint-1000\optimizer.bin
02/16/2025 19:13:02 - INFO - accelerate.checkpointing - Scheduler state saved in F:\ComfyUI_windows_portable_nvidia_new\ComfyUI\models\lora_logs\lora_showcase\checkpoint-1000\scheduler.bin
02/16/2025 19:13:02 - INFO - accelerate.checkpointing - Sampler state for dataloader 0 saved in F:\ComfyUI_windows_portable_nvidia_new\ComfyUI\models\lora_logs\lora_showcase\checkpoint-1000\sampler.bin
02/16/2025 19:13:02 - INFO - accelerate.checkpointing - Random states saved in F:\ComfyUI_windows_portable_nvidia_new\ComfyUI\models\lora_logs\lora_showcase\checkpoint-1000\random_states_0.pkl
Model weights saved in F:\ComfyUI_windows_portable_nvidia_new\ComfyUI\models\lora_logs\lora_showcase\checkpoint-1000\pytorch_lora_weights.safetensors
02/16/2025 19:13:02 - INFO - __main__ - Saved state to F:\ComfyUI_windows_portable_nvidia_new\ComfyUI\models\lora_logs\lora_showcase\checkpoint-1000
Steps: 100%|███████████████████████████████████████████| 1000/1000 [17:25<00:00, 1.03s/it, 1r=0, step_loss=0.00909] Model weights saved in F:\ComfyUI_windows_portable_nvidia_new\ComfyUI\models\lora_logs\lora_showcase\pytorch_lora_weights.safetensors
Steps: 100%|███████████████████████████████████████████| 1000/1000 [17:25<00:00, 1.05s/it, 1r=0, step_loss=0.00909]
#####
#
# Training is finished!
#
#####
#
# Keys of LoRA's .safetensors file have been changed
# The LoRA can be found and loaded by the LoadLoRAModelONLY Node
# You can find the trained LoRA's .safetensors file at:
# F:\ComfyUI_windows_portable_nvidia_new\ComfyUI\models\loras\pokemon_lora.safetensors
#
#####
Prompt executed in 1067.22 seconds
```

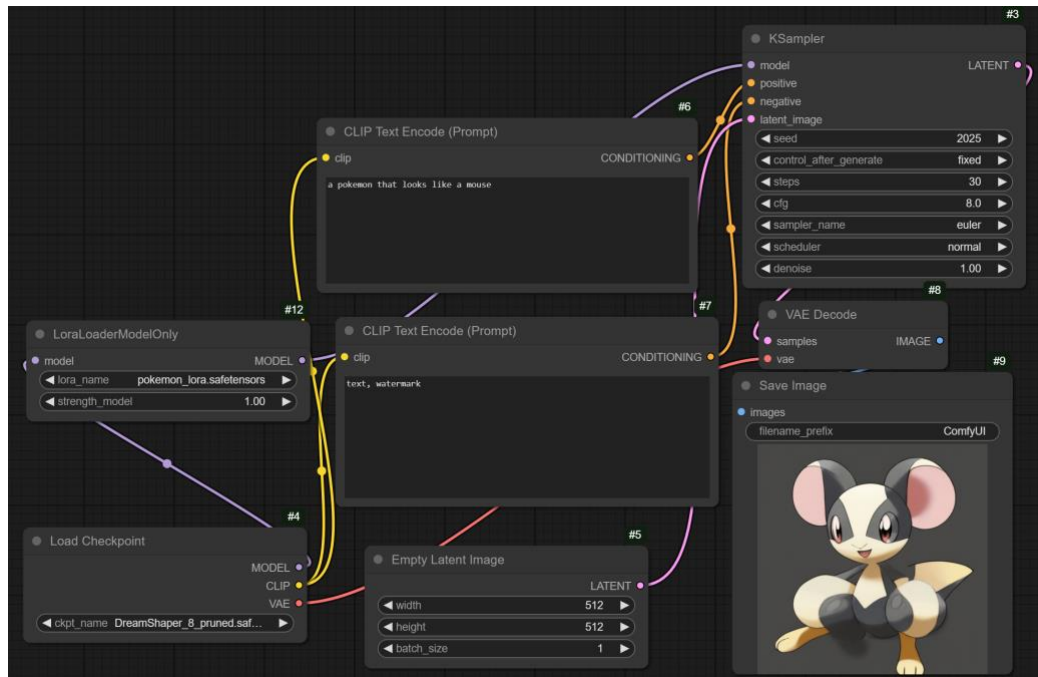
To make sure that the LoRA weights indeed have been successfully trained, you can check the folder **'ComfyUI/models/lora_logs/lora_showcase'**. As shown below, this folder indeed exists. From the generated files we can infer that LoRA weights are saved every 200 steps and are trained 1000 steps in total, consistent with our settings.



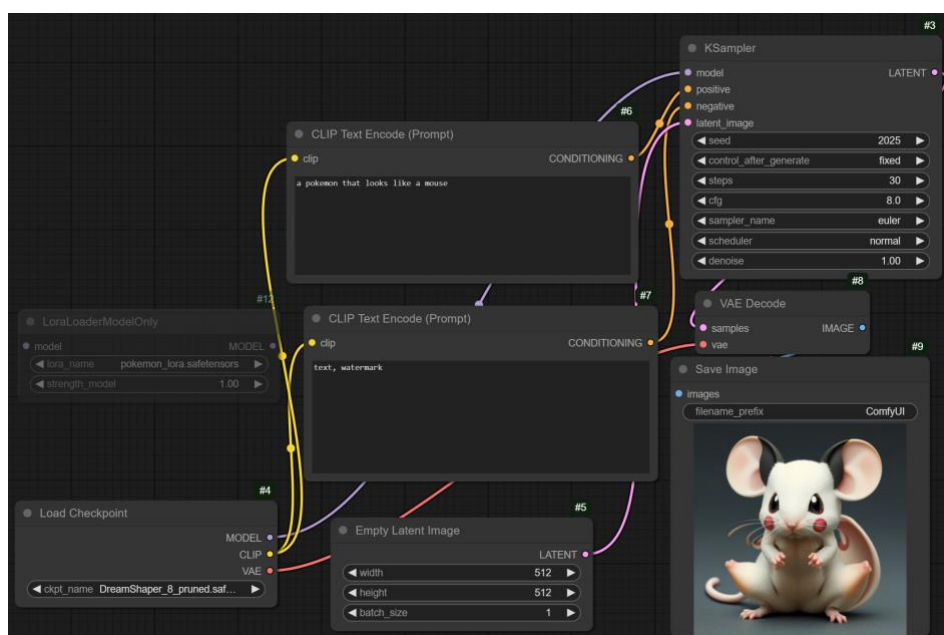
Then we may also investigate the folder '**ComfyUI/models/loras**', in which we see the trained weights '**pokemon_lora.safetensors**'. This further demonstrated that the LoRA training was successful.



To directly show the effect of trained LoRA, we may build a workflow and apply the trained LoRA weights to the DreamShaper 8.0 model. The workflow is very similar to the showcase in Part 2. However, **a major difference is that this time we should use the node 'LoraLoaderModelOnly' to replace 'Load LoRA' node, because in SimpleLoRA, the LoRA adapters are applied to transformer layers of UNet model only.** You may refer to the illustration below for a reference. When putting the positive textual prompts as 'a pokemon that looks like a mouse', the model with the LoRA outputs an image that really looks like a Pokemon in the shape of a mouse.



When we remove the node 'LoraLoaderModelOnly', the output image looks like a real mouse, instead of a Pokemon. **This demonstrates that the LoRA effectively shifts the style of output image from realistic to Pokemon-like.**



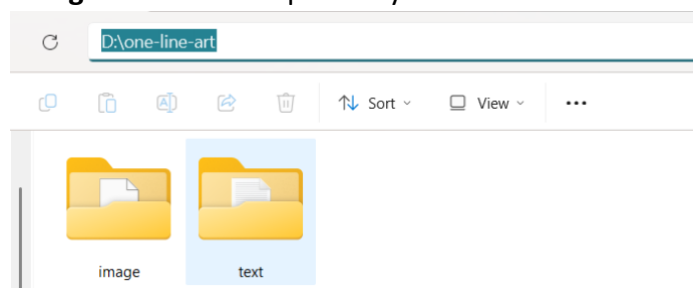
3. Exercise

We've learnt in Part 2 how to LoRA fine-tune a stable diffusion model with a text-to-image dataset downloaded from HuggingFace. In this exercise, let's try something more advanced: LoRA fine-tune a stable diffusion model **with a custom dataset**.

This is similar to what you've done in Part 3, but this time **you have to construct the dataset by yourself**. There are two major steps to construct a custom dataset. First of all, you need to collect some images that you want, and then you need to annotate each of them with a caption, either manually or by an image caption model. In this exercise we show you construct a small dataset of one-line-art images and annotate them manually.

Step 1: Prepare a root folder for your custom dataset and create 'image' and 'text' folders

Let's start by create a root folder of the dataset. The root folder can be everywhere inside your lab machine. In this exercise, we create the root folder at : 'D:\one-line-art'. Then we create another two new folders in the dataset root folder, and name them as 'image' and 'text' respectively.



Step 2: Collect images you like into a the 'image' folder

Download the images inside the 'image' folder you've created in step 1. Note that **it is very critical that you rename the images as a number(index) starting from 0, for example, '0.png', '1.png', so on and so forth**.

In this exercise, we can download the images from '[sd-concepts-library/one-line-drawing](#)' on HuggingFace as examples, rename them and place them inside the 'image' folder. And finally the image folder should look like this:

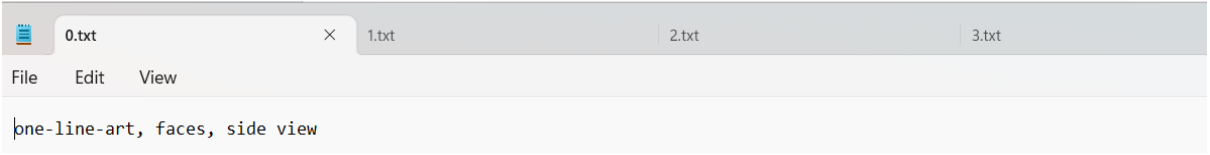


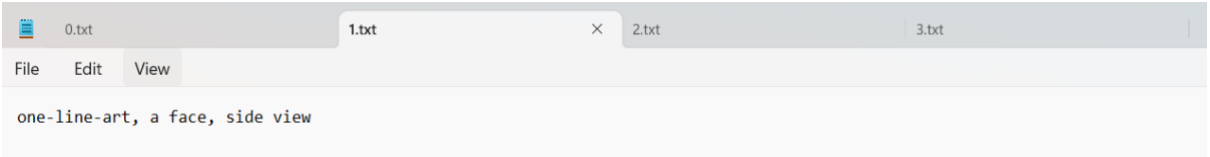
Step 3: Annotate the images and save the text files in 'text' folder

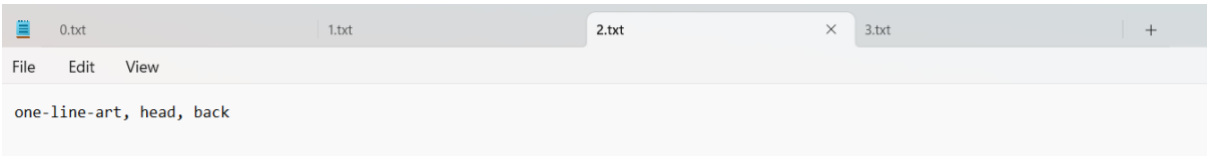
Open the 'text' folder you've created in step 1. Create empty text files and **also rename them as a number starting from 0, such as '0.txt', '1.txt', so on and so forth**. The texts inside '[x].txt' is exactly the annotation (textual prompts) corresponding with the image '[x].image', where [x] is the number/index you renamed. Then you can annotate the images you've downloaded in Step 2 within the text files.

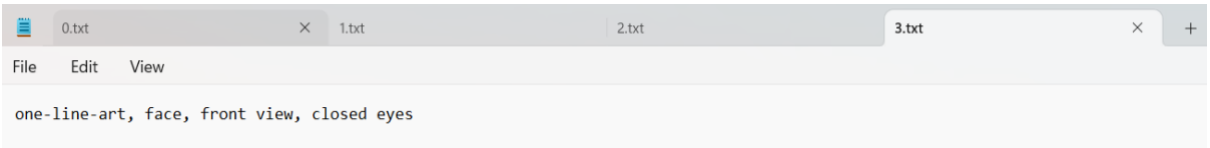
In our examples, since the training images are all ‘one-line-art’ drawings, so we can put the word ‘one-line-art’ in all text files. And we also add some descriptions of the features in each image, such as ‘side view’ for ‘1.jpeg’ and ‘closed eyes’ for ‘3.jpeg’. The text files and their contents will look like this:

Name	Date modified	Type	Size
0.txt	16/2/2025 11:28 pm	Text Document	1 KB
1.txt	16/2/2025 11:29 pm	Text Document	1 KB
2.txt	16/2/2025 11:29 pm	Text Document	1 KB
3.txt	16/2/2025 11:29 pm	Text Document	1 KB





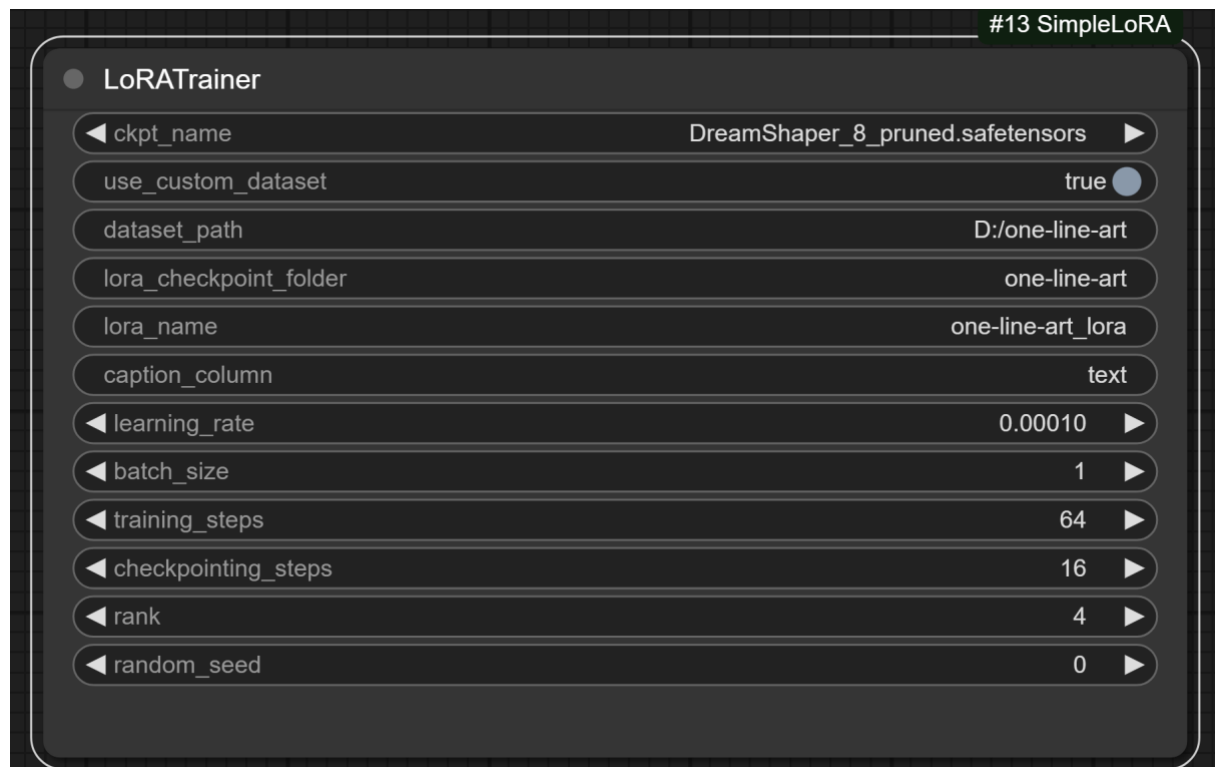




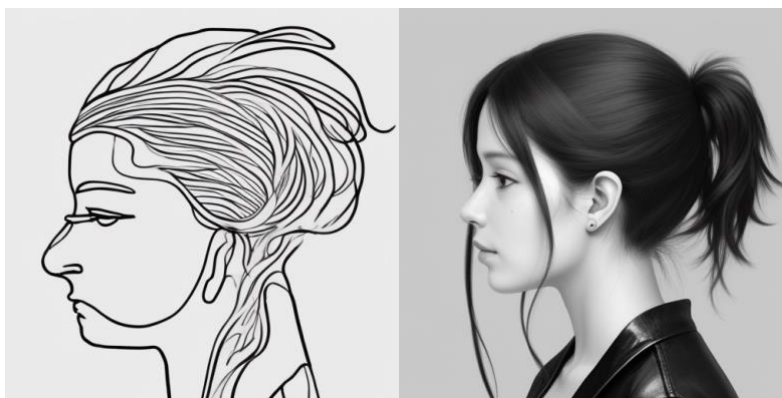
Sometimes when there are too many training images, you may also wish to annotate the images automatically by an image caption network. There are various ways to do so. For example, you can either use the custom node [‘Image Captioning in ComfyUI’](#) to utilize the WD Tagger network, or try to run the [ComfyUI MoonDream Caption](#) workflow and use the visual language model MoonDream for captioning.

Step 4: Run the SimpleLoRA node and LoRA fine-tune the base model with custom dataset

Build the one-node-only workflow for LoRA training with the 'LoRATrainer' node. Set 'use_custom_dataset' to 'true', and put the dataset root folder as the 'dataset_path'. In our examples, the 'dataset_path' should be 'D:\one-line-art'. Since there are only 4 images in a dataset, to avoid overfitting we don't need to train the LoRA so many steps. For example, we can set the 'training_steps' and 'checkpointing_steps' to 64 and 16 respectively. The LoRATrainer with proper settings for our example can look like this:



After the training, we can apply the LoRA to the base network, and put the prompts 'one-line-art, face, side view' into the positive textual prompt to see the result. The image on the left is generated with LoRA, while the one on the right not. By comparison, we can conclude the trained LoRA indeed works.



Final Submission: Please submit the ComfyUI workflow, fine-tuned LoRA weights, and 10 output images generated by the fine-tuned model, as a single zip file. Please do

Credit to SHANG Sifeng

remember to include a prompt text file that contains the 10 prompts used to generate the 10 submitted images.