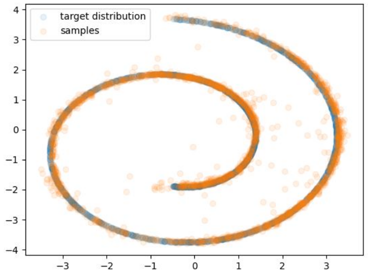
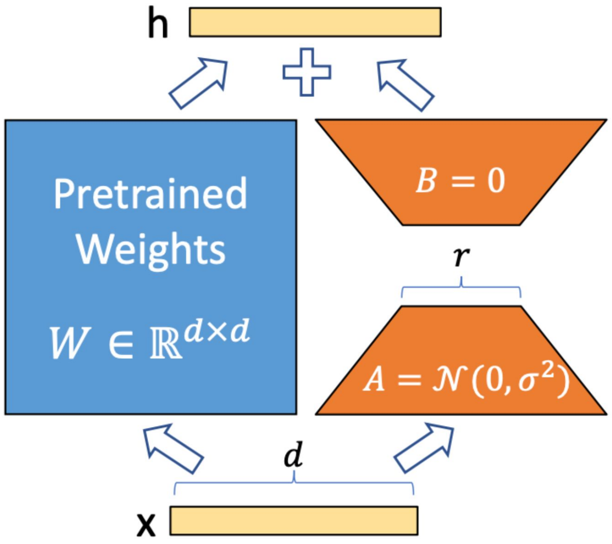
Lab2 - DDIM and LoRA





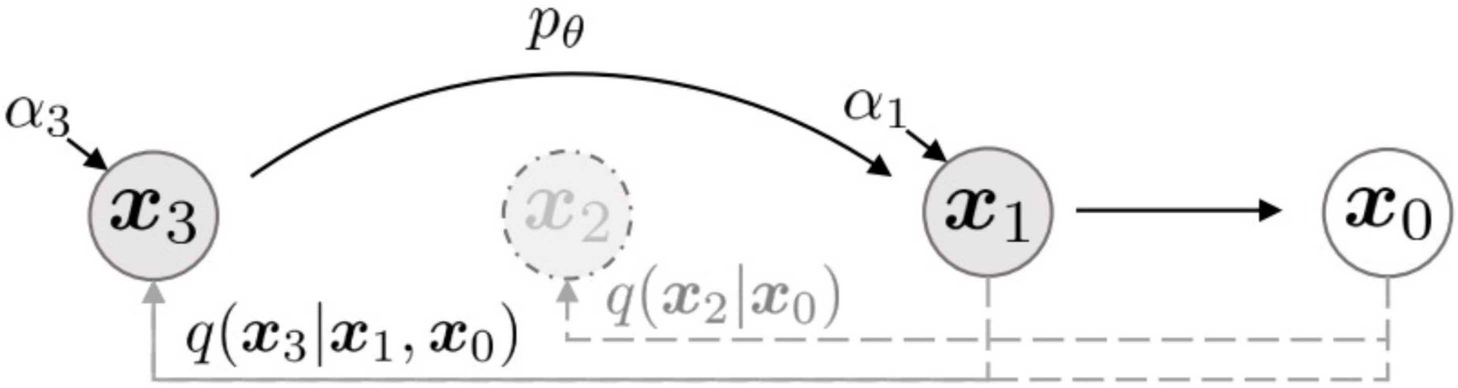
Task 1 - DDIM

[Lab2 Code (download)]

Task 2 - LoRA

Introduction

In task 1, you will implement Denoising Diffusion Implicit Models (DDIM) to accelerate the generation process of pretrained DDPM



Denoising Diffusion Implicit Models, Song et al., ICLR 2021

Introduction

In Task 2, you will train custom LoRA models with a given library

LoRA: Low-Rank Adaptation of Large Language Models, Hu et al., ICLR 2022

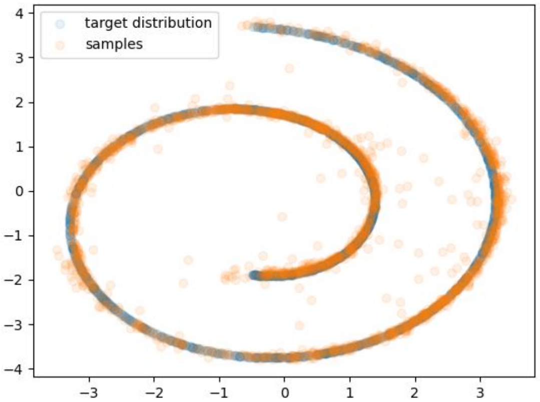
What to Do: Overview

You need to implement:

- **Task 1-1:** [2D Swiss Roll] Reverse Process of DDIMs
- **Task 1-2:** [Image Generation] DDIM Sampling
- **Task 2-1:** Train LoRA on a specific style
- **Task 2-2:** Train DreamBooth with LoRA on a Specific Identity

Task 1 : DDIM

Modify both tasks from Assignment 1 to use DDIM sampling.



Task 1 : DDIM

In the 2D example, replace a single line

$$\boldsymbol{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta^{(t)}(\boldsymbol{x}_t)\right) + \sigma_t \epsilon_t$$

in your DDPM implementation with

$$\boldsymbol{x}_{t-1} = \sqrt{\alpha_{t-1}}\left(\frac{\boldsymbol{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\boldsymbol{x}_t)}{\sqrt{\alpha_t}}\right) + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\,\epsilon_\theta^{(t)}(\boldsymbol{x}_t) + \sigma_t \epsilon_t$$

[Denoising Diffusion Implicit Models, Song et al., ICLR 2021]

Task 1-1 : DDIM [2D Swiss Roll]

#TODO1 - ddim_p_sample

- 2d_plot_diffusion_todo/ddpm.py

Reverse Process

```python
@torch.no_grad()
def ddim_p_sample(self, xt, t, t_prev, eta=0.0):
    """
    One step denoising function of DDIM: $x_t{\tau_i}$ -> $x_{\tau{i-1}}$.

    Input:
        xt (`torch.Tensor`): noisy data at timestep $\tau_i$.
        t (`torch.Tensor`): current timestep (=\tau_i)
        t_prev (`torch.Tensor`): next timestep in a reverse process (=\tau_{i-1})
        eta (float): correspond to η in DDIM which controls the stochasticity of a reverse process.
    Output:
       x_t_prev (`torch.Tensor`): one step denoised sample. (= $x_{\tau_{i-1}}$)
    """
    ######## TODO ########
    # NOTE: This code is used for assignment 2. You don't need to implement this part for assignment 1.
    # DO NOT change the code outside this part.
    # compute x_t_prev based on ddim reverse process.
    alpha_prod_t = extract(self.var_scheduler.alphas_cumprod, t, xt)
    if t_prev >= 0:
        alpha_prod_t_prev = extract(self.var_scheduler.alphas_cumprod, t_prev, xt)
    else:
        alpha_prod_t_prev = torch.ones_like(alpha_prod_t)

    x_t_prev = xt

    ######################
    return x_t_prev
```

Task 1-1 : DDIM [2D Swiss Roll]

Reverse Process

#TODO2 - ddim_p_sample_loop

- 2d_plot_diffusion_todo/ddpm.py

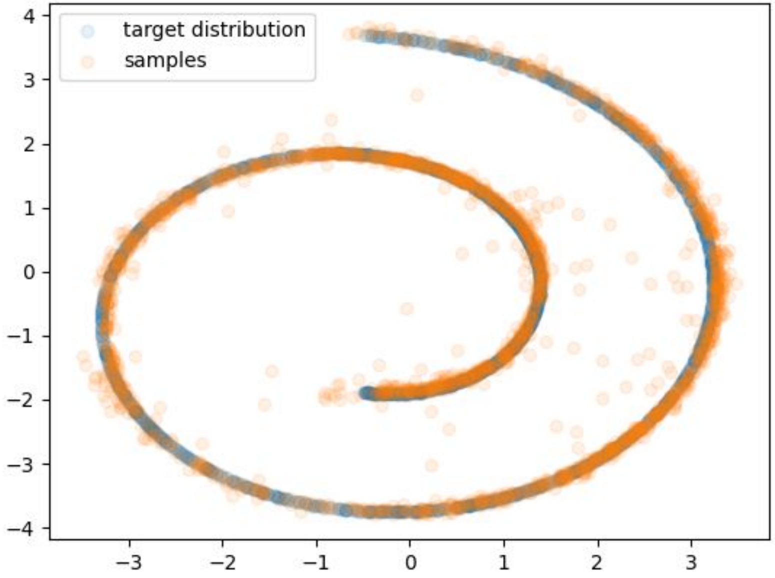
```python
@torch.no_grad()
def ddim_p_sample_loop(self, shape, num_inference_timesteps=50, eta=0.0):
    """
    The loop of the reverse process of DDIM.

    Input:
        shape (`Tuple`): The shape of output. e.g., (num particles, 2)
        num_inference_timesteps (`int`): the number of timesteps in the reverse process.
        eta (`float`): correspond to η in DDIM which controls the stochasticity of a reverse process.
    Output:
        x0_pred (`torch.Tensor`): The final denoised output through the DDPM reverse process.
    """
    ######## TODO ########
    # NOTE: This code is used for assignment 2. You don't need to implement this part for assignment 1.
    # DO NOT change the code outside this part.
    # sample x0 based on Algorithm 2 of DDPM paper.
    step_ratio = self.var_scheduler.num_train_timesteps // num_inference_timesteps
    timesteps = (
        (np.arange(0, num_inference_timesteps) * step_ratio)
        .round()[::-1]
        .copy()
        .astype(np.int64)
    )
    timesteps = torch.from_numpy(timesteps)
    prev_timesteps = timesteps - step_ratio

    xt = torch.zeros(shape).to(self.device)
    for t, t_prev in zip(timesteps, prev_timesteps):
        pass

    x0_pred = xt

    ######################

    return x0_pred
```

Task 1-1 : DDIM [2D Swiss Roll]

#TODO3 - evaluate

- 2d_plot_diffusion_todo/ddpm_tutorial.ipynb

Run all cells in 2d_plot_diffusion_todo/ddpm_tutorial.ipynb and generate 2D points via DDIM sampling

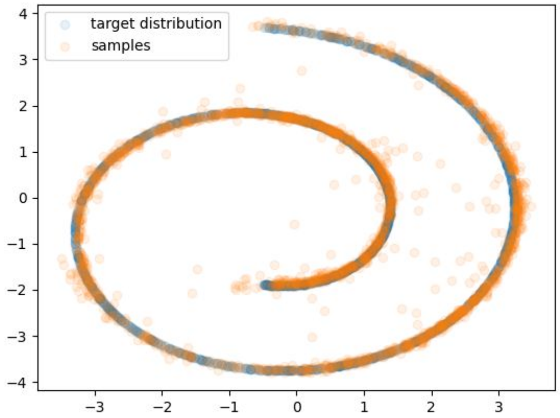


Task 1-1 : DDIM [2D Swiss Roll]

#Report

1. Complete and Explain of all #TODO code (15pts)

2. Fig of evaluation result (2.5 pts)



3. A screenshot of the Chamfer Distance measured using DDIM (2.5 pts)

Task 1-2 : DDIM [Image Generation]

#TODO1 - DDIMScheduler

- image_diffusion_todo/scheduler.py

Implement functions

- set_inference_timesteps
- step

```python
class DDIMScheduler(BaseScheduler):
    def __init__(…
    ):…

    def set_inference_timesteps(self, num_inference_timesteps: int):
        """
        Define the inference schedule (a subset of training timesteps, descending order).
        Inputs:
            num_inference_timesteps (int): number of inference steps (e.g., 50).
        """
        ######## TODO ########
        # Hint:
        #   - Define the DDIM inference schedule based on the given num_inference_timesteps.
        #   - The schedule should be a subset of training timesteps, ordered in descending fashion.
        #   - Store the result in `self.timesteps` (as a torch tensor)
        #   - Store the step ratio in `self._ddim_step_ratio` for later use when computing previous t.
        #   - Compute a `step_ratio` that maps inference steps to training steps.
        # DO NOT change the code outside this part.
        raise NotImplementedError("TODO")
        #######################

    def _get_teeth(self, consts: torch.Tensor, t: torch.Tensor):…

    @torch.no_grad()
    def step(self, x_t: torch.Tensor, t: int, eps_theta: torch.Tensor, predictor: str):
        """
        One step DDIM update: x_t -> x_{t_prev} with deterministic/stochastic control via eta.

        Input:
            x_t: [B,C,H,W]
            t: current absolute timestep index
            eps_theta: predicted noise
            predictor: predictor type
        Output:
            sample_prev: x at previous inference timestep
        """
        ######## TODO ########
        # DO NOT change the code outside this part.
        assert predictor == "noise", "In assignment 2, we only implement DDIM with noise predictor."
        sample_prev = None
        #######################
        return sample_prev
```

Note: In this task, we only implement DDIM with noise predictor

Task 1-2 : DDIM [Image Generation]

#TODO2 - sampling

- image_diffusion_todo/sampling.py

Run the following command to generate samples with DDIM:

```
python sampling.py --ckpt_path {CKPT} --save_dir {SAVE_DIR} \
--sample_method ddim \
--ddim_steps {DDIM_STEPS} --eta {ETA}
```

- Try different **DDIM_STEPS**: 10, 20, 50, 100, 1000
- Try different **ETA**: 0.0, 0.2, 0.5, 1.0

Task 1-2 : DDIM [Image Generation]

#TODO3 - evaluate

- image_diffusion_todo/dataset.py
- image_diffusion_todo/fid/measure_fid.py

STEP 1: Run `python dataset.py` once to prepare the evaluation data.

This will create the eval directory under `data/afhq`.

Do NOT forget to run this. Otherwise, you will get incorrect FIDs!

```
data
└── afhq
    ├── eval
    ├── train
    └── val
```

STEP 2: Run the following command to evaluate FID:

```
python fid/measure_fid.py data/afhq/eval/ {SAMPLE_DIR}
```

Task 1-2 : DDIM [Image Generation]

#Report

1. Complete and Explain of all #TODO code (20pts)

2. Provide a table of evaluation results. (10 pts)

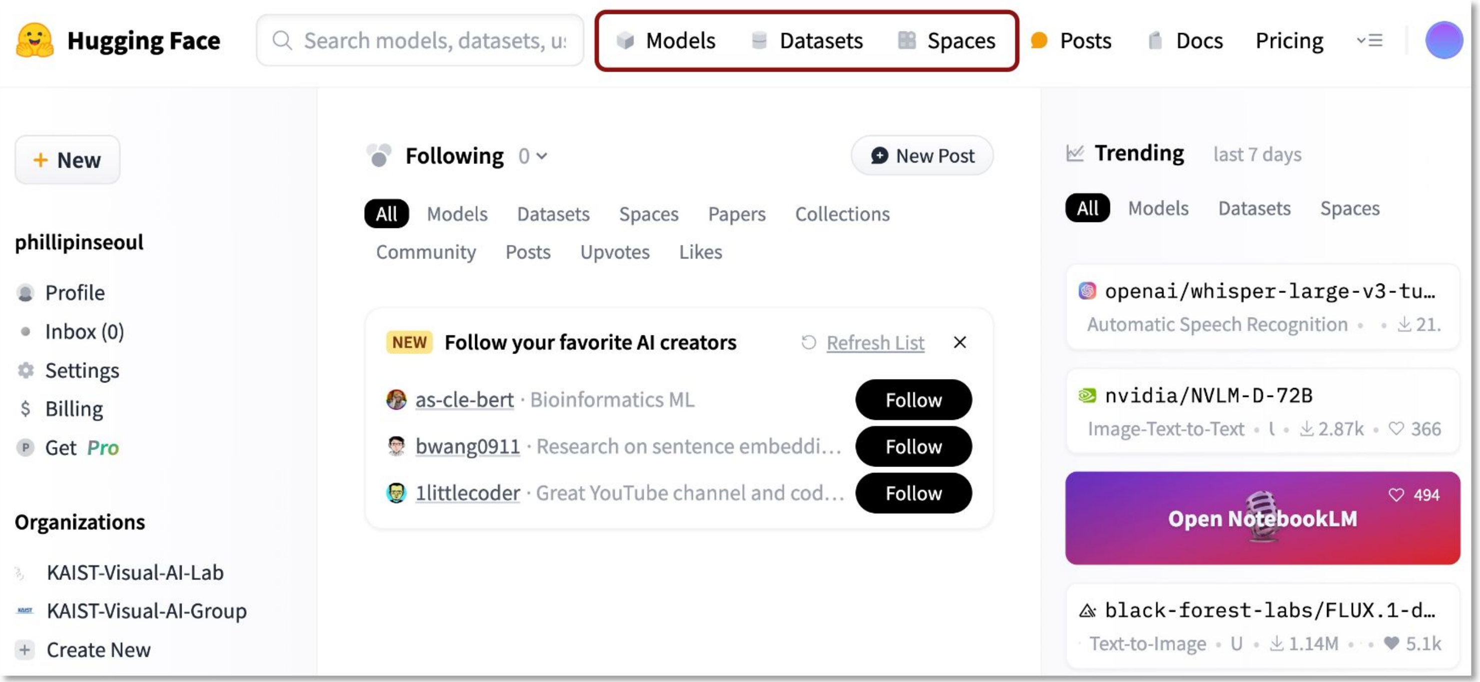
3. Discuss and explain the evaluation results. (10 pts)

FID		S				
		10	**20**	**50**	**100**	**1000**
η	**0.0**	18.24	18.66	21.77	25.55	25.30
	0.2	12.40	16.04	13.91	17.35	19.18
	0.5	16.35	10.48	8.05	6.81	5.80
	1.0	29.18	20.02	9.36	6.27	4.73

Introduction to
Hugging Face and Diffusers

Hugging Face 🤗

An open-source platform that serves as a hub for machine learning applications.



Diffusers Library   D🧨ffusers

Popular Tasks & Pipelines

Task	Pipeline	🤗 Hub
Unconditional Image Generation	DDPM	google/ddpm-ema-church-256
Text-to-Image	Stable Diffusion Text-to-Image	runwayml/stable-diffusion-v1-5
Text-to-Image	unclip	kakaobrain/karlo-v1-alpha
Text-to-Image	DeepFloyd IF	DeepFloyd/IF-I-XL-v1.0
Text-to-Image	Kandinsky	kandinsky-community/kandinsky-2-2-decoder
Text-guided Image-to-Image	Controlnet	lllyasviel/sd-controlnet-canny
Text-guided Image-to-Image	Instruct Pix2Pix	timbrooks/instruct-pix2pix
Text-guided Image-to-Image	Stable Diffusion Image-to-Image	runwayml/stable-diffusion-v1-5
Text-guided Image Inpainting	Stable Diffusion Inpaint	runwayml/stable-diffusion-inpainting
Image Variation	Stable Diffusion Image Variation	lambdalabs/sd-image-variations-diffusers
Super Resolution	Stable Diffusion Upscale	stabilityai/stable-diffusion-x4-upscaler
Super Resolution	Stable Diffusion Latent Upscale	stabilityai/sd-x2-latent-upscaler

e.g. Loading Stable Diffusion from diffusers

```python
import torch
from diffusers import StableDiffusionPipeline

model_id = "runwayml/stable-diffusion-v1-5"
pipe = StableDiffusionPipeline.from_pretrained(
    model_id,
    torch_dtype=torch.float16
)
pipe = pipe.to("cuda")

prompt = "a photo of an astronaut riding a horse on mars"
image = pipe(prompt).images[0]

image.save("astronaut_rides_horse.png")
```

Task 2: Training Custom LoRA Models

#TODO0

You need access to Hugging Face to proceed with Task 2 :

1. Sign into Hugging Face.
2. Obtain your access token at https://huggingface.co/settings/tokens.
3. From your terminal, log into Hugging Face using

```
$ huggingface-cli login
```

and enter your Access Token.

Task 2: Training Custom LoRA Models

#TODO0

4. To check the access to Hugging Face, download Stable Diffusion from Hugging Face and generate an image with it:

```
import torch
from diffusers import StableDiffusionPipeline

model_id = "CompVis/stable-diffusion-v1-4"
device = "cuda"


pipe = StableDiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16)
pipe = pipe.to(device)

prompt = "a photo of an astronaut riding a horse on mars"
image = pipe(prompt).images[0]

image.save("astronaut_rides_horse.png")
```

Task 2: Training Custom LoRA Models

Goal: Train custom LoRA models on different datasets.

Generated with GPT4

"A man with sunglasses"

Task 2: Training Custom LoRA Models

No implementation required!

The main objective of this task is to :

- Gain hands-on experience on customizing diffusion models using LoRA.
- Explore creative use-cases of state-of-the-art diffusion models.

Task 2-1: Train LoRA on a specific style.

#TODO1 - train

You can either use an open-source dataset and train with

```
$ sh scripts/train_lora.sh
```

Or create a custom dataset and train with

```
$ sh scripts/train_lora_custom.sh
```

Task 2-1: Train LoRA on a specific style.

#TODO1 - train

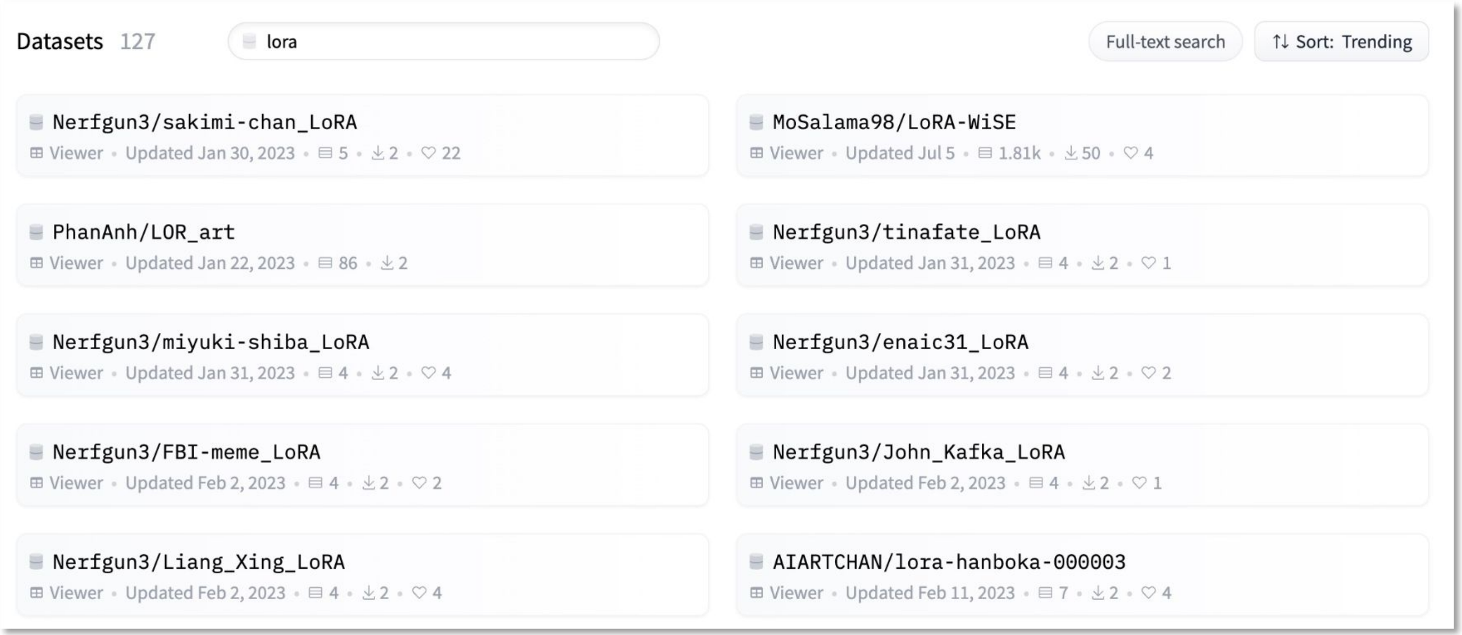
In either case, just simply set the path to the dataset by

```
$ export DATASET_NAME=”$PATH_TO_DATASET”
```

in the given script file.

Task 2-1: Train LoRA on a specific style.

TIP: You can find many open-source LoRA training datasets on Hugging Face.



https://huggingface.co/datasets?modality=modality:image&size_categories=or:%28size_categories:n%3C1K,size_categories:1K%3Cn%3C10K%29&sort=trending&search=lora

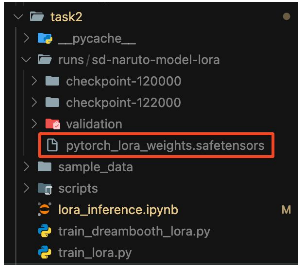
Task 2-2: Train LoRA on a specific identity using DreamBooth + LoRA.

#TODO1 - train

Run `$ sh scripts/train_dreambooth_lora.sh` for training.

Generated with GPT4

Task 2 : Training Custom LoRA Models



#TODO2 - inference

- task_2_lora/lora_inference.ipynb

Load Stable Diffusion

```
pipe = StableDiffusionPipeline.from_pretrained(
    "CompVis/stable-diffusion-v1-4",
    torch_dtype=torch.float16
)
print("[INFO] Successfully loaded Stable Diffusion!")
```

Load LoRA weights

Change to your LoRA weights!

```
# lora_path = "./runs/sd-naruto-model-lora"
lora_path = "./runs/artistic_custom"
# lora_path = "./runs/dreambooth_cat"
# lora_path = None  # if not using LoRA

if lora_path is not None:
    pipe.load_lora_weights(lora_path)
    print("[INFO] Successfully loaded LoRA weights!")

pipe = pipe.to(device)
```

[22]

```
[INFO] Successfully loaded LoRA weights!
```

Inference

Change prompt and seed for diverse outputs!

```
prompt = "a man with sunglasses"

seed = 10

seed_everything(seed)

image = pipe(
    prompt,
    num_inference_steps=30,
    guidance_scale=7.5
).images[0]

image
```

[27]

```
100%|██████████| 30/30 [00:01<00:00, 22.53it/s]
```

Task 2 : Training Custom LoRA Models

#Report

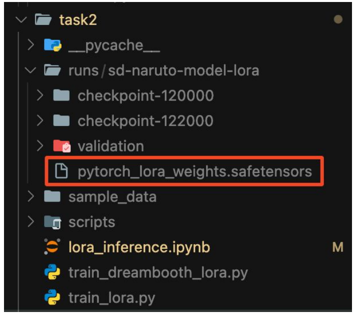
1. [Task 2-1] Description on the dataset used, including its source. (5 pts)
2. [Task 2-1] Visualization of training images and generated image with the corresponding text prompts. (15 pts)
3. [Task 2-2] Description on the dataset used, including its source. (5 pts)
4. [Task 2-2] Visualization of training images and generated image with the corresponding text prompts. (15 pts)

Submit

Create and zip a single folder named {ID}_lab2.zip including:

- The PDF file formatted following the guideline
- **(Task1)** The complete code for Task 1, excluding checkpoints and datasets.
- **(Task2)** Provide the LoRA weight cloud link in a text file. The file name should be: {ID}_task2-1_lora_weight.

```
{ID}_lab2.zip
├── report.pdf
├── task1
│   ├── 2d_plot_diffusion_todo
│   │   ├── __init__.py
│   │   ├── chamferdist.py
│   │   ├── dataset.py
│   │   ├── ddpm_tutorial.ipynb
│   │   ├── ddpm.py
│   │   └── network.py
│   └── image_diffusion_todo
│       ├── dataset.py
│       ├── model.py
│       ├── module.py
│       ├── network.py
│       ├── sampling.py
│       ├── scheduler.py
│       └── train.py
└── task2
    ├── {ID}_task2-1_lora_weight.txt
    └── {ID}_task2-2_lora_weight.txt
```



Thank you