CS492D: Diffusion Models and Their Applications

Assignment 2 Session

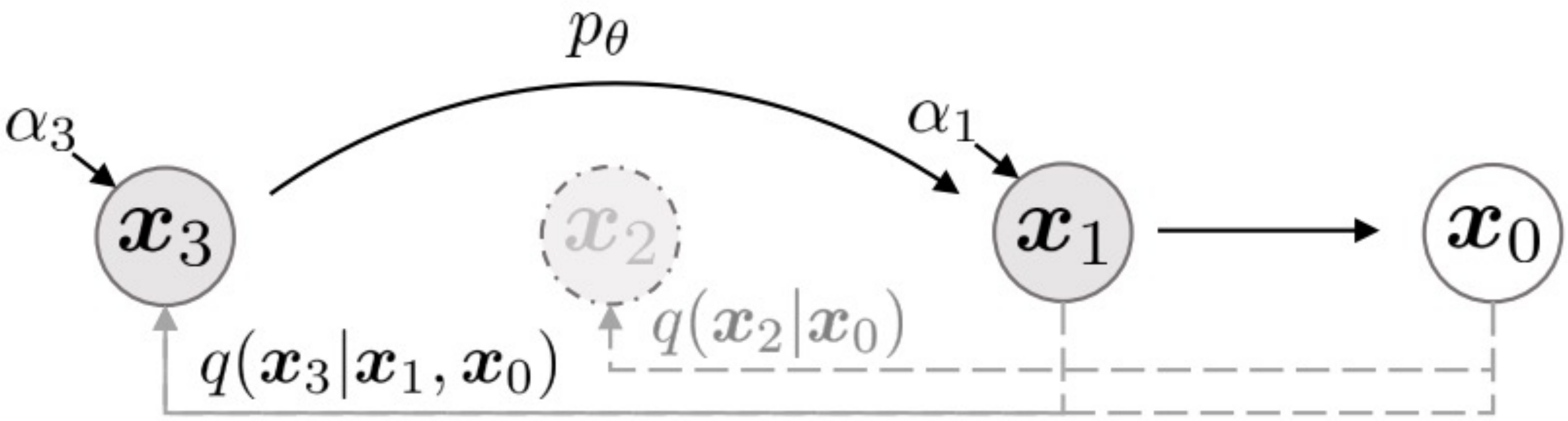
SEUNGWOO YOO

Fall 2024
KAIST

How was Assignment 1?

Introduction

In Assignment 2, we implement **Denoising Diffusion Implicit Models (DDIM)** to accelerate the generation process of pretrained DDPMs.



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

Introduction

We also explore **Classifier-Free Guidance (CFG)**, a simple technique to enhance image quality in conditional generation.

“Pembroke Welsh corgi”

Weak Guidance Scale

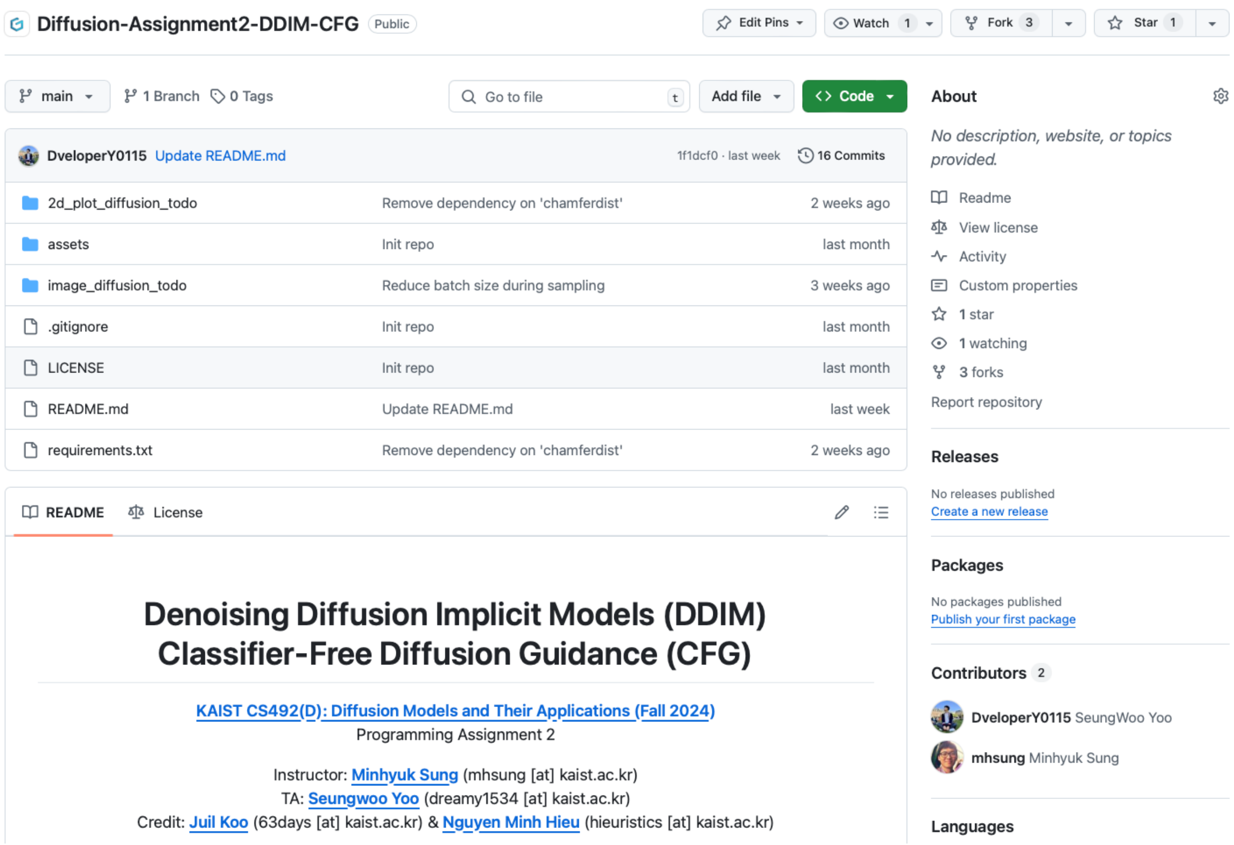
Strong Guidance Scale

Diffusion Models Beat GANs on Image Synthesis, Dhariwal and Nichol, PMLR 2021

Introduction

The skeleton code and instructions are available at:

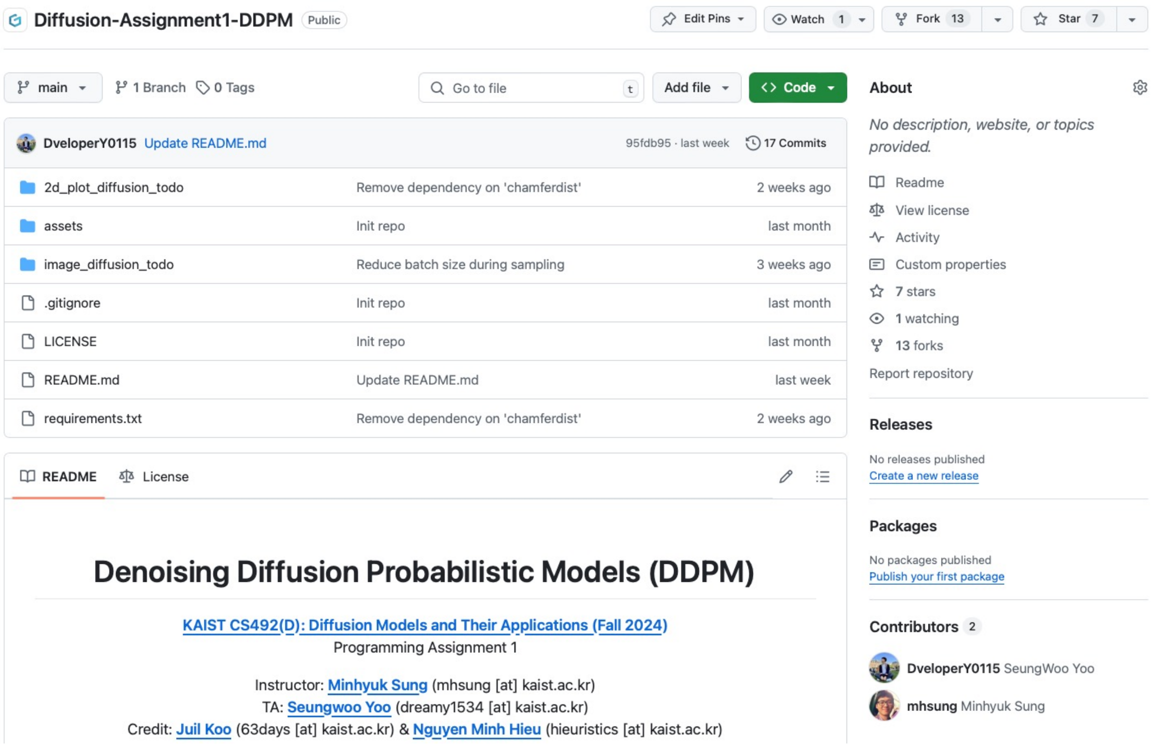
https://github.com/KAIST-Visual-AI-Group/Diffusion-Assignment2-DDIM-CFG

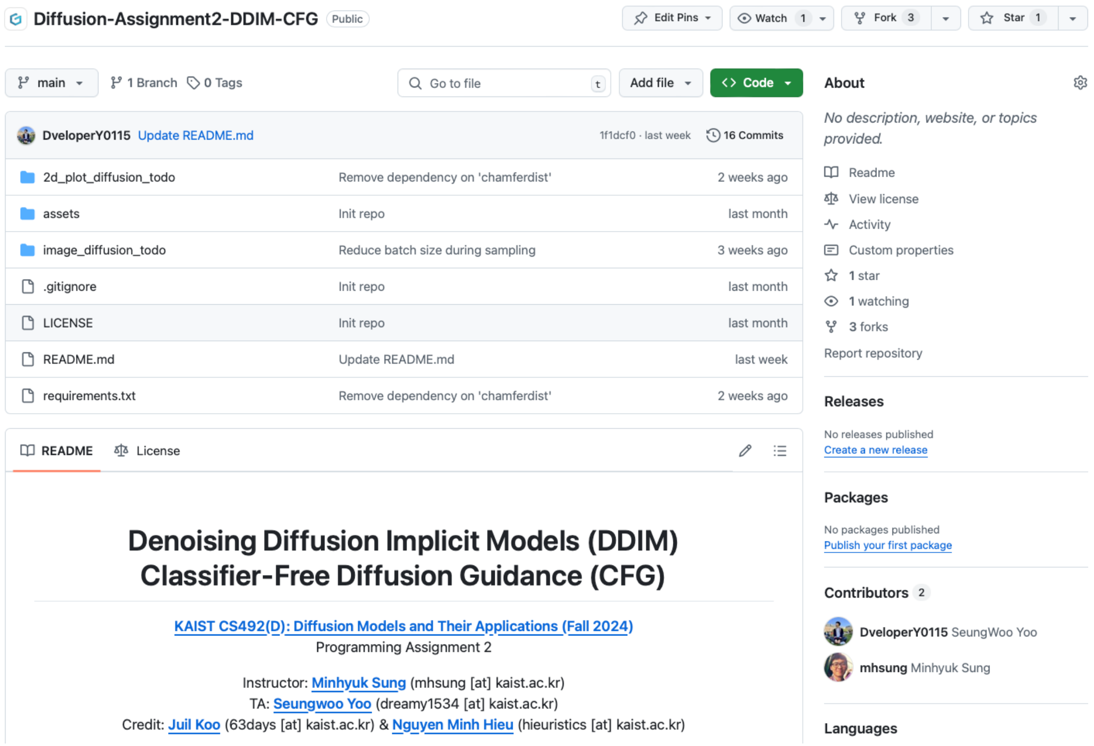


Introduction

Copy and paste your DDPM implementation from Assignment 1.

We assume that you completed it, so finish it before starting Assignment 2.





Important Notes

- All programming assignments are due **two weeks** after the assignment session.
- Late submission will incur **20% penalty** for **each** late day!
- Please carefully check the README of each assignment.
- Missing items in your submission will also incur penalties.

What to Do: Overview

You need to implement:

- [2D Swiss Roll] Reverse Process of DDIMs
- [AFHQ] Class Conditioning Mechanism in U-Net
- [AFHQ] CFG Training and Sampling

What to Do: Task 1

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

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

What to Do: Task 1

Implement functions

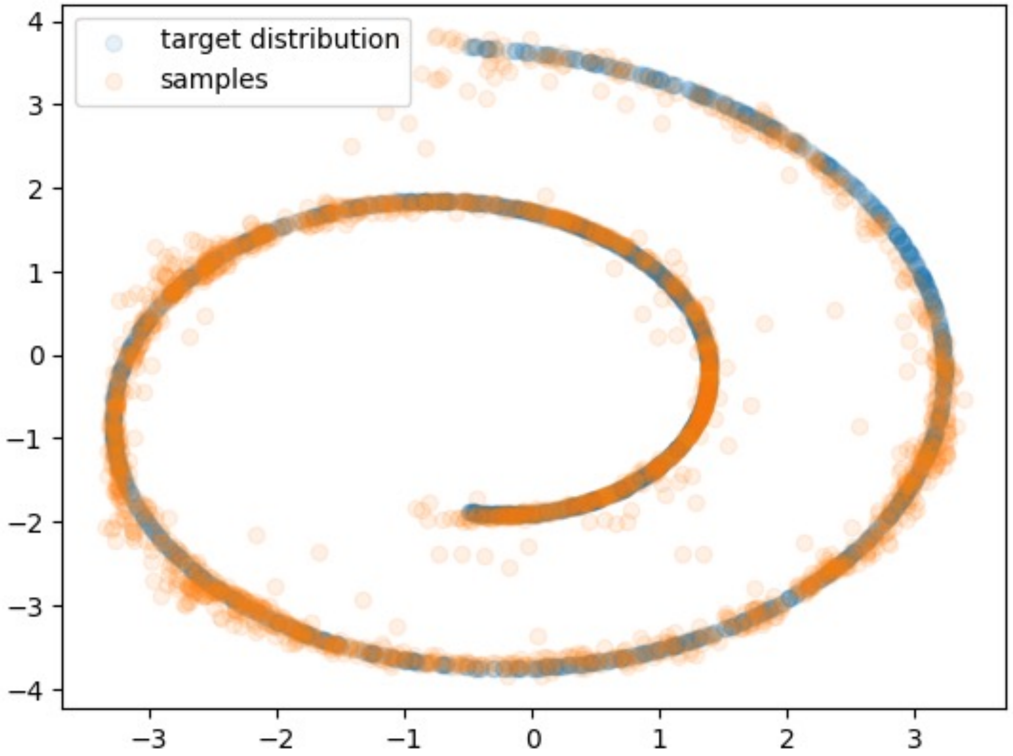
- `ddim_p_sample`
- `ddim_p_sample_loop`

```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

@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)
```

2d_plot_diffusion_todo/ddpm.py

What to Do: Task 1

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



What to Do: Task 2

The AFHQ dataset we used previously contains images of 3 classes.

Wildlife

Cat

Dog

What to Do: Task 2

We will use **one-hot encoding** to distinguish different classes.

Wildlife
(100)

Cat
(001)

Dog
(010)

What to Do: Task 2

Add a class conditioning mechanism in our U-Net implementation.

Hint 1: Use `self.class_embedding`

Hint 2: Add class embeddings to `temb`

```python
def forward(self, x, timestep, class_label=None):
    # Timestep embedding
    temb = self.time_embedding(timestep)

    if self.use_cfg and class_label is not None:
        if self.training:
            assert not torch.any(class_label == 0) # 0 for null.

            ######## TODO ########
            # DO NOT change the code outside this part.
            # Assignment 2-2. Implement random null conditioning in CFG training.
            raise NotImplementedError("TODO")
            #######################

        ######## TODO ########
        # DO NOT change the code outside this part.
        # Assignment 2-1. Implement class conditioning
        raise NotImplementedError("TODO")
        #######################
```

`image_diffusion_todo/network.py`

What to Do: Task 2

Randomly replace some class labels to null (000) vector for CFG training.

Algorithm 1 Joint training a diffusion model with classifier-free guidance

Require: p_{uncond}: probability of unconditional training

1: **repeat**
2: $(\mathbf{x}, \mathbf{c}) \sim p(\mathbf{x}, \mathbf{c})$ ▷ Sample data with conditioning from the dataset
3: $\mathbf{c} \leftarrow \varnothing$ with probability p_{uncond} ▷ Randomly discard conditioning to train unconditionally
4: $\lambda \sim p(\lambda)$ ▷ Sample log SNR value
5: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6: $\mathbf{z}_\lambda = \alpha_\lambda \mathbf{x} + \sigma_\lambda \boldsymbol{\epsilon}$ ▷ Corrupt data to the sampled log SNR value
7: Take gradient step on $\nabla_\theta \left\| \boldsymbol{\epsilon}_\theta(\mathbf{z}_\lambda, \mathbf{c}) - \boldsymbol{\epsilon} \right\|^2$ ▷ Optimization of denoising model
8: **until** converged

```python
def forward(self, x, timestep, class_label=None):
    # Timestep embedding
    temb = self.time_embedding(timestep)

    if self.use_cfg and class_label is not None:
        if self.training:
            assert not torch.any(class_label == 0) # 0 for null.

            ######## TODO ########
            # DO NOT change the code outside this part.
            # Assignment 2-2. Implement random null conditioning in CFG training.
            raise NotImplementedError("TODO")
            #######################

        ######## TODO ########
        # DO NOT change the code outside this part.
        # Assignment 2-1. Implement class conditioning
        raise NotImplementedError("TODO")
        #######################
```

`image_diffusion_todo/network.py`

What to Do: Task 2

Implement noise computation with CFG.

Algorithm 2 Conditional sampling with classifier-free guidance

Require: w: guidance strength
Require: $\mathbf{c}$: conditioning information for conditional sampling
Require: $\lambda_1, \ldots, \lambda_T$: increasing log SNR sequence with $\lambda_1 = \lambda_{\min}$, $\lambda_T = \lambda_{\max}$

$\mathbf{z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = 1, \ldots, T$ **do**
  ▷ Form the classifier-free guided score at log SNR λ_t
  $\tilde{\boldsymbol{\epsilon}}_t = (1 + w)\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, \mathbf{c}) - w\boldsymbol{\epsilon}_\theta(\mathbf{z}_t)$
  ▷ Sampling step (could be replaced by another sampler, e.g. DDIM)
  $\tilde{\mathbf{x}}_t = (\mathbf{z}_t - \sigma_{\lambda_t}\tilde{\boldsymbol{\epsilon}}_t)/\alpha_{\lambda_t}$
  $\mathbf{z}_{t+1} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_{\lambda_{t+1}|\lambda_t}(\mathbf{z}_t, \tilde{\mathbf{x}}_t), (\tilde{\sigma}^2_{\lambda_{t+1}|\lambda_t})^{1-v}(\sigma^2_{\lambda_t|\lambda_{t+1}})^v)$ if $t < T$ else $\mathbf{z}_{t+1} = \tilde{\mathbf{x}}_t$
end for
return $\mathbf{z}_{T+1}$

```python
@torch.no_grad()
def sample(
    self,
    batch_size,
    return_traj=False,
    class_label: Optional[torch.Tensor] = None,
    guidance_scale: Optional[float] = 1.0,
):
    x_T = torch.randn([batch_size, 3, self.image_resolution, self.image_resolution]).to(self.device)

    do_classifier_free_guidance = guidance_scale > 1.0

    if do_classifier_free_guidance:

        ######## TODO ########
        # Assignment 2-3. Implement the classifier-free guidance.
        # Specifically, given a tensor of shape (batch_size,) containing class labels,
        # create a tensor of shape (2*batch_size,) where the first half is filled with zeros (i.e., null condition).
        assert class_label is not None
        assert len(class_label) == batch_size, f"len(class_label) != batch_size. {len(class_label)} != {batch_size}"
        raise NotImplementedError("TODO")
        #######################

    traj = [x_T]
    for t in tqdm(self.var_scheduler.timesteps):
        x_t = traj[-1]
        if do_classifier_free_guidance:
            ######## TODO ########
            # Assignment 2. Implement the classifier-free guidance.
            raise NotImplementedError("TODO")
            #######################
        else:
            noise_pred = self.network(x_t, timestep=t.to(self.device))

        x_t_prev = self.var_scheduler.step(x_t, t, noise_pred)

        traj[-1] = traj[-1].cpu()
        traj.append(x_t_prev.detach())
```

image_diffusion_todo/model.py

What to Do: Task 2

After implementing the functions, start training the model by running

```
python train.py --use-cfg
```

⚠️ **Do NOT forget to add the flag `--use-cfg`.**

⚠️ **Otherwise, the trained model CANNOT be used for CFG sampling!**

What to Do: Task 2

After training your model, generate samples by

```
python sampling.py –ckpt-path {CKPT PATH} \
    --save-dir {SAVE DIR}
    --use-cfg --cfg-scale {CFG Scale}
```

Try CFG scale of 0.0 and 7.5 and observe how sample quality changes.

⚠️ **Just like before, do NOT forget to add flag `--use-cfg`.**

What to Do: Task 2

Compute the FID by running

```
Python fid/measure_fid.py {GT_DIR} {GEN_DIR}
```

Specifically, use

- GT_DIR: `data/afhq/eval`
- GEN_DIR: The path you passed as `save_dir` to `sampling.py`

What to Submit

Compile the following items into a PDF file: `{NAME}_{ID}.pdf`.

Task 1

- A screenshot of the Chamfer Distance measured using DDIM;
- A visualization of samples generated using your DDIM.

Task 2

- A screenshot of the computed FIDs with CFG scale= {0.0, 7.5}
- 8 images generated with CFG scale = {0.0, 7.5} (16 images in total)

What to Submit

Create a single ZIP file `{NAME}_{ID}.zip` including:

- The PDF file, formatted according to the guideline;
- Your code *without* any model checkpoints, training data, and outputs.

Your score will be deducted by 10% for *each* missing item.

Please check carefully!

Grading

You will receive up to 20 points from this assignment.

Task 1

- 10 points: Achieve CD lower than 60 in DDIM sampling.
- 5 points: Achieve CD greater, or equal to 60 and less than 80.
- 0 point: Otherwise.

Grading

You will receive up to 20 points from this assignment.

Task 2

- 10 points: Achieve FID lower than 30 in both CFG scales = {0.0, 7.5}.
- 5 points: Achieve FID between 30 and 50 in one of the two CFG scales.
- 0 point: Otherwise.

Thank You