Spring 2025
Introduction to Artificial Intelligence

Offline Learning Module 3: Generative AI

1. Implementation & Results

Part 1: Anime Face Generation(65%)

Part 1-1: Denoising Process (15%)

- init()

The content of the init() function is to generate the required alphas and alphas_bar . This is done th rough simple subtraction and the functions provided by PyTorch.

```
##############################
# TODO1-1-1: Denoising Process - Initialization
# Begin your code
def cosine_beta_schedule(timesteps):
    s = 0.008
    f = lambda t: math.cos((t / timesteps + s) / (1 + s) * math.pi * 0.5) ** 2
    beta = lambda t: min(0.999, 1 - f(t) / f(t - 1))
    return torch.Tensor([beta(t) for t in range(1, timesteps + 1)])

if beta_schedule == 'linear':
    beta_schedule_fn = linear_beta_schedule
elif beta_schedule == 'cosine':
    beta_schedule_fn = cosine_beta_schedule
else:
    raise ValueError(f'unknown beta schedule {beta_schedule}')

betas = beta_schedule_fn(timesteps, **schedule_fn_kwargs)

alphas = 1 - betas
alphas_bar = torch.cumprod(alphas, dim=0)
return betas, alphas, alphas_bar

# End your code
##############################
```

The corresponding mathematical operations are as follows. Since the math function in word is terri ble, all the equations are shown in picture.

$$\text{Given } \beta_t;\ \alpha_t = 1 - \beta_t,\ \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$$

- add_noise_forward()

The function to add noise is completed according to the formula given in spec and refer to the app endix method.

```
##############################
# TODO1-1-2: Denoising Process - Forward function
# Begin your code

alphas_bar = self.get_buffer("alphas_bar")
x_t = noise * ((1 - alphas_bar[t]) ** 0.5).view(-1, 1, 1, 1)
x_t += ((alphas_bar[t]) ** 0.5).view(-1, 1, 1, 1) * x_start

return x_t
# End your code
##############################
```

The corresponding mathematical operations are as follows:

$$q(x_t|x_0) = \mathcal{N}(x_t;\ \sqrt{\bar{\alpha}_t}x_0,\ (1 - \bar{\alpha}_t)I)$$

With the contents of appendix, we can further get the following equation.

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + (1 - \bar{\alpha}_t) \times \epsilon \quad \text{where } \epsilon \sim \mathcal{N}(0, 1)$$

- denoise_backward()

This function also updates the predicted results according to the content of appendix. Note that the original x_t should be used as x_start.

If x_t is regarded as "image of step t" according to the comment the parameters that call this functi on in section 1-2-1 need to be modified. I confirmed with the TA and learned that there was an err or in the comment.

```
##############################
# TODO1-1-3: Denoising Process - Backward function
# Begin your code
# x_t for x_start
alphas_bar = self.get_buffer("alphas_bar")
ab_t = alphas_bar[t]
ab_tp = alphas_bar[t_prev] if t_prev >= 0 else torch.tensor(1.0, device=x_t.device)

sigma_t = eta * ((1 - ab_tp) / (1 - ab_t)) ** 0.5 \
    * (1 - ab_t / ab_tp) ** 0.5
ut_xt = (ab_tp) ** 0.5 * x_t \
    + (1 - ab_tp - sigma_t ** 2) ** 0.5 * pred_noise
x_prev = noise * sigma_t + ut_xt

return x_prev
# End your code
##############################
```

The corresponding mathematical operations are as follows:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\ \mu_t(x_t), \sigma_t^2 I)$$

$$\mu_t(x_t) = \sqrt{\overline{\alpha}_{t-1}}\hat{x}_0 + \sqrt{1 - \overline{\alpha}_{t-1} - \sigma_t^2} \cdot \epsilon_\theta(x_t, t)$$

$$\sigma_t = \eta \cdot \sqrt{\frac{(1 - \overline{\alpha}_{t-1})}{(1 - \overline{\alpha}_t)}} \cdot \sqrt{1 - \frac{\overline{\alpha}_t}{\overline{\alpha}_{t-1}}}$$

With the contents of appendix, we can further get the following equation.

$$x_{t-1} = \mu_t(x_t) + \sigma_t \times \epsilon$$

Part 1-2: Result Visualization (10%)

- Describe how you visualized the denoising progress and loss curves.

There is no requirement to provide screenshots here, so I will just stick to the simple description.

To demonstrate the denoise process, we can arrange the image outputs at different time steps in or der to see how the image is gradually restored from noise to a clear image.

In practice, this code uses matplotlib.pyplot to create a 5-row × N-column grid image, where N is t he number of steps in the denoise process. Each column corresponds to the image output of a time step, and the first 5 images are selected for display. This visualization method can clearly show ho w the model gradually removes noise and restores the original image at each time step.

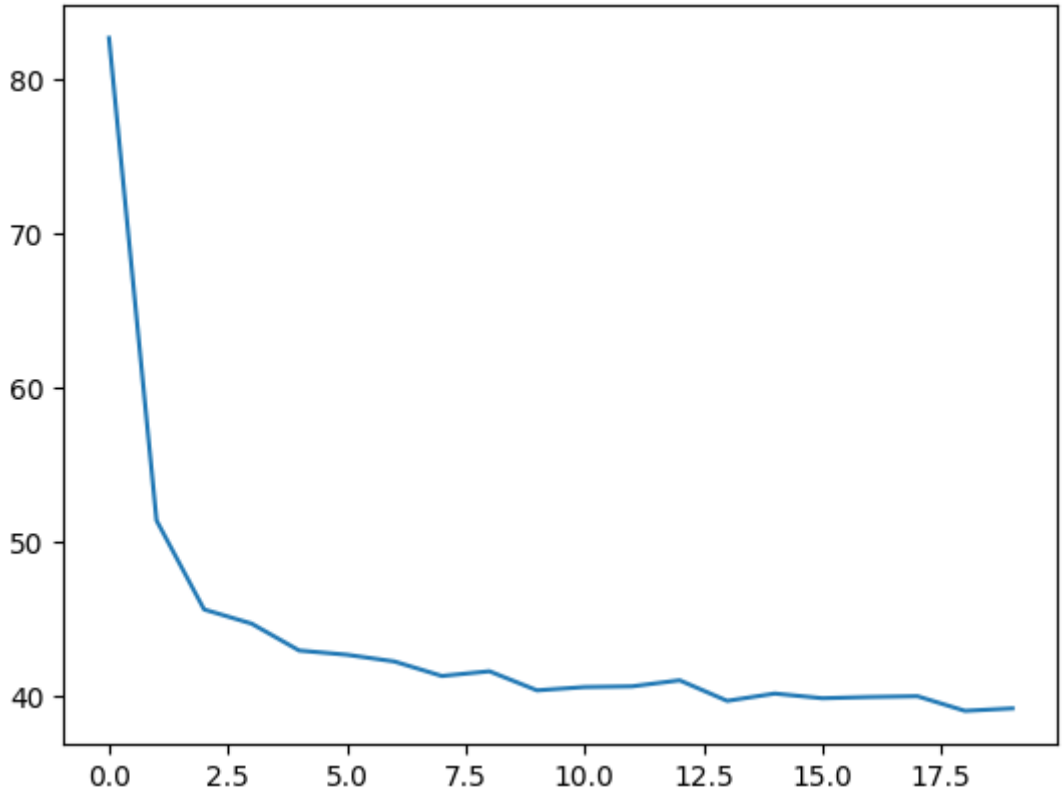
To draw the loss curve, I simply use the plot function and the existing fields to draw the image.

- Show a denoising progress image.

Denoising progess image:

- Plot your training loss curve.

Training loss curve:



Part 1-3: Evaluation Baseline (20%)

- Include a screenshot of your final **FID score** and **AFD rate**.

```
FID score: 88.09
AFD rate:  85.40%
Final score: 15/20
```

Result & Discussion (20%)

- Discuss any observed issues (e.g., over-smoothing, artifacts).

I made an interesting observation. Initially I wanted to train faster, so I used a portion of the tr aining data. I noticed that when I trained with a small amount of data, I got better results. The resulting images were closer to the faces. When I used all the data, most of the images were di storted.

However, when using the same amount of data, the impact of modifying the parameters seems to be limited. At least the difference is not easily noticeable to the human eye.

- Explain which training strategies you used (e.g., data augmentation, deeper model, beta sc hedule changes, etc.).

I found that the original model didn’ t train well. When using the preset parameters, the traini ng results gradually turned into black and white blocks in the first few steps of denoise. I learn ed that this might be because the model size was not enough to learn faces, so I increased the n umber of channels to 64.

In addition, I also found that using cosine function as a beta schedule would have better result s.

In terms of learning rate, a lower rate with a longer training time would have better results. Thi s is also a direction for improvement.

One change I'm not sure if it made a difference is that when I was initially debugging the mod el, I set clip_x_start to True in the model_predictions() function. I didn't change this back whe n I was sure the model was working.

- Provide experimental results (e.g., evaluation score, visual comparisons) to demonstrate th e effectiveness of the modifications.

I don't have a complete record of the denoise scores or situations under different parameters. H owever, I can provide some records from the experiment.

After 10000 training steps, channel size 32, learning rate 1e-4 and ema 0.98, a set of denoise i mages are obtained as follows. Although a little blurry, you can see that the results of these pic tures are not very ideal. The face shape and facial features are a little distorted. This is why I d id not upload this set of pictures to test the results.

Part 2: Optical Illusion Generation (35%)

Part 2-1: Prompt Design (5%)

- List your **two text prompts**, explain their differences, and justify your design.

prompt1: A red fox emerging from swirling flames, with fiery orange and gold colors, mystica l atmosphere.
promtp2: A castle tower rising from swirling flames when viewed upside down, with fiery ora nge and gold colors, mystical atmosphere

The difference between the two prompts is mainly in the perspective and the object. Since I wa nt to be more inverted pictures, I added this keyword to prompt. In addition, different objects a re also used as the protagonists, one is the biological fox and the other is a non-biological buil ding.

- Explain your creative thinking and how the prompts support the optical illusion task.

I use my creativity to the objects as the protagonist. Through unified style (mystery) and backg round (flame), the model can use these elements to make the content of the picture more consi stent, but retain the characteristics of two objects.

Part 2-2: Viewing Transformation (5%)

- Provide screenshots of your code and describe your implementation of IdentityView and R otate180View

The program in the 2-2 part is quite simple, and the job itself is to return a picture itself. In the part that needs to be rotated, the work is only done through built-in functions, without unneces sary content.

```
##############################
# TODO2-2: Viewing Transformation
# Begin your code
class IdentityView:
    def __init__(self):
        pass
    def view(self, im):
        return im
    def inverse_view(self, noise):
        return noise

class Rotate180View:
    def __init__(self):
        pass
    def view(self, im):
        return TF.rotate(im, 180)
    def inverse_view(self, noise):
        return TF.rotate(noise, 180)

views = [IdentityView(), Rotate180View()]

# End your code
##############################
```

Part 2-3: Denoising Operation (10%)

- Provide screenshots of your code and explain how you modified the diffusion process to s upport multi-view denoising.

Part 2-3 has too much program content, so it is divided into multiple screenshots.

The original program only used the first prompt and a single view and performed denoising. T o meet the requirements of the assignment, I modified it to use all prompts and use a loop to pr ocess each view.

In addition, because multiple images need to be averaged, the view of the images must be unifi ed. Finally, the average is calculated.

In the program, we must consider the differences in stage 1 and 2. This will affect the type and size of the variables passed in. It is necessary to ensure that the program uses consistent tensor variables. Sometimes dynamic adjustments are also required.

In the previous part of the program, you can see that I designed the same weights at different s tages. Originally, because the scores of the pictures I generated at the beginning were not idea l, I planned to adjust the content of the pictures through weighting. Later, I found that this appr oach would have the opposite effect.

```
##############################
# TODO2-3: Denoising Operation
# The following code only computes and applies the noise
# from the first prompt and original view. Please revise
# the denoising process according to spec.

# Begin your code

@torch.no_grad()
def denoising_loop(model, noisy_images, prompt_embeds, views,
                   timesteps, guidance_scale, generator, noise_level=None, upscaled=None):

    num_prompts = prompt_embeds.shape[0]
    original_noise_level = noise_level
    total_steps = len(timesteps)

    for i, t in enumerate(tqdm(timesteps)):

        progress = i / total_steps
        if progress < 0.3:
            view_weights = [0.5, 0.5]
        elif progress < 0.7:
            view_weights = [0.5, 0.5]
        else:
            view_weights = [0.5, 0.5]

        viewed_images =[]
        for view in views:
            viewed_images.append(view.view(noisy_images))
        viewed_images = torch.cat(viewed_images, dim=0)
```

```
        # If upscaled is provided (stage 2), concatenate with noisy_images
        if upscaled is not None:
            viewed_upscaled = []
            for view in views:
                viewed_upscaled.append(view.view(upscaled))
            viewed_upscaled = torch.cat(viewed_upscaled, dim=0)
            model_input = torch.cat([viewed_images, viewed_upscaled], dim=1)
        else:
            model_input = viewed_images

        # Duplicate inputs for CFG
        # Model input is: [ neg_0, neg_1, ..., pos_0, pos_1, ... ]
        model_input = torch.cat([model_input] * 2)
        model_input = model.scheduler.scale_model_input(model_input, t)

        current_noise_level = None
        if original_noise_level is not None: # stage 2
            noise_value = original_noise_level[0] if original_noise_level.numel() > 0 else original_noise_level
            current_noise_level = noise_value.unsqueeze(0).repeat(model_input.shape[0])


        # Predict noise estimate
        noise_pred = model.unet(
            model_input, t, encoder_hidden_states=prompt_embeds,
            class_labels=current_noise_level, cross_attention_kwargs=None, return_dict=False
        )[0]

        # Extract uncond (neg) and cond noise estimates
        noise_pred_uncond, noise_pred_text = noise_pred.chunk(2)
```

```
        # Split into noise estimate and variance estimates
        # Split predicted noise and predicted variances
        splited_size = model_input.shape[1] // (2 if upscaled is not None else 1)
        noise_pred_uncond, _ = noise_pred_uncond.split(splited_size, dim=1)
        noise_pred_text, predicted_variance = noise_pred_text.split(splited_size, dim=1)

        # Apply CFG only to noise prediction
        noise_pred = noise_pred_uncond + guidance_scale * (noise_pred_text - noise_pred_uncond)
        # Split noise predictions for each view
        noise_preds_per_view = noise_pred.chunk(num_prompts, dim=0)
        var_preds_per_view = predicted_variance.chunk(num_prompts, dim=0)

        # Apply inverse view transformations to each noise prediction
        inverse_viewed_noises = []
        inverse_viewed_vars = []
        for noise_pred_view, var_pred_view, view in zip(noise_preds_per_view, var_preds_per_view, views):
            inverse_viewed_noises.append(view.inverse_view(noise_pred_view))
            inverse_viewed_vars.append(view.inverse_view(var_pred_view))

        weighted_noise = sum(w * noise for w, noise in zip(view_weights, inverse_viewed_noises))
        weighted_var = sum(w * var for w, var in zip(view_weights, inverse_viewed_vars))


        # Combine averaged noise with averaged predicted variance
        combined_prediction = torch.cat([weighted_noise, weighted_var], dim=1)

        # Compute the previous noisy sample x_t -> x_t-1
        noisy_images = model.scheduler.step(combined_prediction, t, noisy_images, generator=generator, return_dict=False)[0]

    return noisy_images

# End your code
############################
```

Part 2-4: Evaluation Baseline (5%)

- Show the optical illusion result and the CLIP score.

- result:

  The result pictures are screenshots from kaggle notebook website.

- CLIP score:

```
Prompt: A red fox emerging from swirling flames, with fiery orange and gold colors, mystical a
tmosphere
CLIP Score: 0.3683

Prompt: A castle tower rising from swirling flames when viewed upside down, with fiery orange
and gold colors, mystical atmosphere
CLIP Score: 0.3519
```

Result & Discussion(10%)

- Discuss how prompt design strategies affect the result.

The prompt will affect the final score of the image. If the main characters of the two prompts a re too different (for example, in terms of color or brightness), the image will be biased towards one of them, and the score of the other one will be lower.

To solve this problem, you can give the prompt from the part other than the main character. Fo r example, standardize the same background, color, style, etc.

- Include failure cases: discuss examples where the visual anagram failed to represent the int ended word or concept, and analyze why the failure occurred.

I have a failed example of the prompt I tested for the first time. The main character is the sam e, but I did not set the same background or style, so the result was not ideal. The score of the n egative prompt did not reach the baseline.

```
Prompt: A red fox with piercing amber eyes in a snowy forest
CLIP Score: 0.3440

Prompt: An ancient castle tower with glowing windows silhouetted against a dark sky
CLIP Score: 0.2553
```

---

2. Results link

Please provide the **download link** to your **trained model checkpoints** and **image generation results** f rom Part 1. You must upload the files to **Google Drive** only, and ensure that the download link is **p ublicly accessible.**

If the link is unavailable or restricted at the time of grading, **no excuses will be accepted**, and you may receive no points for this part.

link: https://drive.google.com/drive/folders/1aynk1dY8FHltdliOHF-yxZZbVqJi4x0F?usp=sharing

```
google drive
├ generated
|  ├ 1.jpg
|  ├ 2.jpg
|  ├ …
|  └ 1000.jpg
```


model.pt