

第 6 次实验报告

2022111663 刘博飞

2024 年 12 月 14 日

1 实验环境介绍

1. 操作系统

版本 Windows 11 专业版

版本号 23H2

安装日期 2024/10/10

操作系统版本 22631.4317

体验 Windows Feature Experience Pack 1000.22700.1041.0

2. GPU 型号

NVIDIA-SMI 560.94			Driver Version: 560.94	CUDA Version: 12.6		
GPU	Name	Driver-Model	Bus-Id	Disp.A	Volatile	Uncorr. ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.
0	NVIDIA GeForce RTX 3090	WDDM	00000000:01:00.0	Off	N/A	
32%	28C	P8	12W / 350W	0MiB / 24576MiB	0%	Default
						N/A
Processes:						
GPU	GI	CI	PID	Type	Process name	GPU Memory Usage
ID			ID			
No running processes found						

图 1: GPU 型号

3. torch 版本

torch 2.4.1+cu121 pypi_0 pypi

torchaudio 2.4.1+cu121 pypi_0 pypi

torchvision 0.19.1 pypi_0 pypi

4. resize 尺寸

256 × 256

2 LoRA 与 Stable Diffusion 简介

2.1 模型微调 (finetune)

Fine-tuning (微调)，是指在新数据集上调整预训练模型的权重，从而提高模型在特定领域，或特定任务上的性能。预训练模型 (Pre-trained Model)，或者说基础模型 (Foundation Model)，已经可以完成很多任务，比如回答问题、总结数据、编写代码等。但是，并没有一个模型可以解决所有的问题，尤其是行业内的专业问答、关于某个组织自身的信息等，是通用大模型所无法触及的。在这种情况下，就需要使用特定的数据集，对合适的基础模型进行微调，以完成特定的任务、回答特定的问题等。

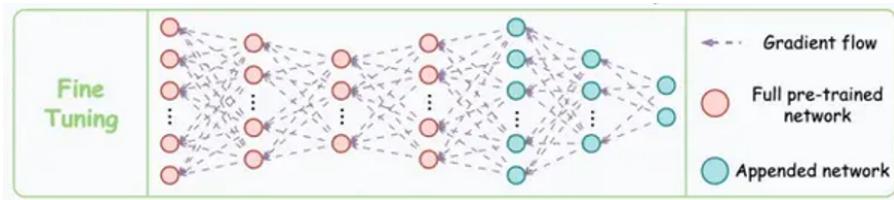


图 2: finetuning 的一种示例

2.2 LoRA

LoRA (Low-Rank Adaptation) 是一种用于在预训练模型基础上进行高效微调 (Fine-Tuning) 的算法，特别适用于大规模语言模型 (LLMs)。LoRA 通过引入低秩矩阵的方式来适应和调整模型参数，从而在保持预训练模型原有能力的同时，显著减少微调的计算成本和存储需求。

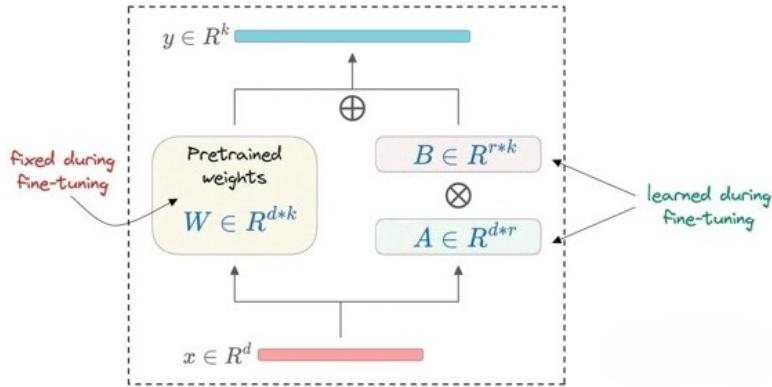


图 3: LoRA 示意图

假设 x 是输入对象参数， W 是原始的权重矩阵， ΔW 是权重的更新或调整。在 LoRA 中，这个 ΔW 被分解为两个低秩矩阵 A 和 B 的乘积。LoRA 的实现可以用下式表示。

$$(W + \Delta W)x = Wx + ABx \quad (1)$$

由于矩阵 A 和 B 的维度与 ΔW 相比要小得多，从而显著减少了可训练参数的数量。这种低秩更新不仅简洁，而且能在保留预训练后基础模型能力的同时，放大了其针对新任务或新数据集所需的特定适应性。

2.3 Stable Diffusion

Stable Diffusion 是一种文本到图像的生成方法。基于潜在扩散的机器学习模型，其主要用于根据文本的描述生成详细的图像。该模型由慕尼黑大学的 CompVis 研究团体开发，是各种生成性人工神经网络之一，由初创公司 StabilityAI、CompVis 与 Runway 合作开发，并得到 EleutherAI 和 LAION 的支持。

Stable Diffusion 的核心思想是：由于每张图片满足一定规律分布，利用文本中包含的这些分布信息作为指导，把一张纯噪声的图片逐步去噪，最终生成一张跟文本信息匹配的图片。生成图片的流程如下所示。

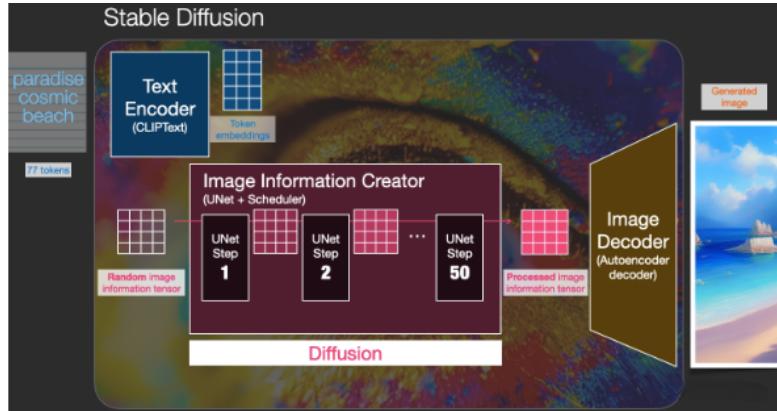


图 4: stable diffusion 图片生成流程

Stable Diffusion 由 3 个部分组成：变分自编码器 VAE、U-Net 和一个文本编码器。与其学习去噪图像数据（在“像素空间”中），而是训练 VAE 将图像转换为低维潜在空间。添加和去除高斯噪声的过程被应用于这个潜在表示，然后将最终的去噪输出解码到像素空间中。在前向扩散过程中，高斯噪声被迭代地应用于压缩的潜在表征。每个去噪步骤都由一个包含 ResNet 骨干的 U-Net 架构完成，通过从前向扩散往反方向去噪而获得潜在表征。最后 VAE 解码器通过将表征转换回像素空间来生成输出图像。

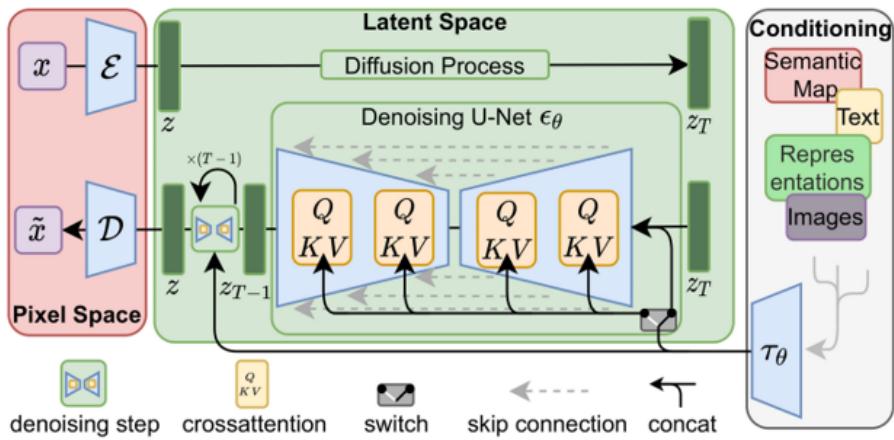


图 5: stable diffusion 模型架构

由于图像去噪过程主要是在 U-Net 模块完成的，因此 LoRA 微调也主要在这一模块上实施。

3 实验过程

3.1 预训练模型下载

输入以下代码，从魔塔社区下载 Stable Diffusion v1-5 预训练模型，并将模型放入项目根目录中。

```
1 from modelscope import snapshot_download
2 model_dir = snapshot_download('AI-ModelScope/stable-diffusion-v1-5')
```

3.2 数据集准备

由于实验环境限制，数据集选择的体量较小，为 5 张同一只狗的图片，prompt text 统一选择为“a dog”。

输入以下代码可以下载数据集，并将其放入项目根目录中。

```
1 from modelscope.msdatasets import MsDataset
2 ds = MsDataset.load('buptwq/lora-stable-diffusion-finetune',
3 → subset_name='default', split='train')
```

数据集图片如下所示：



图 6: 数据集图片

3.3 LoRA 层添加

训练代码详见 lora.py，这里主要说明一下添加 lora 层的代码以及参数设置。

首先需要冻结 unet 部分原有参数的梯度传播：

```
1 for param in unet.parameters():
2     param.requires_grad_(False)
```

之后使用 huggingface 官方提供的参数高效微调 (PEFT) 技术在 unet 部分添加 lora 层，参数高效微调 (PEFT) 可以使预训练模型高效适应下游应用，而无需微调所有模型参数。PEFT 支持广泛使用的大型语言模型低秩适应 (LoRA)。

```
1 unet_lora_config = LoraConfig(  
2     r=4,  
3     lora_alpha=4,  
4     init_lora_weights="gaussian",  
5     target_modules=["to_k", "to_q", "to_v", "to_out.0"],  
6     )  
7     unet.add_adapter(unet_lora_config)
```

这里设置了 LoRA 的低秩分解维度 (r) 为 4,LoRA 缩放参数 (lora_alpha) 也为 4，即不对 ΔW 进行缩放，lora 矩阵以高斯分布初始化，目标模块包含 “q”，“k”，“v” 以及 “out”。

3.4 图像处理设置

图像编码前使用了 transformer 模块进行 resize 以及数据增强，transformer 模块参数设置如下：

```
1 train_transforms = transforms.Compose(  
2     [  
3         transforms.Resize(256, interpolation=transforms.  
4             ↪ InterpolationMode.BILINEAR),  
5         transforms.CenterCrop(256) ,  
6         transforms.RandomHorizontalFlip() ,  
7         transforms.ToTensor(),  
8         transforms.Normalize([0.5], [0.5]),  
9     ]  
)
```

3.5 训练参数设置

本次实验 lora 微调共进行了 100 个 epoch，优化器参数设置如下：

```
1 optimizer_cls = torch.optim.AdamW  
2 optimizer = optimizer_cls(  
3     lora_layers,  
4     lr=1e-4,  
5     )
```

4 实验结果

4.1 损失曲线

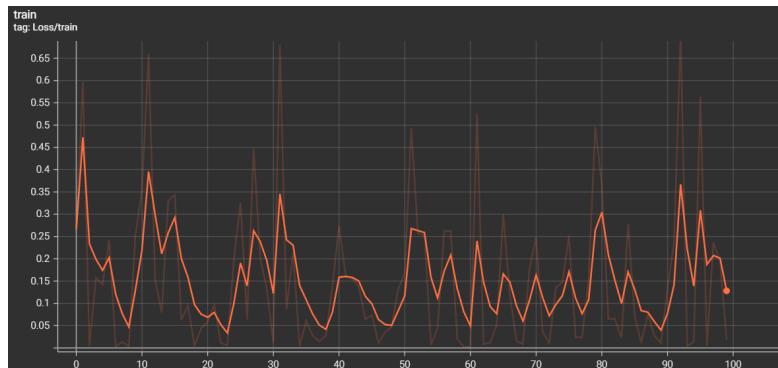


图 7: 损失曲线

可以看到 loss 收敛效果并不好，这可能与数据集规模较小，导致 lora 层没能学习到完整的模式信息有关。

4.2 图像生成情况

实验中分别采用了原模型 (stable-diffusion-v1-5) 以及 lora 微调后的模型 (fine_tuned_lora_unet)，分别生成了与 prompt 相关 (“a dog”) 以及无关 (“a cat”) 的图片，如下所示。

model:stable-diffusion-v1-5 , prompt: “a dog”



图 8: stable-diffusion-v1-5 , a dog

model:fine_tuned_lora_unet , prompt: “a dog”



图 9: fine_tuned_lora_unet , a dog

model:stable-diffusion-v1-5 , prompt: “a cat”



图 10: stable-diffusion-v1-5 , a cat

model:fine_tuned_lora_unet , prompt: “a cat”



图 11: fine_tuned_lora_unet , a cat

从生成图片效果来看，模型已经基本学习到了训练样本的一些较明显的特征（如伸舌头，毛色特征等），但是生成图片的质量不高，会出现一些比较奇怪的地方，这与训练集体量较小可能有关系。

此外一个比较有趣的地方是，在生成 a cat 的图像时，预训练模型能够生成比较多样化种类和风格的图片，而 lora 微调后的模型生成对象较为固定，并且似乎具备了一些与训练数据相似的特征（毛色）。这可能是因为猫与狗由于属于同一类别，潜在向量有部分重叠，在训练 lora 时，将部分训练后的特征映射到它们共有的潜在向量上，从而导致了特征的相似性。

4.3 与训练整个模型的有监督微调 (SFT) 对比

本实验在以上内容的基础上还对比了 lora 和 sft (全参数) 微调的训练参数量以及时长，为了方便比较，实验仅训练了一个 epoch，结果截图如下：

```
loading model...
loading dataset...
number of training parameters of lora: 797184
start training...
Epoch 1/1: 100%|██████████| 5/5 [00:12<00:00,  2.55s/it, loss=0.246]
```

图 12: lora

```

loading model...
loading dataset...
number of training parameters of sft: 859520964
start training...
Epoch 1/1: 100%|██████████| 5/5 [00:35<00:00, 7.01s/it, loss=0.0107]

```

图 13: sft

由此可见, lora 微调确实大大减少了训练的参数量 (约减少为原来的 1/1000), 并且减少了训练时间 (接近原来的 1/3)

表 1: lora 与 sft 比较

项目	lora	sft
时长	12s	35s
速度	2.55s/it	7.01s/it
参数量	797184	859520964

5 DreamBooth 微调

5.1 DreamBooth 原理简介

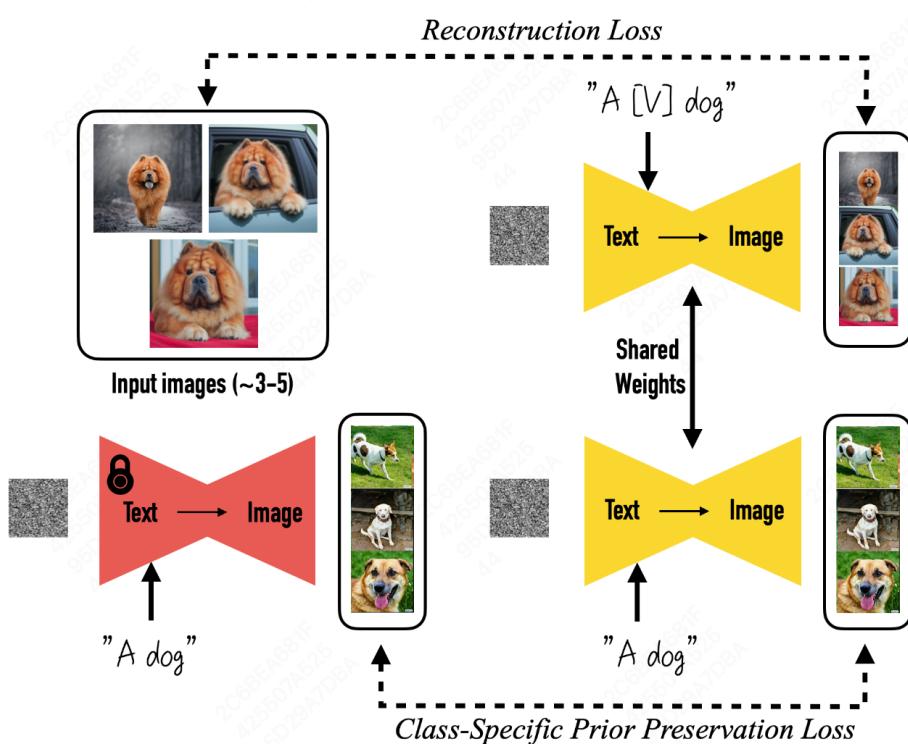


图 14: dreambooth 微调

DreamBooth 微调的想法和实现较为容易理解，主要是为了解决当前预训练模型并不能模仿给定参考图片中物体的样子在不同情景中来生成新颖图片的问题（即参考图片中的物体的样子基本不变，而改变其背景等）。

DreamBooth 作者希望将输入图片中的物体与一个特殊标识符绑定在一起，即用这个特殊标记符来表示输入图片中的物体。因此作者为微调模型设计了一种 prompt 格式：a [identifier] [class noun]，即将所有输入图片的 prompt 都设置成这种形式，其中 identifier 是一个与输入图片中物体相关联的特殊标记符，class noun 是对物体的类别描述。这里之所以在 prompt 中加入类别，是因为作者想利用预训练模型中关于该类别物品的先验知识，并将先验知识与特殊标记符相关信息进行融合，这样就可以在不同场景下生成不同姿势的目标物体。

5.2 训练过程

5.2.1 预训练模型与数据集准备

预训练模型依然采用 stable-diffusion-v1-5，在此基础上进行微调。

对于数据集的选择，DreamBooth 的一大优势就是可以选用少量的目标图片就能获得很好的结果，因此本实验依然选择上文所述的 5 张图片作为数据集的 instance data 部分，并设置 instance prompt 为“a sks dog”。实际上，本实验所采用的数据集正是 DreamBooth 微调使用的经典数据集之一。

对于数据集的 class data 部分，由于实验环境限制，本实验仅通过预训练模型 (stable-diffusion-v1-5) 生成了额外的 45 张图片，并设置 class prompt 为“a dog”。

DreamBooth 数据集形式如下图所示：

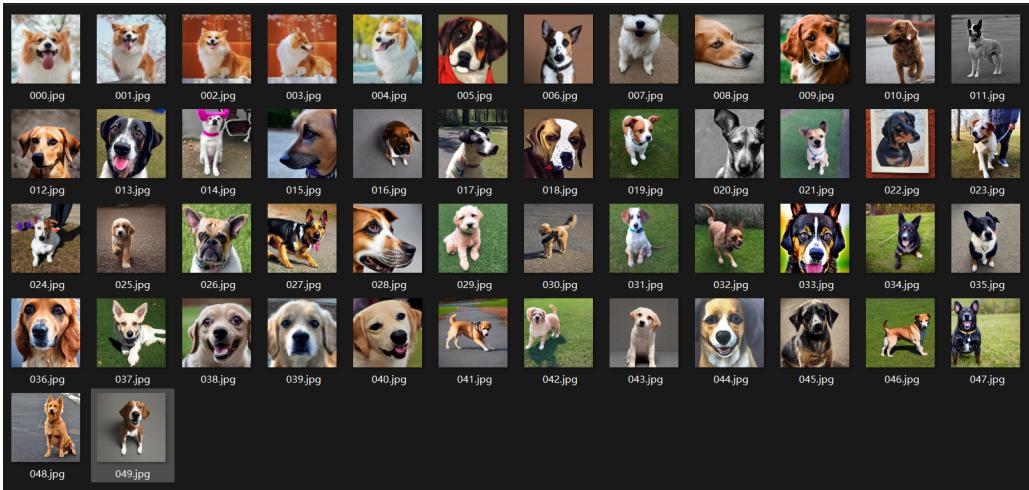


图 15: DreamBooth 数据集

5.2.2 训练参数设置

训练代码详见 `dreambooth.py`，主要参数设置与 lora 相同，训练 epoch 数为 20。

为了节省时间，本实验 `dreambooth` 微调也采用了 lora 微调技术。

5.3 训练结果

5.3.1 损失曲线

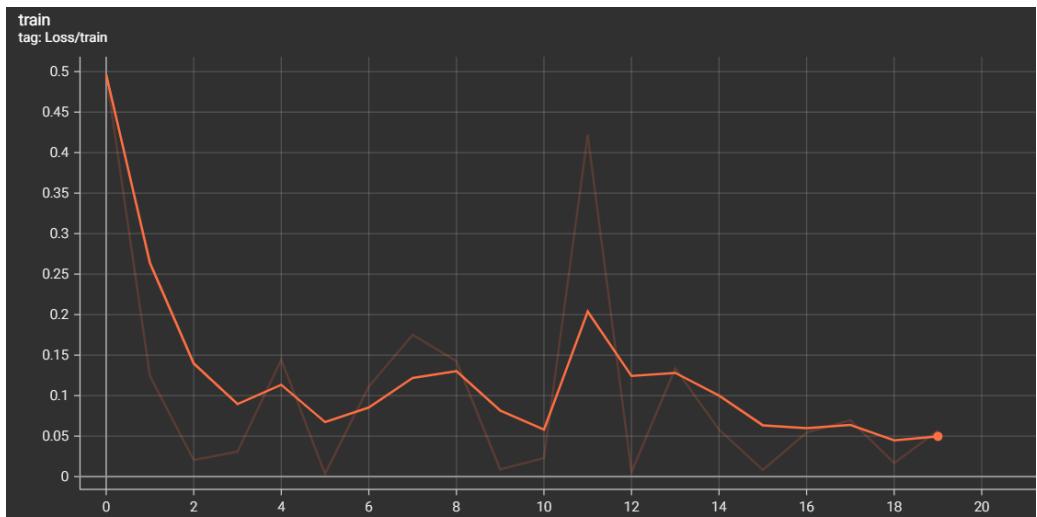


图 16: dreambooth loss

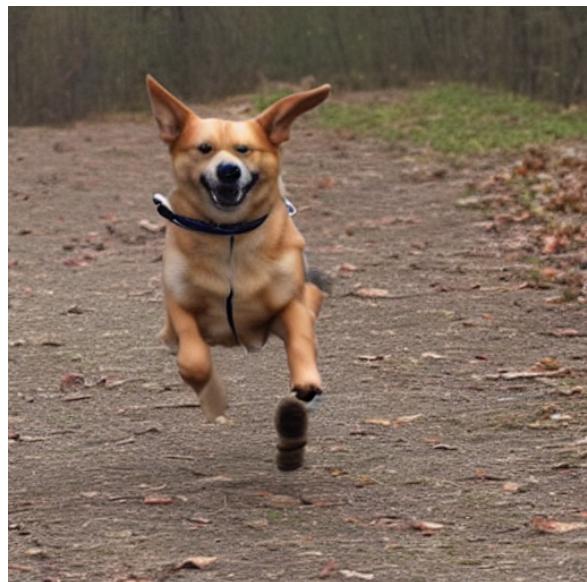
可以看到训练损失逐渐下降但还未收敛，如果调高 epoch 数量，损失应该会进一步收敛。

5.3.2 图像生成情况

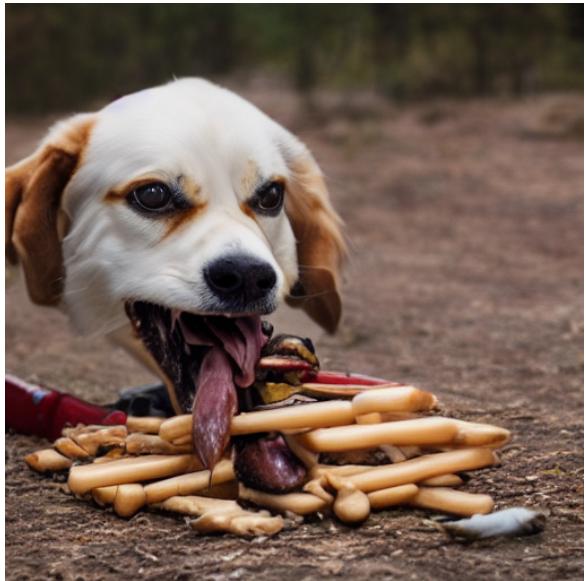
为了检验 DreamBooth 微调效果，本实验最终选用了以下 prompt: “a sks dog”、“a sks dog running”、“a sks dog eating”、“a sks dog swimming”，生成效果如下所示。



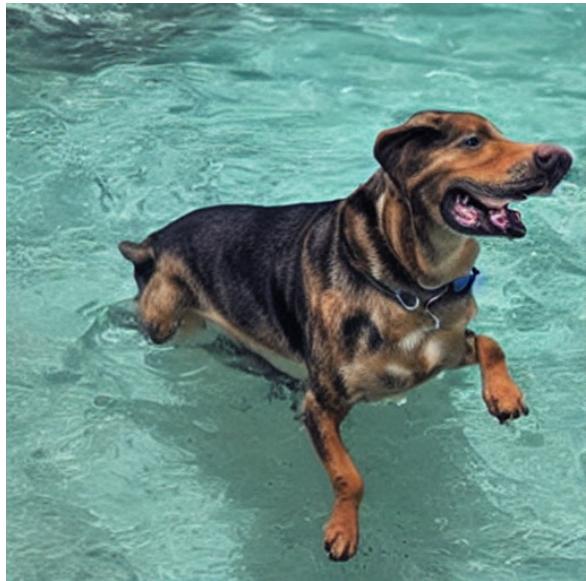
(a) a sks dog



(b) a sks dog running



(c) a sks dog eating



(d) a sks dog swimming

图 17: DreamBooth 微调效果

实现效果其实不尽人意，主要原因在于 class data 的数量太少，以及训练步数不足。原论文的 class data 数量在 200 张以上，总训练步数在 800 步以上。如果满足条件，预期结果应该如下图所示：

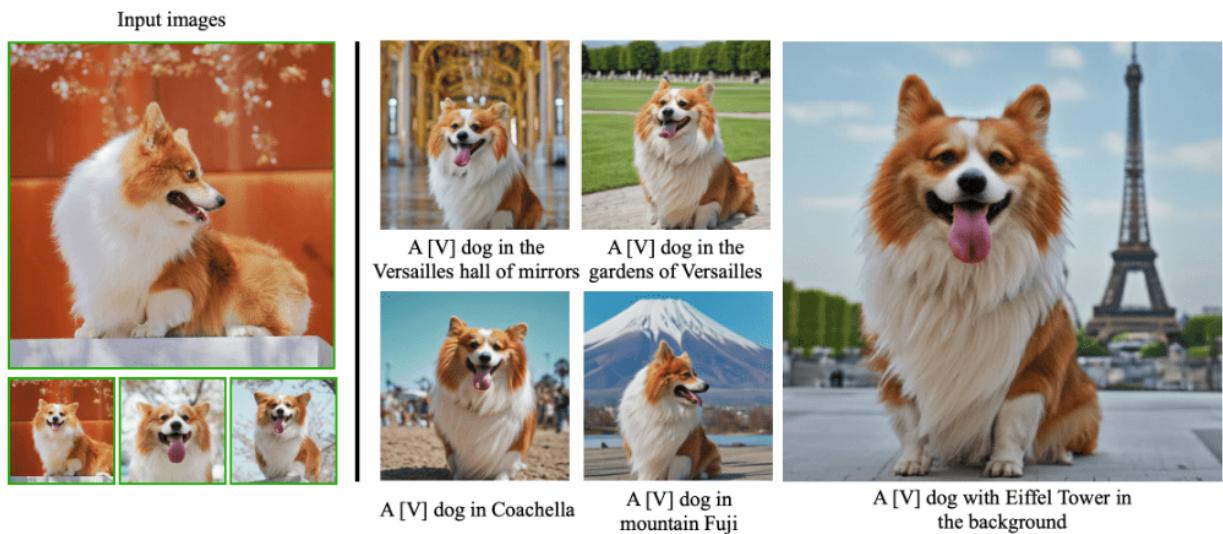


图 18: dreambooth 微调样例