LoRA Style Fine-Tuning with Stable Diffusion 1.5: "Ghibli Market"

Noas Shaalan Sayeeda Begam Goutham Muralikrishnan Muhid Abid Adam Lo
University of Technology Nuremberg

1. Introduction

We extended Stable Diffusion 1.5 with a new style token (<sks>) and fine-tuned it using LoRA so that the prompt *"a busy market, in <sks> style"* produces Ghibli-like images. This brief report focuses on process: what we tried, what failed (and why), what worked, and how we handled the main challenges. We organized the work into: (Adam) SD 1.5 pipeline, (Sayeeda) tokenizer/encoder edit for <sks>, (Noas) LoRA on UNet and CLIP text encoder, (Goutham) training and parameter tuning, and (Muhid) evaluation testing with various parameters to find best option.

2. Approach (What We Tried)

- **Pipeline:** Loaded SD 1.5 via `diffusers`, built a small dataloader for `512/`, and tried different schedulers.
- **Tokenization:** We introduced a new placeholder token <sks> into the tokenizer. This token was not part of the original vocabulary of the Stable Diffusion model, so we explicitly added it using add-token() function. After adding the token, we resized the text encoder's embedding matrix to accommodate the expanded vocabulary by using resize-token-embeddings(). This allowed the model to treat <sks> as a distinct concept during training.
- **LoRA:** Placed adapters on UNet attention (`to_q, to_k, to_v, to_out.0`) and CLIP text-encoder projections (`q_proj, k_proj, v_proj, out_proj`); base weights frozen; only adapters trainable. The embedding for <sks> was trained alongside the LoRA adapter weights and saved for reliable use during inference.
- **Training:** with rank 8 and trained for 844 steps using a batch size of 4. The training loop involved encoding images into latents, used a DDPMScheduler for noise scheduling and saved both the LoRA weights and the learned token embedding for <sks> in one file.
- **Evaluation:** Reloaded local adapters for inference, rendered multiple seeds of the required prompt, tried with difference negative prompts, and saved at least three images based on most effective result.

3. What Didn't Work (Why)

- **1: Randomized Seeding** Even after training, there were certain seed values that did not give desired Ghibli-like images. This was caused due to the fact that for certain seeds the model was not able to convert the pre-trained output into the fine-tuned LoRa result expected with the <sks> style"
- **2: Training w/ Fixed Prompt** Initially the "busy market" prompt was attached to each image during training, which caused the model to output Ghibli style, but not an actual market scene. After removing prompts during training, the output became much closer to expected.
- **3: Embedded tokens during training** The model learns to predict the original noise from the noisy latents, conditioned on a prompt that includes custom style token <sks>. We changed our training script to also retrieves the learned embedding vector for <sks> from the text encoder's embedding matrix. These components are merged into a single dictionary and saved as a .safetensors file, which can later be loaded for inference to reproduce the learned style.

4. What Worked

- **<sks> token embedding** worked sucessfully, and the model was able to recognize it. The only issue came when saving the weight for inference, which is discussed in next section
- **Negative Prompts** We added negative prompts during inference to avoid unclear images, We added negative prompt like "low-contrast, artifacts, etc." which gave much better outputs.
- **Parameters during inference** default 50 steps with CFG=7.5 produced good detail at steps with moderate guidance. This gave us the output images with

crisp style and better mimic the cell shaded style of Ghibli.

- **LoRa on U-Net and Text Encoder:** Inference with the trained LoRA weights produced stylized outputs that reflected the training data.

5. Main Challenges & Fixes

- **Loading Weight Tensor for Evaluation** → discard using Load-Lora() functions for SD1-5, and output tensor in PEFT format, and load in using state-dict() function.
- **Inferring with Embedded Token:** During training, the token embedding was trained in the text-encoder but not propogated to inference → Saving the token embedding alongside LoRA weights enabled proper restoration during inference.
- **Style consistency:** Initial evaluation had inconsistent outputs with odd anatomy and ugly images → using negative prompts along with longer training made output style consistent.

6. Lessons & Conclusion

With a narrow target and limited data set, LoRA fine-tuning with a properly initialized token was enough to meet the goal. The exercise clarified how SD1-5 is separated into U-net and Text encoders. We learned that when tokenizers are added, the fine tuning must also learn those embeddings. It also became clear how the token updates flow into the encoder, how adapters are saved during training and loaded down stream, and how schedulers (both during inference and training) trade detail for speed. Due to limited size of dataset, we also saw the importance of hyperparameters, like LR, Lora Rank, Batch Size, and noise schedulers. Another interesting aspect was how adding negative tokens greatly improved the output images. The final model reliably generates *"a busy market, in <sks> style."*

7. Reproducibility & Deliverables

All python scripts will work out of the box. Simply run the train-lora.py script which will save the tensors. Existing weights can be loaded by running the eval-lora.py will automatically output 3 sample images.

We packaged:

- `lora_out/pytorch_lora_weights.safetensors`
- `code/train_lora.py`, `code/eval_lora.py` (clear CLI instructions)
- `samples/` (≥ 3 outputs)
- `requirements.txt` (or `environment.yml`)
- `report.pdf` (this document, ≤ 2 pages)

Resources: Diffusers | PEFT/LoRA | LoRA paper

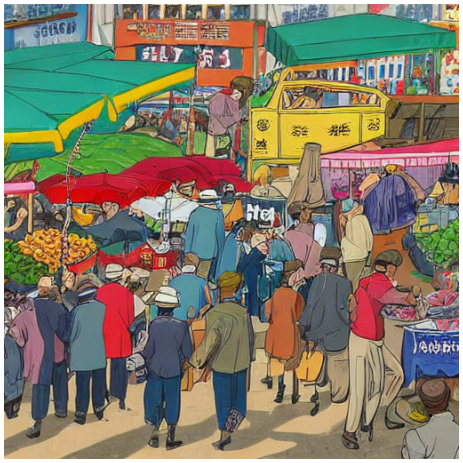

Figure 1: Sample Output 1

Figure 2: Sample Output 2

Figure 3: Sample Output