Spring 2025

Introduction to Artificial Intelligence

Homework 5: Generative AI

1. Implementation & Results

Part 1: Anime Face Generation(65%)

Part 1-1: Denoising Process (15%).

1. **init()**：Initialize the diffusion chain:

   - betas control the strength of noise added at each diffusion step, which are generated by a schedule (linear / cosine). The schedule determines how noise strength increase s step by step.
   - Alphas = 1 − betas represents the proportion of the original signal
   - Alphas_bar is the cumulative product of alpha, indicates how much of the original i mage is preserved after t steps.

```
############################
# TODO1-1-1: Denoising Process - Initialization
# Begin your code

if beta_schedule == 'linear':
    beta_schedule_fn = linear_beta_schedule
    betas = beta_schedule_fn(timesteps, **schedule_fn_kwargs)
else:
    betas = cosine_beta_schedule(timesteps, **schedule_fn_kwargs)

# alphas = 1 - betas (forward process variance retention)
alphas = 1.0 - betas

# alphas_bar = cumulative product of alphas
alphas_bar = torch.cumprod(alphas, dim=0)

return betas, alphas, alphas_bar

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

2. **add_noise_forward()**：Add noise to the original image x0 to obtain the noisy image x_t at timestep t.

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

Given x0, the density of x_t is a normal distribution with mean = sqrt($\bar{\alpha}_t$)*x0 and varian ce of 1 − $\bar{\alpha}_t$. Which means：

$$xt = \sqrt{\bar{\alpha}_t} \cdot x0 + \sqrt{1-\bar{\alpha}_t} \cdot \epsilon, \epsilon \sim N(0, I)$$

I implemented this formula in this section.

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

# extract alphas_bar
alphas_bar_t = extract(self.alphas_bar, t, x_start.shape)

# forward diffusion formula
x_t = torch.sqrt(alphas_bar_t) * x_start + torch.sqrt(1.0 - alphas_bar_t) * noise

return x_t

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

3. **denoise_backward()**：Reverse the diffusion process to iteratively generate the image

(1) Formula

The model predicts a clean image pred_x_start and noise pred_noise, then we compute th e previous step image x_prev for timestep t_prev.

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

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

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

So x_prev could be obtained with this formula：

$$x_t = \sqrt{\alpha_{t-1}} \cdot x\hat{}0 + \sqrt{1-\alpha_{t-1}-\sigma t^2} \cdot \epsilon\hat{}t + \sigma t \cdot z$$

(2) Implementation

- But when t goes from 0 -> -1, means it’ s the last one, just return the predicted clea n image.

```
# boundary case for t_prev = -1 (final step)
if t_prev < 0:
    return pred_x_start
```

- Extract $\bar{\alpha}_t$ and $\bar{\alpha}_{prev}$

```
if t_prev >= 0:
    alphas_bar_prev = self.alphas_bar[t_prev]
else:
    alphas_bar_prev = torch.tensor(1.0, device=pred_x_start.device)

alphas_bar_t = self.alphas_bar[t]
```

- σ_t is determined by eta, the bigger eta is, the bigger the noise is, which would increase the diversity of sample.

```
sigma_t = eta * torch.sqrt((1 - alphas_bar_prev) / (1 - alphas_bar_t) * (1 - alphas_bar_t / alphas_bar_prev))
```

- Calculate mu_t and apply formula

```
# DDIM formula
mu_t = expand(torch.sqrt(alphas_bar_prev)) * pred_x_start + expand(torch.sqrt(1 - alphas_bar_prev - sigma_t**2)) * pred_noise
x_prev = (mu_t + expand(sigma_t) * noise)
```

Part 1-2: Result Visualization (10%).

1. Denoising progress

   First, during each denoising loop, I saved the intermediate image at fixed intervals and always included the last step. For each saved step, I stored 5 generated images (img[:5]) in the samples list.

   Next, I stacked all collected images into a tensor and applied self.unnormalize to map the pixel values back to the [0, 1] range. I also used clamp to ensure numerical stability during visualization.

   Finally, I used plt.subplots(5, num_steps+1) to create a grid of subplots, with 5 rows representing 5 images per step, and columns representing different timesteps. For each sample in samples, I plotted the 5 corresponding images vertically in a column, using imshow on each subplot axis. The top row includes a title indicating the current step number.

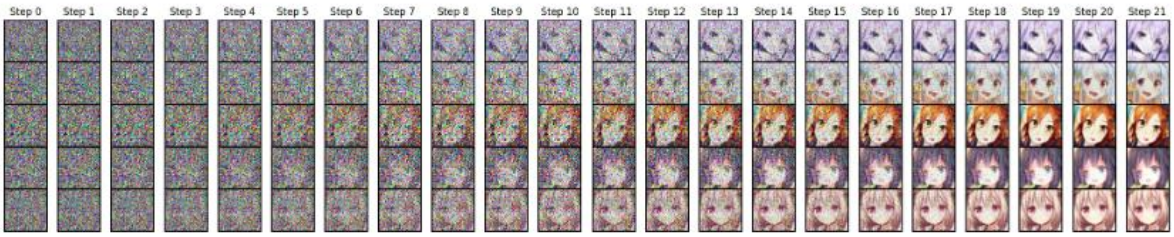
```python
##############################
# TODO1-2-1:  Result Visualization - Denoising Progress
# The following code is a denoising loop from a completed
# noise to an anime face image. You need to visualize the
# denoising progress from this code with 5 different images.

# Begin your code

step_interval = max(1, len(time_pairs) // 20)

for step_idx, (time, time_prev) in enumerate(tqdm(time_pairs, desc='sampling loop time step')):
    # 修正：確保時間步在有效範圍內
    time = max(0, min(time, total_timesteps - 1))
    time_cond = torch.full((batch,), time, device=device, dtype=torch.long)
    self_cond = x_start if self.self_condition else None
    pred_noise, x_start, *_ = self.model_predictions(img, time_cond, self_cond, clip_x_start=True, rederive_pred_noise=True)

    noise = torch.randn_like(x_start)
    img = self.denoise_backward(x_start, time, time_prev, pred_noise, noise, eta)
    imgs.append(img)

    # store samples at specific intervals
    if step_idx % step_interval == 0 or step_idx == len(time_pairs) - 1:
        samples.append(img[:5])

ret = img if not return_all_timesteps else torch.stack(imgs, dim=1)
ret = self.unnormalize(ret)


num_steps = len(samples) - 1
fig, axes = plt.subplots(5, num_steps + 1, figsize=(2*(num_steps + 1), 10), squeeze=False)

for col, sample in enumerate(samples):
    sample_unnorm = self.unnormalize(sample)  # [5, C, H, W]
    sample_unnorm = torch.clamp(sample_unnorm, 0.0, 1.0)

    for row in range(5):
        img_single = sample_unnorm[row]  # [C, H, W]
        axes[row, col].imshow(img_single.permute(1,2,0).cpu())
        axes[row, col].axis('off')
        if row == 0:
            axes[row, col].set_title(f"Step {col}")

plt.tight_layout()
plt.savefig("denoising_progress.png")
plt.show()

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

2. Loss curves

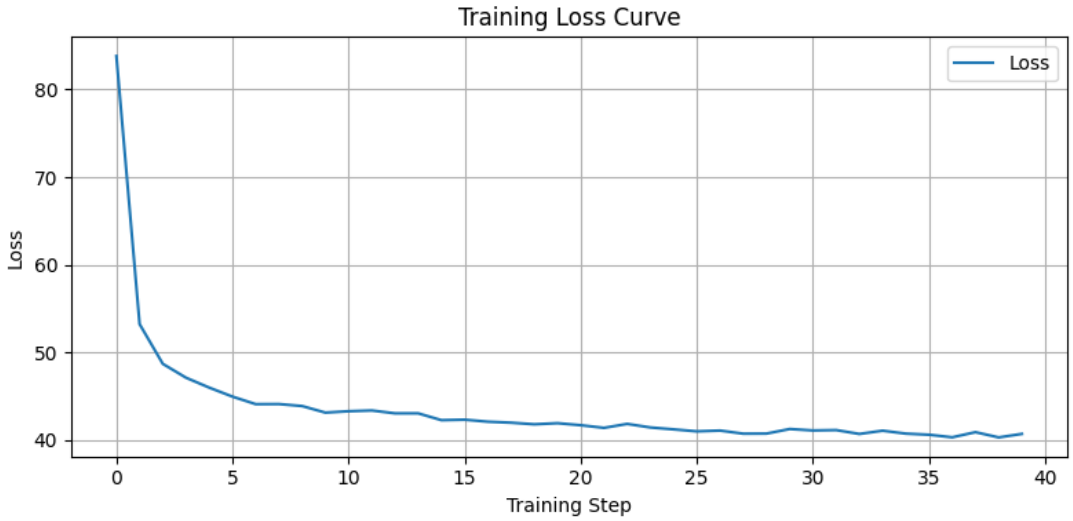
To monitor whether the model is training stably and converging, I record the loss value at e ach training step in losses, and then use matplotlib to plot the loss curve, which helps to ob serve if the model is converging properly, and whether the learning rate needs to be adjuste d.

```python
##############################
# TODO1-2-2:  Result Visualization - Loss curve
# You can use self.losses directly to plot the loss curve.
# Begin your code

losses = self.losses
if isinstance(losses, torch.Tensor):
    losses = losses.cpu().numpy()
else:
    losses = [float(l) for l in losses]

plt.figure(figsize=(8, 4))
plt.plot(losses, label='Loss')
plt.xlabel('Training Step')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

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

- Show a denoising progress image.



- Plot your training loss curve.



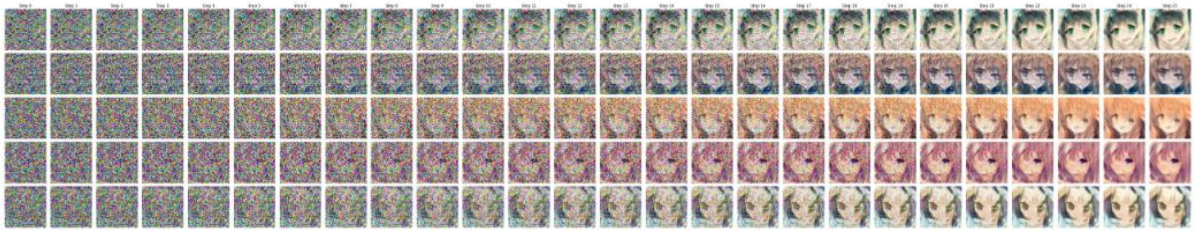
Part 1-3: Evaluation Baseline (20%)

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

```
FID score: 75.07
AFD rate:  92.70%
Final score: 18/20
```

Result & Discussion (20%)

1. Observation



Above is the output from the original model without any adjustment. As can be compared in th e denoising progress image part (progress image from last part), although the generated images has already looked quite natural and can be easily recognized as anime faces, there are still so me noticeable artifacts in the results. Such as these 2 cases below, certain facial structures look unnaturally collapsed or warped, and in some samples the eyes are misaligned or exhibit abnor mal shapes.

2. Training strategies

To improve model performance and visual quality, I used the following training strategies:

(1) Data Augmentation

I applied several data augmentation techniques to improve the model’ s robustness and g eneralization:

- Color Jitter (brightness, contrast, saturation)
- Random Horizontal Flip

(2) Cosine Beta Schedule

a. Why change beta schedule

- The Linear beta schedule increases beta linearly, which causes noise strength to rise too sharply in the later steps, often making the generated images unstable an d degraded.
- The Cosine Beta Schedule smooths the noise schedule, keeping more structure o f the image throughout the process and making sampling more stable.

b. Cosine Beta Schedule

- For early steps： slower noise increase, better image retention
- For middle/late steps：smoother noise transition, more stable sampling, better v isual quality
- Implementation

I implemented the Cosine Beta Schedule based on the formula：

$$\alpha_t^{cumprod} = \frac{cos^2\left(\frac{\frac{t}{T}+s}{1+s} * \frac{\pi}{2}\right)}{cos^2\left(\frac{s}{1+s} * \frac{\pi}{2}\right)}$$

$$\beta t = 1 - \frac{\alpha_{t+1}^{cumprod}}{\alpha_t^{cumprod}}$$

```python
def cosine_beta_schedule(timesteps: int, s: float = 0.008):
    steps = timesteps + 1
    x = torch.linspace(0, timesteps, steps, dtype=torch.float64)
    alphas_cumprod = torch.cos(((x / timesteps + s) / (1 + s)) * math.pi / 2) ** 2
    alphas_cumprod = alphas_cumprod / alphas_cumprod[0]
    betas = 1. - (alphas_cumprod[1:] / alphas_cumprod[:-1])
    return betas.clamp(min=0.0, max=0.999)
```

- Add register buffer for alphas

  Since the resulting $\bar{\alpha}_t$ and β_t values are computed using floating point approxi mations. If we were to recompute these values multiple times during training an d sampling, the results might not be exactly consistent due to floating-point prec ision errors

```python
register_buffer('betas', betas)
register_buffer('alphas', alphas) # add register !!!
register_buffer('alphas_bar', alphas_bar)
register_buffer('alphas_bar_prev', alphas_bar_prev)
```

(3) Hyperparameter

a. Increase model capacity (channels, dim_mults)

I increased the number of channels and the U-Net depth so that the model can better le arn fine details in the face dataset.

b. Increase diffusion steps

Using timesteps=1000 allows each denoising step to be more refined, helping to re duce oversmoothing and preserve more details.

c. Lower learning rate

I reduced it to 2e-4, making the training more stable, which is more suitable when trai ning a larger model for a longer period.

d. Increase train_num_steps

Training for 40000 steps allows the larger model to fully converge. Additionally, cosin e schedule typically requires longer training to show its advantages.

e. Set grad_steps=2

Using gradient accumulation helps stabilize batch normalization and is equivalent to tr aining with a larger batch size, which leads to more stable learning

```
########## Hyper-Parameters ##########
seed = 114514
path = '/kaggle/input/diffusion/faces/faces'
IMG_SIZE = 64
batch_size = 32
train_num_steps = 40000
lr = 2e-4
grad_steps = 2
ema_decay = 0.995

channels = 64
dim_mults = (1, 2, 4, 8)

timesteps = 1000
sampling_timesteps = 500
beta_schedule = 'cosine'
schedule_fn_kwargs = {'s': 0.008},

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

3. Experiments result
   a. Take the original design as the baseline

```
FID score: 144.13
AFD rate:  65.80%
Final score: 0/20
```

   b. Original design with Data augmentation

```
FID score: 143.52
AFD rate:  71.70%
Final score: 0/20
```

   Both FID score and AFD rate has improved, especially the AFD rate.

   c. Original design with Cosine Beta Schedule

```
FID score: 142.65
AFD rate:  59.90%
Final score: 0/20
```

   Both FID score and AFD rate did not improve as expected.

   I think the reason is that the cosine schedule is designed to gradually add noise at the begin ning of training, unlike the linear schedule that distributes noise more uniformly. As a resu lt, models using Cosine Beta Schedule generally require a longer training period to fully le verage its advantages in smoother denoising and better fine-detail preservation. Since I onl y trained for 10k steps, the model likely did not converge well yet under the cosine schedul e. According to prior works (Nichol & Dhariwal 2021), Cosine Beta Schedule tends to out perform linear when trained longer. Therefore, I believe that with longer training, the benef its of the Cosine schedule would become more apparent.

Part 2: Optical Illusion Generation (35%)

Part 2-1: Prompt Design (5%)

- List your **two text prompts**, explain their differences, and justify your design.
  - prompt_1 = "A painting of spaceship on top of the whole earth."

- prompt_2 = "A painting of a hot air balloon."

Apparently, these 2 prompt describe 2 different thing. Prompt1 is describing a spaceship fl oating above the earth；Prompt2 is describing a hot air balloon.

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

I came up with this idea by thinking about how to design a scene that would look complete ly different when flipped upside down, and this concept came out.

I utilized the shape similarity between the Earth and a hot air balloon, which means the Ear th could be mapped to the balloon itself, while the spaceship could correspond to the baske t of the hot air balloon after rotation.

To ensure that the two views appear more coherent and compatible, I adjusted the style by adding the keyword “painting”, so that the color, level of detail, and overall visual consi stency would be aligned between both prompts.

Part 2-2: Viewing Transformation (5%)

1. IdentityView：Keeps the image unchanged, which is used to represent the normal upri ght orientation in the multi-view pipeline.
   - The view() method directly returns the original image.
   - The inverse_view() method returns the original noise.
2. Rotate180View：Rotates the image by 180 degrees, which is used to represent a upsid e-down orientation in the multi-view pipeline.
   - The view() method:
     - If the input is a PyTorch Tensor, it performs a flip along both height and w idth dimensions
     - If the input is a PIL image, it calls .rotate(180) to perform the rotation
   - The inverse_view() method reverses the transformation:
     - For tensors, it applies the same flip again.
     - For PIL images, it rotates by 180° again to restore the original orientation.

```python
import torchvision.transforms as T

############################
# TODO2-2: Viewing Transformation
# Begin your code

# views = [IdentityView(), Rotate180View()]
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):
        if isinstance(im, torch.Tensor):
            return im.flip(-1, -2)
        else:
            return im.rotate(180)
    def inverse_view(self, noise):
        if isinstance(noise, torch.Tensor):
            return noise.flip(-1, -2)
        else:
            return noise.rotate(180)

views = [IdentityView(), Rotate180View()]

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

Part 2-3: Denoising Operation (10%)

1. Pipeline design

In the baseline implementation, the denoising loop only utilized the first prompt and the first vi ew, which could not support multi-view illusions.

To address this, I modified the denoising loop to simultaneously process multiple views. For ea ch view, I applied the corresponding positive and negative prompt embeddings, transformed th e latent image using view.view(), predicted the noise, and then reverted the predicted noise bac k to the original latent space using inverse_view().

Finally, I fused the noise predictions from all views and used the combined result to update the latent image.2. Implementation modified

1. Multi-view loop

   I modified the denoising loop to iterate over all views defined in the views list, which in t his assignment contains two views: IdentityView and Rotate180View. Each view transforms the latent image using view.view(noisy_images), and is paired with its corresponding positive/negative prompt embeddings.
2. CFG

I use classifier-free guidance for each view to strengthen the conditioning on the prompts.

This enhances the clarity of view-specific features, which is crucial for achieving effectiv e multi-view illusions.

3. inverse_view

After predicting the noise for each view, I used view.inverse_view() to map the predicted n oise back to the original latent space.

This ensures that when the latent image is updated, all noise predictions are aligned in the same coordinate space, avoiding artifacts caused by inconsistent

4. Channel interleaving

In the beginning, I observed that my denoising operation was simply overlapping two im ages generated from two prompts under two different views, resulting in a blended and a mbiguous output without a clear illusion effect.

To address this issue and to further enhance the separation of view-specific representatio ns within the latent space, I applied channel interleaving when exactly two views are use d.

For each denoising timestep, I combined the noise predictions from the two views such t hat:

- Even-numbered channels of the latent space are filled with noise predicted from vi ew 0.
- Odd-numbered channels of the latent space are filled with noise predicted from vie w 1.

And the variance predictions from both views are simply averaged to maintain sampling stability.

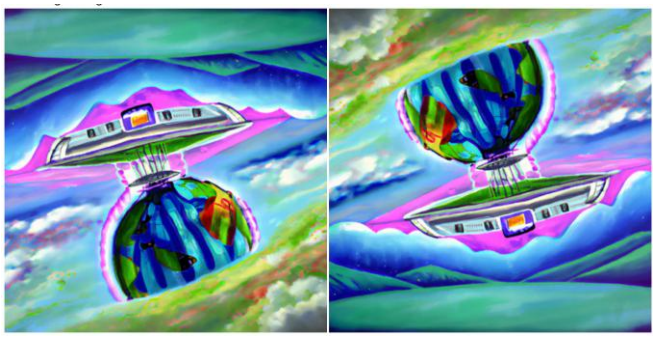
This design allows the latent space to encode different view-dependent features into dist inct channel segments.

When the image is decoded, the decoder, which typically relies on local and spatial patte rns, will naturally emphasize different channel segments depending on the viewing trans formation applied.

```
num_views = len(views)
assert prompt_embeds.shape[0] == 2 * num_views

# neg / pos
neg_embeds, pos_embeds = prompt_embeds.split(num_views)

if noise_level is not None:
    noise_level = torch.cat([noise_level] * 2)

for t in tqdm(timesteps):
    view_preds = []

    for v_idx, view in enumerate(views):
        cur_prompt = torch.stack([
            neg_embeds[v_idx],
            pos_embeds[v_idx]
        ])

        if upscaled is not None:
            img_in = torch.cat([
                view.view(noisy_images),
                view.view(upscaled)
            ], dim=1)
        else:
            img_in = view.view(noisy_images)


        img_in = torch.cat([img_in, img_in], dim=0)
        img_in = model.scheduler.scale_model_input(img_in, t)


        noise_pred = model.unet(
            img_in, t,
            encoder_hidden_states=cur_prompt,
            class_labels=noise_level,
            cross_attention_kwargs=None,
            return_dict=False
        )[0]

        noise_uncond, noise_cond = noise_pred.chunk(2, dim=0)
        C = img_in.shape[1] // (2 if upscaled is not None else 1)
        noise_u, _      = noise_uncond.split(C, dim=1)
        noise_c, var_c  = noise_cond.split(C, dim=1)
        noise_out = noise_u + guidance_scale * (noise_c - noise_u)
        pred = torch.cat([noise_out, var_c], dim=1)
        view_preds.append(view.inverse_view(pred))

    if num_views == 2:
        p0, p1 = view_preds
        C_total = p0.shape[1]
        noise0, var0 = p0[:, :C], p0[:, C:]
        noise1, var1 = p1[:, :C], p1[:, C:]

        interleaved = torch.zeros_like(noise0)
        interleaved[:, 0::2, ...] = noise0[:, 0::2, ...]
        interleaved[:, 1::2, ...] = noise1[:, 1::2, ...]

        var_avg = (var0 + var1) * 0.5
        noise_pred_avg = torch.cat([interleaved, var_avg], dim=1)

    else:
        noise_pred_avg = torch.mean(torch.stack(view_preds, dim=0), dim=0)

    noisy_images = model.scheduler.step(
        noise_pred_avg, t, noisy_images,
        generator=generator,
        return_dict=False
    )[0]
```

Part 2-4: Evaluation Baseline (5%)

- Show the optical illusion result and the CLIP score.
- Illusion result



- CLIP score

```
Prompt: A painting of spaceship on top of the whole earth.
CLIP Score: 0.3072

Prompt: A painting of a hot air balloon.
CLIP Score: 0.3235
```

Result & Discussion(10%)

1. Effect of Prompt Design Strategies

In designing prompts for the optical illusion task, I primarily focused on selecting object pa irs with similar geometric structures.

Specifically, I utilized the shape similarity between the Earth and a hot air balloon. The Ea rth can visually correspond to the balloon；The spaceship positioned above the Earth can map to the basket of the hot air balloon when rotated.

To further enhance visual coherence, I applied a unified artistic style by including the keyw ord “painting” in both prompts. This helps align the global style, making it easier for th e model to focus on learning the transformation between objects rather than reconciling sty listic differences.

Through iterative experimentation, I also realized that the background consistency is a cruc ial factor. If the background is consistent between views, the model can focus more effecti vely on encoding the transformation of the main objects.

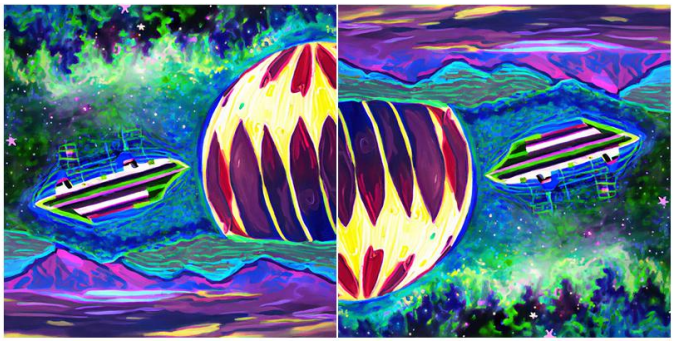
2. Failure Case

Based on this insight, I attempted to explicitly control the background. I modified my prompts to include background descriptions:

- Prompt 1: "A painting of spaceship on top of the natural color earth with outer space background."

- Prompt 2: "A painting of a hot air balloon with night background."

However, this attempt failed to produce the intended illusion (as can been in the photo belo w). I think the reason is that although I mentioned the similar background, it’ s still not th at clear to generate similar style for night and outer space, and the position between spaces hip and the earth is wrong so that it doesn’ t look like a hot air balloon from upside-down view. Furthermore, the color of the earth itself is weird so that there’ s a large difference b etween it and real earth.



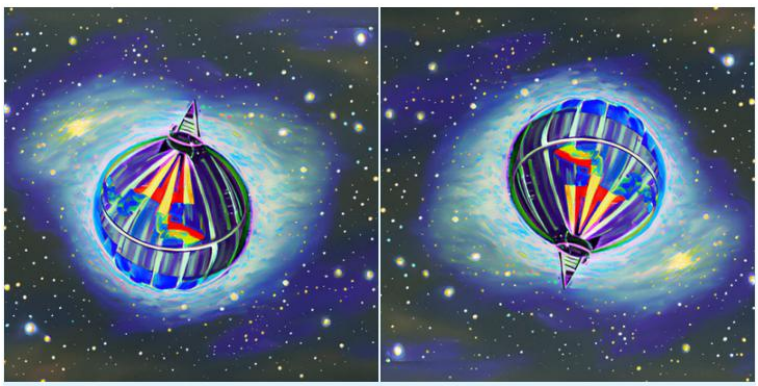
3. Successful Case

To improve the result, I refined the prompts further by enforcing a clearly consistent backg round and explicit object positioning:

- Prompt 1: "A painting of a spaceship flying above the Earth, with a dark starry sky ba ckground. The Earth is fully visible and centered. The spaceship is positioned above t he Earth."

- Prompt 2: "A painting of a hot air balloon floating in the sky, with a dark starry sky b ackground. The balloon is fully visible and centered. A small basket is attached below the balloon."

This more precisely prompt significantly improved the outcome：



Also improved the CLIP score compared to the original one：

```
Prompt: A painting of
CLIP Score: 0.3309

Prompt: A painting of
CLIP Score: 0.3904
```

4. Conclusion

Overall, I think shape similarity, consistent style, background alignment, and clear object positioning can significantly improve the quality of multi-view optical illusions.

---

2. Results link

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

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

https://drive.google.com/drive/folders/1SYfhBcxkhgA6R021e8cxgp6xfyDBr3-K?usp=sharing