# 实验四：古诗词转水墨画（大模型）

学号：2351753　姓名：黄保翔　专业：计算机科学与技术　日期：6.10

## 一、问题概述

### 1.1 问题的直观描述

**目标**：利用 Stable Diffusion（深度学习模型），将古诗自动转换为水墨画

**数据集**：已经给出，本次实验选择 Paint4Poem-Web-famous-subset-20250402T083354Z-001.zip 文件，内有 csv 文件用于匹配文字描述和图像编码，还有压缩包文件用于存储对应编码图片。

**任务要求**：首先基于提供的 Stable Diffusion 模型代码和预训练参数搭建实验环境，然后利用给定的训练和测试数据集进行模型训练和参数调整，最终实现古诗到水墨画的自动转换功能。

### 1.2 解决问题的思路和想法

#### 1.2.1 环境配置

##### 1.2.1.1 cuda 环境配置

输入指令：nvidia-smi 查找对应的 cuda 版本

```
C:\Users\DELL>nvidia-smi
Sun Jun  8 14:49:43 2025
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 566.35                 Driver Version: 566.35         CUDA Version: 12.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                     Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                                                         MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce RTX 3060 ...   WDDM  | 00000000:01:00.0 Off |                  N/A |
| N/A   56C    P8             12W /  128W |     22MiB /   6144MiB |      0%      Default |
|                                                                         N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      5864    C+G   ...inaries\Win64\EpicGamesLauncher.exe      N/A |
|    0   N/A  N/A     32812    C+G   ...ekyb3d8bbwe\WsaClient\WsaClient.exe      N/A |
|    0   N/A  N/A     37788    C+G   D:\Program Files\QQ.exe                     N/A |
+-----------------------------------------------------------------------------+
```

进入官网选择合适的版本进行安装：CUDA Toolkit Archive | NVIDIA Developer

完成后查看 nvcc -version

```
C:\Users\DELL>nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2024 NVIDIA Corporation
Built on Fri_Jun_14_16:44:19_Pacific_Daylight_Time_2024
Cuda compilation tools, release 12.6, V12.6.20
Build cuda_12.6.r12.6/compiler.34431801_0
```

### 1.2.1.2 pytorch 配置

进入 https://pytorch.org/ 选择对应的版本指令进行安装，完成后检查：

python -c "import torch; print('PyTorch 版本:', torch.__version__); print('CUDA 是否可用:', torch.cuda.is_available())"

```
PyTorch 版本: 2.7.1+cu118
CUDA 是否可用: True
```

至此主要的环境已经配置完成，由于本地显卡显存不足，因此将上述步骤同步实现远程服务器中。

### 1.2.1.3 swanlab 可视化操作

登录 swanlab，使用 swanlab 库进行完成模型训练过程的可视化

```
NVIDIA GeForce RTX 3060 Laptop GPU
swanlab: swanlab version 0.6.1 is available!  Upgrade: `pip install -U swanlab`
swanlab: Tracking run with swanlab version 0.6.0
swanlab: Run data will be saved locally in E:\AI_learning\swanlog\run-20250608_163646-3ad2dfde
swanlab: 👋 Hi tosakikolumia, welcome to swanlab!
swanlab: Syncing run SD1-5_古诗图像生成 to the cloud
swanlab: 🏠 View project at https://swanlab.cn/@tosakikolumia/SD-Poem
swanlab: 🚀 View run at https://swanlab.cn/@tosakikolumia/SD-Poem/runs/qh7w491k7eiuv9n9sgqni
```

## 1.2.2 Stable Diffusion 模型微调

基于 PyTorch 和 Hugging Face 的 Diffusers 库，实现了 Stable Diffusion 模型的微调流程。解决问题的主要步骤思路如下。

1. 模型加载

加载预训练的 Stable Diffusion v1.5 模型，包括变分自编码器（VAE）、CLIP 文本编码器、UNet 模型和调度器（DDPMScheduler）。

2. 对数据进行预处理

包括加载数据集，进行图像预处理，和文本预处理。图像预处理包括：调整大小、裁剪、随机翻转、归一化。文本预处理包括使用 CLIPTokenizer 将古诗文本编码为固定长度的 token。

3. 进行训练，主要操作为

冻结 VAE 和文本编码器，仅训练 UNet 模型（可选使用 EMA 进行平滑）。使用 DDPM（去噪扩散概率模型）进行训练，添加噪声并预测目标。 支持 SNR 加权损失、梯度累积、混合精度训练等优化。

4.验证与保存

设定 validation_prompts 与 validation_epochs 每隔一段时间记录使用 validation_prompts 生成的图片，记录实验结果。保存训练检查点和最终模型。

下一章介绍每一步的具体操作

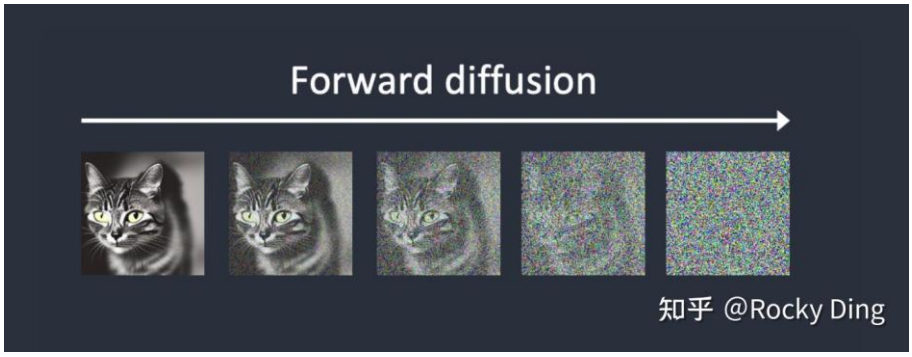
# 二、算法设计

## 2.1 Stable_Diffusion 原理

Stable Diffusion 是一种基于扩散模型（Diffusion Model）的生成式人工智能技术，广泛用于文本到图像的生成任务。其核心算法原理结合了去噪扩散概率模型（DDPM）、变分自编码器（VAE）和条件生成技术。

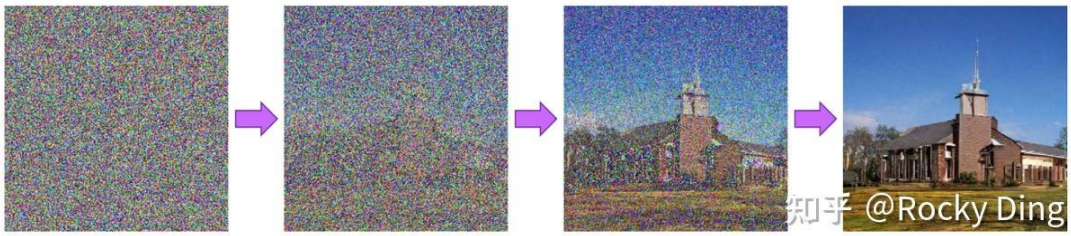
### 前向扩散过程

给定一张真实图像 x0 x_0 x0，通过一系列时间步 t=1,2,…,T t = 1, 2,…, T t=1,2,…,T，逐步向图像添加高斯噪声。 每一步的噪声添加遵循一个马尔可夫链，生成越来越模糊的图像最终接近纯噪声（各向同性高斯分布）。




### 反向扩散过程

从纯噪声开始，通过学习一个逆向概率分布，逐步去噪生成接近真实图像的样本。




### 核心组件与优化机制

为提高计算效率并支持条件生成，Stable Diffusion 采用以下核心组件：

1. **变分自编码器（VAE）**

   VAE 将高分辨率图像（如 512x512）编码为低维潜在表示（如 64x64），通过编码器 E(x) E(x) E(x) 压缩图像为潜在表示 z z z，并通过解码器 D(z) D(z) D(z)

重建图像。这种潜在空间操作显著降低显存需求，使模型可在消费级 GPU 上运行（如 RTX 3090/4090）。在训练脚本中，VAE 被冻结，仅用于图像编码和解码，专注于潜在空间的扩散过程。

2. **UNet 网络**

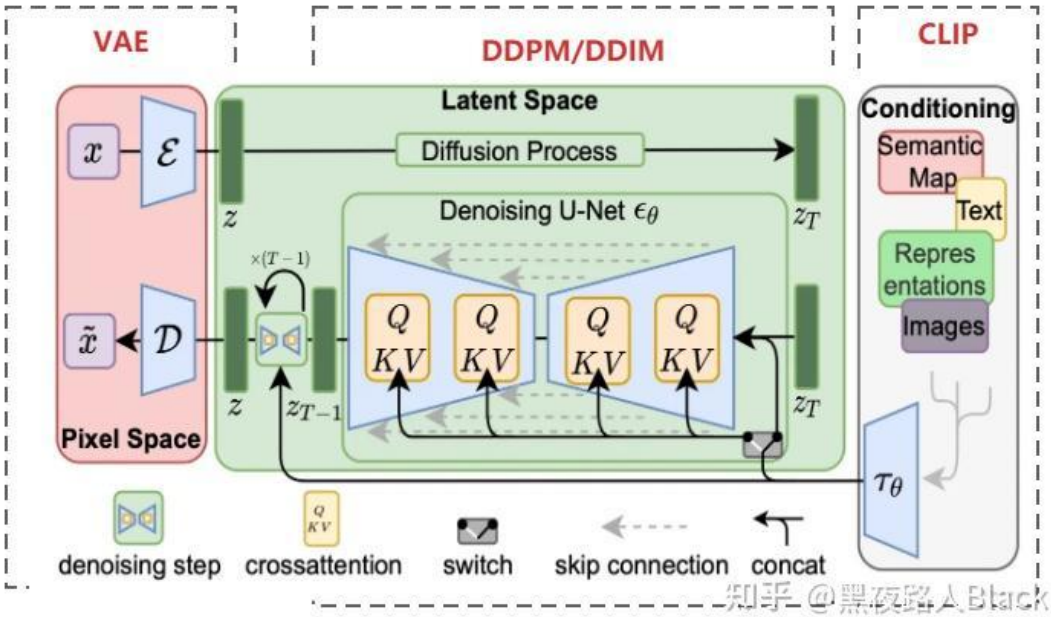
   UNet 是去噪过程的核心，采用编码器-解码器结构，结合跳跃连接保留多尺度特征。输入包括带噪潜在表示 xt x_t xt、时间步 t t t 和条件信息（如古诗的 CLIP 嵌入），通过交叉注意力机制融合文本信息，输出预测噪声 ϵθ(xt,t,c) \epsilon_\theta(x_t, t, c) ϵθ(xt,t,c)。在脚本中，仅微调 UNet 以适配古诗图像生成任务，保持 VAE 和 CLIP 编码器不变。

3. **CLIP 文本编码器**

   CLIP（对比语言-图像预训练）将文本提示（如古诗）编码为固定长度的嵌入向量，输入 UNet 的交叉注意力层，指导生成过程。CLIP 的强大语义理解能力确保生成图像与古诗的意境（如“山川寂静，月光如水”）一致。在训练脚本中，CLIP 编码器被冻结，仅提供条件嵌入。

4. **噪声调度器**

   噪声调度器（如 DDPMScheduler）控制前向和逆向过程中的噪声强度，支持 epsilon 或 v-prediction 预测模式。推理阶段可采用加速采样器（如 DDIM），减少步数（如脚本中的 20 步）以提高效率。噪声调度参数 βt \beta_t βt 通常采用线性或余弦调度，确保去噪过程的稳定性。




## 2.2 微调模型设计思路

在有数据集和预训练模型的基础上，进行微调设置

### 2.2.1 数据处理与预处理

第一步是适配“Paint4Poem-Web-famous-subset”数据集。通过 load_dataset 加载数据，CSV 格式包含图像文件名（image_id）和古诗文本（poem），再通过图像文件名找到文件夹中的图像。

图像预处理使用 torchvision.transforms，包括以下几个步骤:

- 调整分辨率（如 512x512，args.resolution），使用双线性插值；
- 中心裁剪（args.center_crop）或随机裁剪，增强数据多样性；
- 随机水平翻转（args.random_flip），提高模型鲁棒性；
- 归一化为[-1, 1]，适配 VAE 输入。

文本预处理采用上文提到的 CLIPTokenizer，将古诗编码为固定长度 toke（max_length），超出部分截断，生成 input_ids。

### 2.2.2 模型训练设计

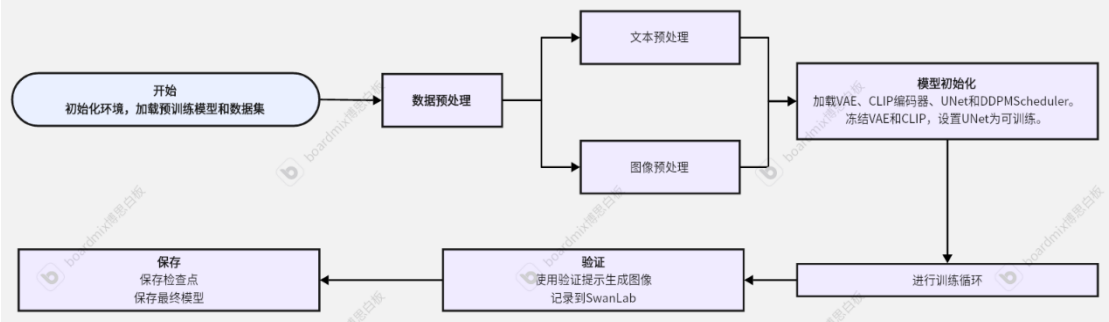
微调的核心是优化 UNet 模型，而冻结 VAE 和 CLIP 文本编码器，以保留预训练模型的通用特征提取能力。

### 2.2.3 验证与模型保存

每隔 args.validation_epochs 个 epoch，使用 args.validation_prompts（预定义古诗提示）通过 Stable Diffusion Pipeline 生成图像（num_inference_steps=20）。生成图像记录到 SwanLab（swanlab.log），便于可视化评估意境一致性。

每 args.checkpointing_steps 步保存模型状态至 output_dir/checkpoint-{global_step}，并通过 args.checkpoints_total_limit 限制检查点数量，避免存储空间溢出。

训练完成后，保存完整的 Stable Diffusion Pipeline（包括 UNet、VAE、CLIP 编码器），支持后续推理。

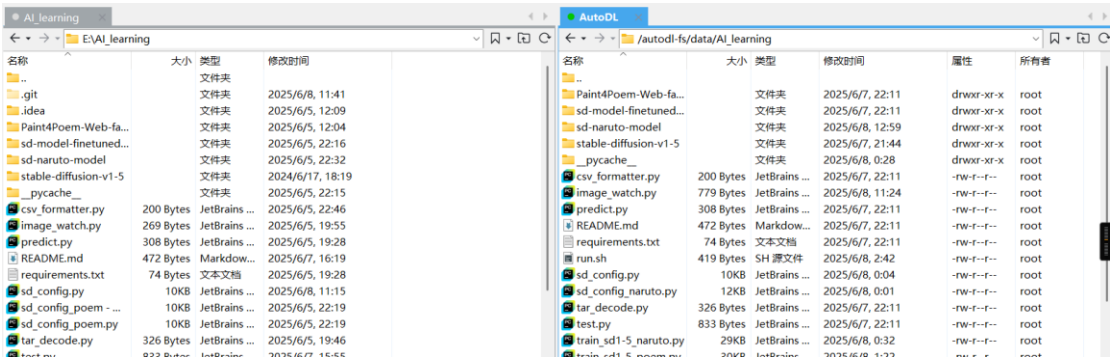



# 三、微调具体实现

## 3.1 初始化环境

由于本地配置不足，本次实验在服务器上进行

将所需文件上传至服务器端，加载预训练模型和数据集




1. 导入实验需要的库

```
import …
```

2. 配置 SwanLab 跟踪器，记录训练损失、学习率和生成图像。

```
    swanlab.init(project="SD-Poem", experiment_name="SD1-5_古诗图像生成
", ...)
    swanlab_tracker = SwanLabTracker(project="SD-
Poem", experiment_name="SD1-5_古诗图像生成", ...)
```

3. 初始化 Accelerator（accelerate 库），支持分布式训练、混合精度（FP16/BF16）和梯度累积。

```
    accelerator = Accelerator(
        gradient_accumulation_steps=args.gradient_accumulation_steps,
        mixed_precision=args.mixed_precision,
        log_with=swanlab_tracker,
        project_config=accelerator_project_config,
    )
```

## 3.2 数据加载与预处理

加载“Paint4Poem-Web-famous-subset”数据集（CSV 或 Parquet 格式），对图像和文本进行预处理，生成适合模型输入的格式。

1. 映射图像和文本列（image_id 和 poem）

```
DATASET_NAME_MAPPING = {
    "Paint4Poem-Web-famous-subset": ("image_id", "poem"),
}
```

2. 使用 CLIPTokenizer 将古诗编码为 input_ids，支持截断和填充。

```
def tokenize_captions(examples, is_train=True):
    captions = []
```

```
    for caption in examples[caption_column]:
        if isinstance(caption, str):
            captions.append(caption)
        elif isinstance(caption, (list, np.ndarray)):
            captions.append(random.choice(caption) if is_train else caption[0])
        inputs = tokenizer(captions, max_length=tokenizer.model_max_length, padding="max_length", ..
.)
      return inputs.input_ids
```

3. 调整分辨率、裁剪、随机翻转、归一化，生成 pixel_values（张量格式）。

```
train_transforms = transforms.Compose([
    transforms.Resize(args.resolution, interpolation=transforms.InterpolationMode.BILINEAR),
    transforms.CenterCrop(args.resolution) if args.center_crop else transforms.RandomCrop(args.resolution),
    transforms.RandomHorizontalFlip() if args.random_flip else transforms.Lambda(lambda x: x),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5]),
])
```

4. 对图像进行预处理，应用上文提到的变换，返回预处理后的数据包含了经过变换的图像张量（"pixel_values"）和标记词化的描述性文字（"input_ids"）。若图像加载失败，生成黑色图像占位。

```
def preprocess_train(examples):
    images = []
    for image_id in examples[image_column]:
        image_path = os.path.join(args.train_data_dir, "images", f"{image_id}.jpeg")
        try:
            image = Image.open(image_path).convert("RGB")
        except:
            black_image = Image.new("RGB", (args.resolution, args.resolution), (0, 0, 0))
            images.append(black_image)
        images.append(image)
    examples["pixel_values"] = [train_transforms(image) for image in images]
    examples["input_ids"] = tokenize_captions(examples)
      return examples
```

5. 通过 DataLoader 批量加载数据

```
if args.dataset_name is not None:
    dataset = load_dataset(args.dataset_name, args.dataset_config_name, ...)
else:
    dataset = load_dataset("csv", data_files={"train": os.path.join(args.train_data_dir, "POEM_IMAGE_clean.csv")}, ...)
train_dataset = dataset["train"].with_transform(preprocess_train)
```

```
train_dataloader = torch.utils.data.DataLoader(train_dataset, shuffle=True, collate_fn=collate_fn, ...)
```

## 3.3 模型加载与配置

加载预训练 Stable Diffusion v1.5 模型（VAE、CLIP 编码器、UNet、调度器），配置训练参数，冻结非训练部分。

1. 从预训练模型中加载稳定扩散模型所需的各个组件，包括 VAE、CLIP 编码器、UNet 和 DDPMScheduler

```
noise_scheduler = DDPMScheduler.from_pretrained(args.pretrained_model_name_or_path, subfolder="scheduler")
tokenizer = CLIPTokenizer.from_pretrained(args.pretrained_model_name_or_path, subfolder="tokenizer", ...)
text_encoder = CLIPTextModel.from_pretrained(args.pretrained_model_name_or_path, subfolder="text_encoder", ...)
vae = AutoencoderKL.from_pretrained(args.pretrained_model_name_or_path, subfolder="vae", ...)
unet = UNet2DConditionModel.from_pretrained(args.pretrained_model_name_or_path, subfolder="unet", ...)
```

2. 冻结 VAE 和 CLIP 编码器，仅训练 UNet，减少参数量

```
        vae.requires_grad_(False)  # 冻结 VAE
        text_encoder.requires_grad_(False)  # 冻结 CLIP 编码器
        unet.train()  # 设置 UNet 为训练模式
```

3. 为条件 U-Net（UNet2DConditionModel）创建指数移动平均（EMA）模型

```
    if args.use_ema:
      ema_unet = UNet2DConditionModel.from_pretrained(
          args.pretrained_model_name_or_path, subfolder="unet", revision=args.revision, variant=args.variant
      )
      ema_unet = EMAModel(ema_unet.parameters(), model_cls=UNet2DConditionModel, model_config=ema_unet.config)
```

4. 设置优化器与学习率调度器

```
optimizer = torch.optim.AdamW(unet.parameters(), lr=args.learning_rate, betas=(args.adam_beta1, args.adam_beta2), .)
lr_scheduler = get_scheduler(args.lr_scheduler, optimizer=optimizer, num_warmup_steps=args.lr_warmup_steps, ...)
```

## 3.4 模型训练流程

1. 设置训练轮数：

```
for epoch in range(first_epoch, args.num_train_epochs): …
```

对于每一轮训练：

2. 加载数据

for step, batch in enumerate(train_dataloader):…

2. 梯度累计上下文

```
with accelerator.accumulate(unet):
```

3. 将输出图像编码到潜在空间

```
latents = vae.encode(batch["pixel_values"].to(weight_dtype)).latent_dist.sample()
latents = latents * vae.config.scaling_factor
```

4. 添加噪声

```
noise = torch.randn_like(latents)
if args.noise_offset:
    noise += args.noise_offset * torch.randn((latents.shape[0], latents.shape[1], 1, 1), ...)
```

```
timesteps = torch.randint(0, noise_scheduler.config.num_train_timesteps, (bsz,), ...)
noisy_latents = noise_scheduler.add_noise(latents, noise, timesteps)
```

5. 将文本编码为嵌入向量

```
encoder_hidden_states = text_encoder(batch["input_ids"], return_dict=False)[0]
```

6. 前向传播

```
model_pred = unet(noisy_latents, timesteps, encoder_hidden_states, return_dict=False)[0]
```

7. 损失计算

```
if args.snr_gamma is None:
    loss = F.mse_loss(model_pred.float(), target.float(), reduction="mean")
else:
    snr = compute_snr(noise_scheduler, timesteps)
    mse_loss_weights = torch.stack([snr, args.snr_gamma * torch.ones_like(timesteps)], dim=1).min(dim=1)[0]
    loss = F.mse_loss(model_pred.float(), target.float(), reduction="none")
    loss = loss.mean(dim=list(range(1, len(loss.shape)))) * mse_loss_weights
    loss = loss.mean()
```

8. 反向传播

```
accelerator.backward(loss)
```

9. 更新模型参数并调整学习率

```
if accelerator.sync_gradients:
    accelerator.clip_grad_norm_(unet.parameters(), args.max_grad_norm)
optimizer.step()
lr_scheduler.step()
optimizer.zero_grad()
```

10. 更新 EMA 模型的参数

```
if accelerator.sync_gradients:
    if args.use_ema:
        ema_unet.step(unet.parameters())
```

## 3.5 验证模型质量

1. 构建推理管道（StableDiffusionPipeline）

```
pipeline = StableDiffusionPipeline.from_pretrained(
    args.pretrained_model_name_or_path, vae=accelerator.unwrap_model(vae), ...)
pipeline = pipeline.to(accelerator.device)
```

2. 遍历预先设置好的提示词，用 StableDiffusionPipeline 生成图片

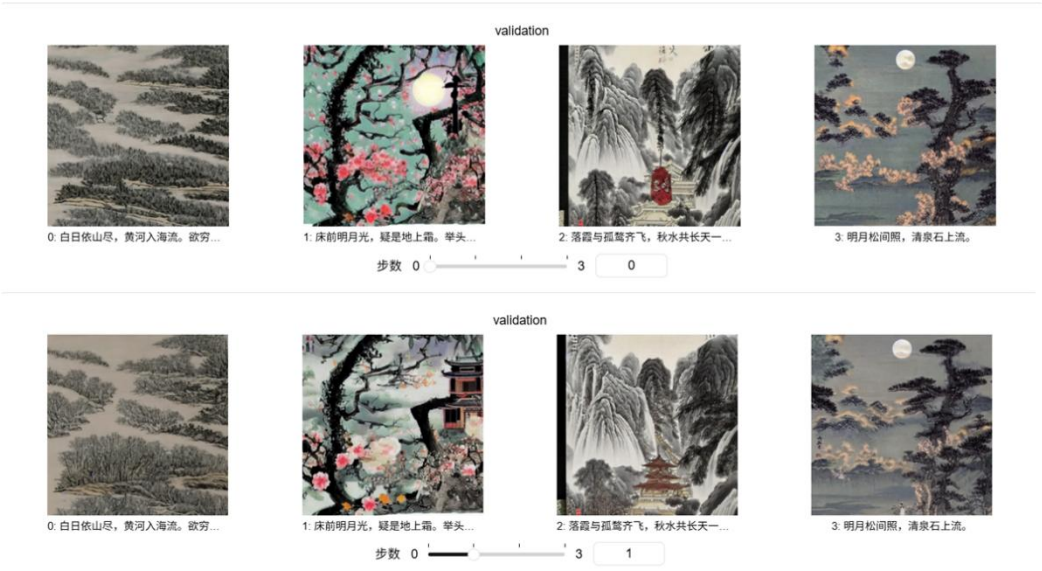
```
for i in range(len(args.validation_prompts)):
    with torch.autocast(accelerator.device.type):
        image = pipeline(args.validation_prompts[i], num_inference_steps=20, generator=generator).images[0]
    images.append(swanlab.Image(image, caption=f"{i}: {args.validation_prompts[i]}"))
```

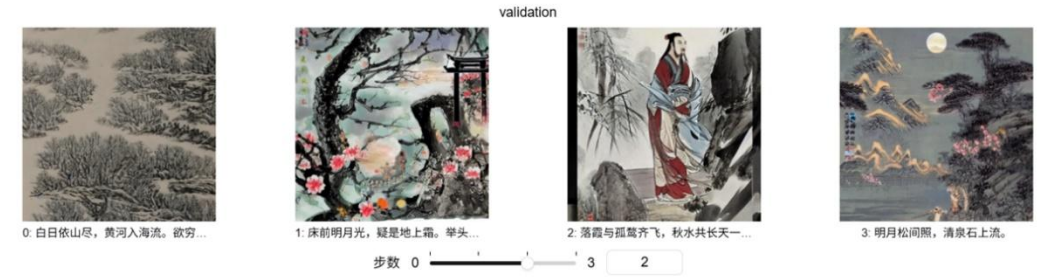
## 3.6 保存训练检查点和最终模型

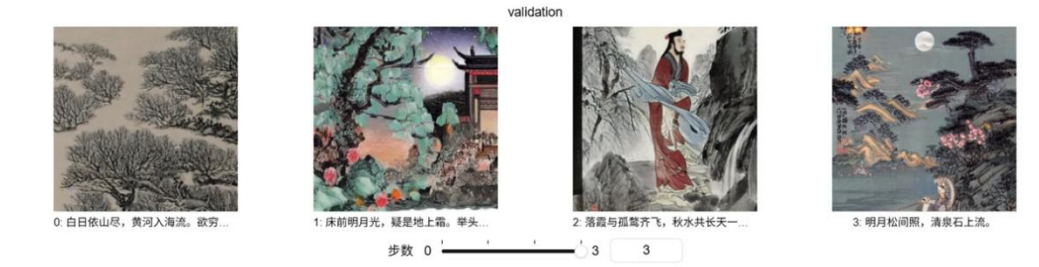
```
if global_step % args.checkpointing_steps == 0:
    if accelerator.is_main_process:
        save_path = os.path.join(args.output_dir, f"checkpoint-{global_step}")
        accelerator.save_state(save_path)
        logger.info(f"Saved state to {save_path}")
if accelerator.is_main_process:
    unet = unwrap_model(unet)
    if args.use_ema:
        ema_unet.copy_to(unet.parameters())
    pipeline = StableDiffusionPipeline.from_pretrained(
        args.pretrained_model_name_or_path, text_encoder=text_encoder, vae=vae, unet=unet, ...)
    pipeline.save_pretrained(args.output_dir)
```

# 四、实验结果

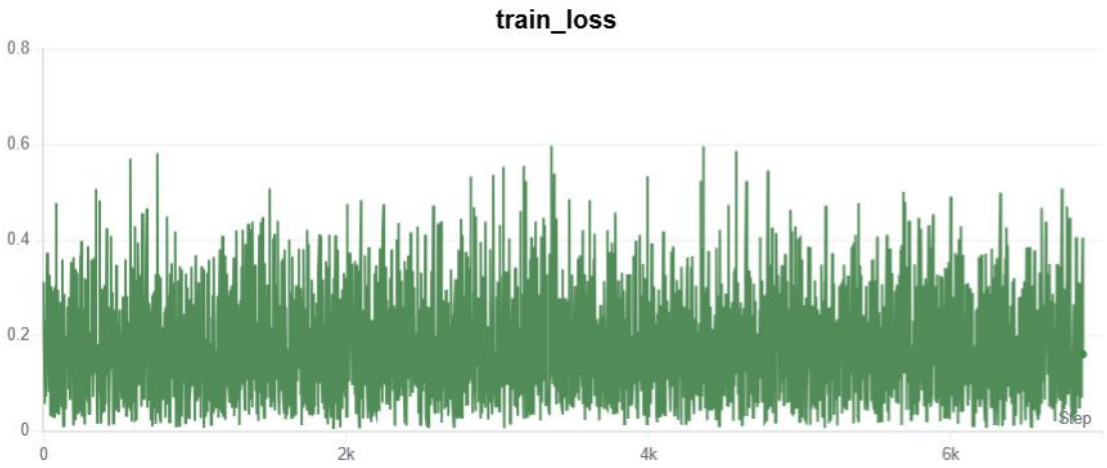
在进行训练时，每一个 epoch 进行模型的验证，使用特定的 prompt 进行图片生成，存储在 swanlab 中进行查看。






训练过程中的 loss 损失图像




使用最终保存的模型进行验证

使用提示词： 接天莲叶无穷碧，映日荷花别样红

提示词：明月松间照，清泉石上流

使用原模型进行比对：

**左图为原模型不经过微调**得到的图像，**右图为使用水墨画数据集进行微调**后得到的图像，可见我们使用数据集进行微调后能有效的控制模型根据古诗输出水墨画风格的图像。

**选做部分：**

## 更改数据集

我们还可以通过更改数据集的方式控制模型生成写实的图像

我们选取 Huggingface 上的 laion/laion400m-preview 数据集进行测试

引入

```
from datasets import load_dataset
# 加载 laion400m 的一个小样本（建议用于预览）
dataset = load_dataset("laion/laion400m-preview")
```

由于代码较为相同，在此不多叙述，结果如下：

提示词：A lonely smoke rises straight in the vast desert, The long river reflects the setting sun as a perfect circle.

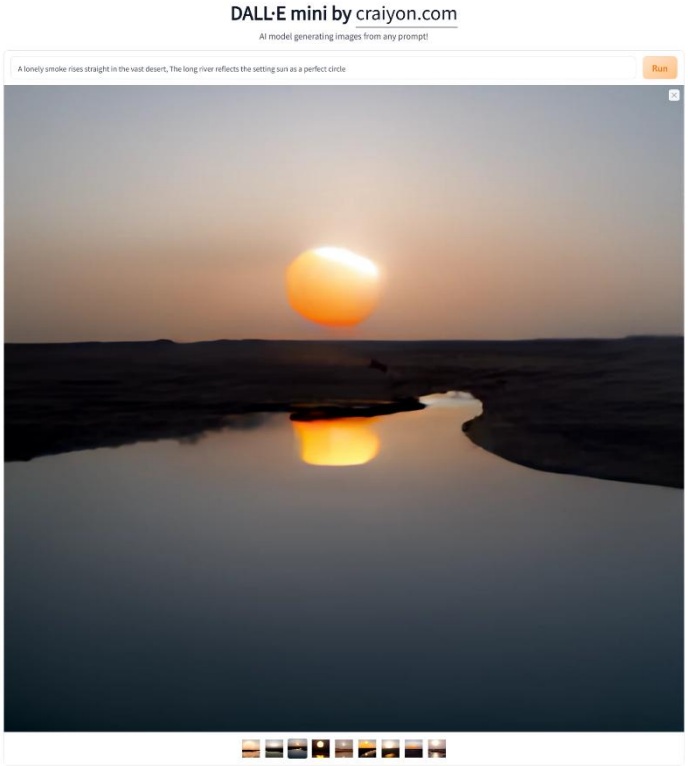
可见在写实风格数据集下，图像生成更有真实感。

## 不同模型生成图像的尝试

使用可视化训练平台 Hugging Face Spaces（Zero-Code 试验平台）尝试不同的模型效果

挑选轻量化模型 DALL·E Mini，进行尝试

提示词：A lonely smoke rises straight in the vast desert, The long river reflects the setting sun as a perfect circle（大漠孤烟直，长河落日圆）




由于 DALL·E Mini 是轻量化的文生图模型，因此图片质感较差，但是对于 prompt 的理解能力较好。

## 对模型提出改进

该部分将在下一章说明模型的改进方法

# 五、总结与分析

## 5.1 实验过程中遇到的问题及解决方案

### 5.1.1 数据集解压错误

数据集使用 tar.gz 格式压缩，windows 无法直接解压

处理方式：

1. 将压缩包上传至服务器进行解压

2. 使用 wsl 在本地进行解压再上传至服务器

```
tosakikolumia@DESKTOP-ARG52QI:~$ cd /mnt/e
tosakikolumia@DESKTOP-ARG52QI:/mnt/e$ cd AI_learning
tosakikolumia@DESKTOP-ARG52QI:/mnt/e/AI_learning$ cd Paint4Poem-Web-famous-subset
tosakikolumia@DESKTOP-ARG52QI:/mnt/e/AI_learning/Paint4Poem-Web-famous-subset$ tar -xzvf images.tar.gz
```

### 5.1.2 图片找不到/图片损坏

数据集中大部分为 jpeg 格式图片，少部分为 png 格式。为了方便使用，将所有的 png 格式文件转为 jpeg 格式文件

```python
import os
from PIL import Image
input_dir = './Paint4Poem-Web-famous-subset/images'       # png 图片所在文件夹
output_dir = './Paint4Poem-Web-famous-subset/images'     # jpeg 图片输出文件夹
os.makedirs(output_dir, exist_ok=True)
for filename in os.listdir(input_dir):
    if filename.lower().endswith('.png'):
        png_path = os.path.join(input_dir, filename)
        jpeg_name = os.path.splitext(filename)[0] + '.jpeg'
        jpeg_path = os.path.join(output_dir, jpeg_name)
        try:
            with Image.open(png_path) as img:
                rgb_img = img.convert('RGB')  # PNG 可能有透明通道，JPEG 不支持
                rgb_img.save(jpeg_path, 'JPEG')
                print(f'已转换: {filename} -> {jpeg_name}')
        except Exception as e:
            print(f'转换失败: {filename}，原因: {e}')
```

除此之外，给定的图片中还有一些编码错误，对于编码存在问题的图片使用黑色的图片进行替代

```python
        try:
            image = Image.open(image_path)
            images.append(image.convert("RGB"))
        except Exception as e:
            logger.error(f"Failed to load image {image_path}: {str(e)}")
            # Create a black image (512x512 by default, matching args.resolution)
            black_image = Image.new("RGB", (args.resolution, args.resolution), (0, 0, 0))
```

```
                images.append(black_image)
```

### 5.1.3 CSV 格式错误

CSV 文件中有格式错误，缺少逗号，为此需要进行格式修正

运行

python csv_formatter.py 可得到正确格式的 CSV 用于输入标签和对应的图像编号。

## 5.2 模型优化方式

在进行模型训练时，直接训练时间较长，通过使用各种算法优化技术进行缩短时间增加准确度。

### 5.2.1 xFormers 高效注意力机制

xFormers 内存高效注意力机制通过多种优化技术，如分块注意力、低秩近似和稀疏注意力，显著减少了 Transformer 模型中注意力机制的内存使用和计算复杂度。

代码中使用：

```
if args.enable_xformers_memory_efficient_attention:
    pipeline.enable_xformers_memory_efficient_attention()
```

### 5.2.2 混合精度训练

通过使用半精度（float16）和单精度（float32）浮点格式来加速训练过程并减少内存使用。混合精度训练可以在不影响模型精度的前提下，显著提高训练速度和效率。

```
accelerator = Accelerator(
    gradient_accumulation_steps=args.gradient_accumulation_steps,
    mixed_precision=args.mixed_precision,
    log_with=swanlab_tracker,
    project_config=accelerator_project_config,
)
# 训练和推理时
with torch.autocast(accelerator.device.type):
    image = pipeline(args.validation_prompts[i], num_inference_steps=20, generator=generator).images[0]
```

### 5.2.3 指数滑动平均

EMA（指数滑动平均，Exponential Moving Average）是一种在训练过程中维护模型参数平均值的技术。通过使用 EMA，可以在训练过程中获得更平滑和稳定的模型参数，从而提高生成图像的质量和模型的泛化能力。

```
if args.use_ema:
    ema_unet = UNet2DConditionModel.from_pretrained(
        args.pretrained_model_name_or_path, subfolder="unet", revision=args.revision, variant=args.variant
```

```
    )
    ema_unet = EMAModel(ema_unet.parameters(), model_cls=UNet2DConditionModel, model_config=ema_unet.config)
# 训练过程中
if args.use_ema:
    ema_unet.step(unet.parameters())
# 验证前
if args.use_ema:
    ema_unet.store(unet.parameters())
    ema_unet.copy_to(unet.parameters())
# 验证后
if args.use_ema:
    ema_unet.restore(unet.parameters())
```

### 5.2.4 TF32 支持

TF32（TensorFloat-32）是 NVIDIA 在 Volta 架构及更高架构的 GPU 中引入的一种混合精度计算模式。TF32 利用了 32 位浮点数的优势，同时结合了更低精度的计算来加速矩阵乘法操作，从而提高训练和推理的性能。

```
if args.allow_tf32:
    torch.backends.cuda.matmul.allow_tf32 = True
```

### 5.2.5 Dream Training

Dream Training 是一种在生成模型训练过程中引入额外噪声或扰动的技术，旨在提高模型的鲁棒性和生成图像的多样性。通过在训练过程中引入噪声，模型可以更好地学习到更丰富的特征，从而生成更加高质量和多样的图像。

```
if args.dream_training:
    noisy_latents, target = compute_dream_and_update_latents(
                        unet, noise_scheduler, timesteps, noise, ...)
```