Lab5:
MaskGIT for Image Inpainting

312551133

鄧長軒

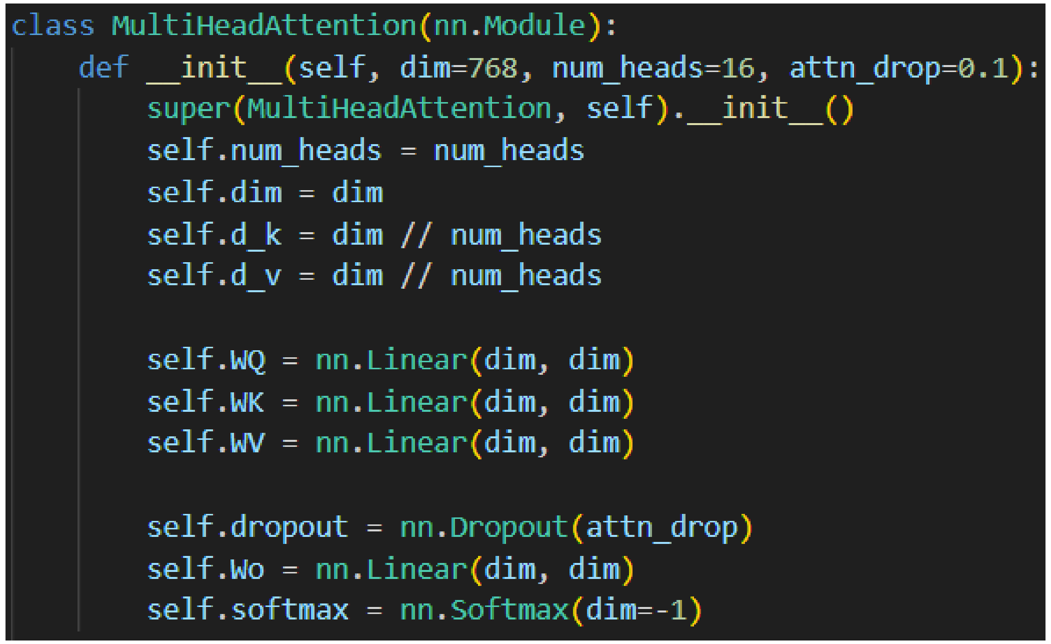
1. Introduction

在Lab5中，我實作了MaskGIT中的Multi-Head Attention，學習訓練 bidirectional transformer，與iterative image inpainting。

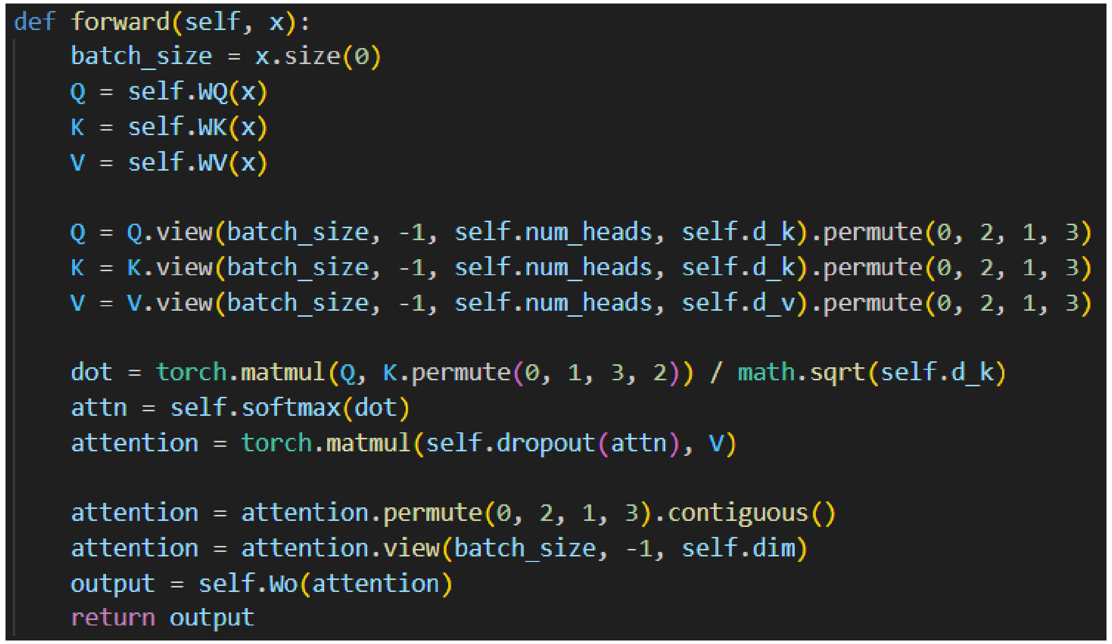
透過Lab5，我了解MaskGIT的流程與詳細原理，把圖片透過 VQGAN encode成latent，隨機把latent的一部份遮住，用 bidirectional transformer預測遮住的部分，再decode回圖片，藉此來完成圖片修復。最後得到FID score = 36.95。

2. Implementation Details

A. The details of Multi-Head Self-Attention

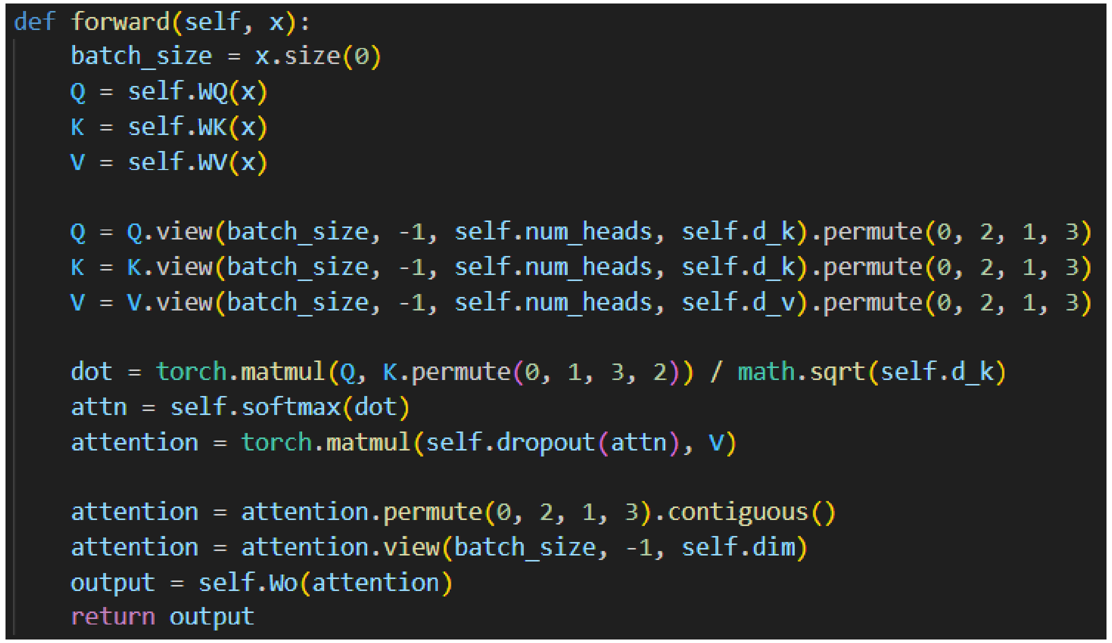


```python
class MultiHeadAttention(nn.Module):
    def __init__(self, dim=768, num_heads=16, attn_drop=0.1):
        super(MultiHeadAttention, self).__init__()
        self.num_heads = num_heads
        self.dim = dim
        self.d_k = dim // num_heads
        self.d_v = dim // num_heads

        self.WQ = nn.Linear(dim, dim)
        self.WK = nn.Linear(dim, dim)
        self.WV = nn.Linear(dim, dim)

        self.dropout = nn.Dropout(attn_drop)
        self.Wo = nn.Linear(dim, dim)
        self.softmax = nn.Softmax(dim=-1)
```



```python
def forward(self, x):
    batch_size = x.size(0)
    Q = self.WQ(x)
    K = self.WK(x)
    V = self.WV(x)

    Q = Q.view(batch_size, -1, self.num_heads, self.d_k).permute(0, 2, 1, 3)
    K = K.view(batch_size, -1, self.num_heads, self.d_k).permute(0, 2, 1, 3)
    V = V.view(batch_size, -1, self.num_heads, self.d_v).permute(0, 2, 1, 3)

    dot = torch.matmul(Q, K.permute(0, 1, 3, 2)) / math.sqrt(self.d_k)
    attn = self.softmax(dot)
    attention = torch.matmul(self.dropout(attn), V)

    attention = attention.permute(0, 2, 1, 3).contiguous()
    attention = attention.view(batch_size, -1, self.dim)
    output = self.Wo(attention)
    return output
```

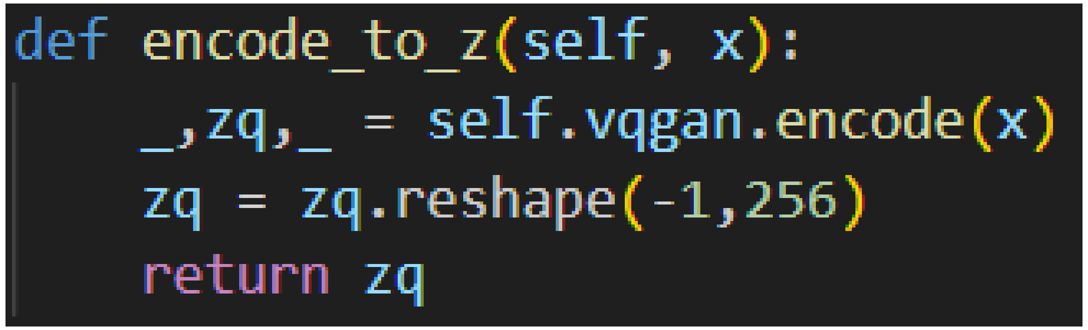
A. The details of Multi-Head Self-Attention



```python
def forward(self, x):
    batch_size = x.size(0)
    Q = self.WQ(x)
    K = self.WK(x)
    V = self.WV(x)

    Q = Q.view(batch_size, -1, self.num_heads, self.d_k).permute(0, 2, 1, 3)
    K = K.view(batch_size, -1, self.num_heads, self.d_k).permute(0, 2, 1, 3)
    V = V.view(batch_size, -1, self.num_heads, self.d_v).permute(0, 2, 1, 3)

    dot = torch.matmul(Q, K.permute(0, 1, 3, 2)) / math.sqrt(self.d_k)
    attn = self.softmax(dot)
    attention = torch.matmul(self.dropout(attn), V)

    attention = attention.permute(0, 2, 1, 3).contiguous()
    attention = attention.view(batch_size, -1, self.dim)
    output = self.Wo(attention)
    return output
```

- 首先先將x透過Linear得到QKV。
- 然後把QKV分成多頭，並把batch跟頭移到前兩個維度，以便後續計算。
- 將Q乘上K transpose，除以根號dk。
- 算softmax與dropout後，乘上V。
- 調整形狀，通過最後的Linear得到最後輸出。

B. The details of your stage2 training (MVTM, forward, loss)



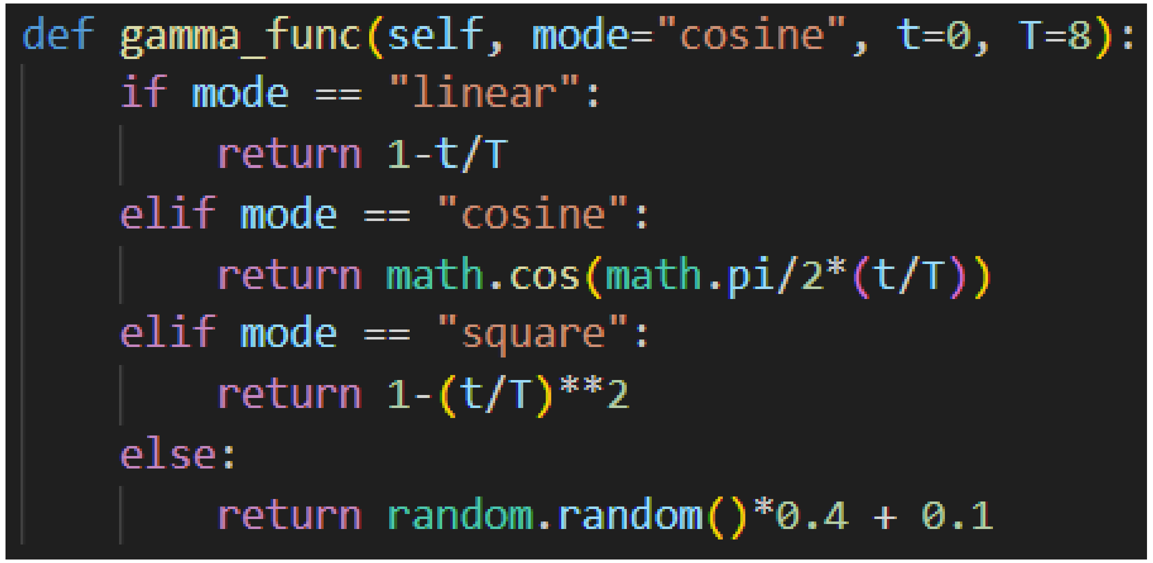
```
def encode_to_z(self, x):
    _,zq,_ = self.vqgan.encode(x)
    zq = zq.reshape(-1,256)
    return zq
```

encode to z:

input x是image，透過encode得到zq，形狀改變成

[batch size, token length]後回傳。

B. The details of your stage2 training (MVTM, forward, loss)

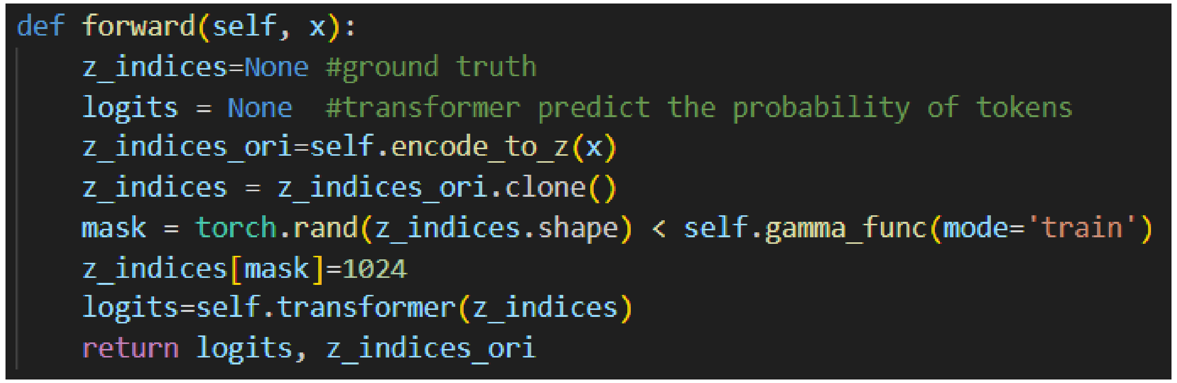


```python
def gamma_func(self, mode="cosine", t=0, T=8):
    if mode == "linear":
        return 1-t/T
    elif mode == "cosine":
        return math.cos(math.pi/2*(t/T))
    elif mode == "square":
        return 1-(t/T)**2
    else:
        return random.random()*0.4 + 0.1
```

gamma func:

train時隨機返回0.1~0.5。

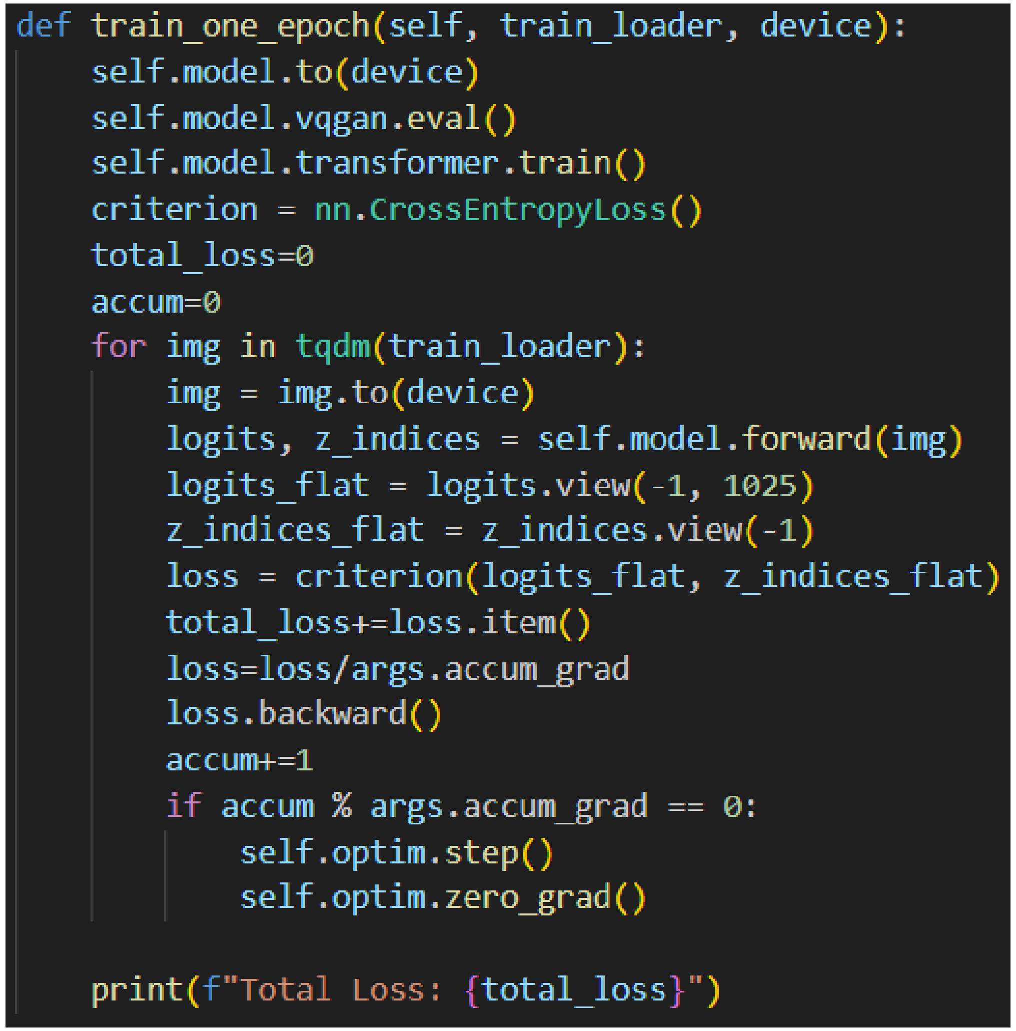
B. The details of your stage2 training (MVTM, forward, loss)



```
def forward(self, x):
    z_indices=None #ground truth
    logits = None  #transformer predict the probability of tokens
    z_indices_ori=self.encode_to_z(x)
    z_indices = z_indices_ori.clone()
    mask = torch.rand(z_indices.shape) < self.gamma_func(mode='train')
    z_indices[mask]=1024
    logits=self.transformer(z_indices)
    return logits, z_indices_ori
```

- 首先將x encode成z indices ori。
- z indices ori是之後要回傳的，所以複製一份z indices以供修改。
- 隨機產生mask，並把小於gamma的值設為1，代表被遮住。
- 把被遮住的z indices設為1024。
- 通過transformer得到logits後回傳。

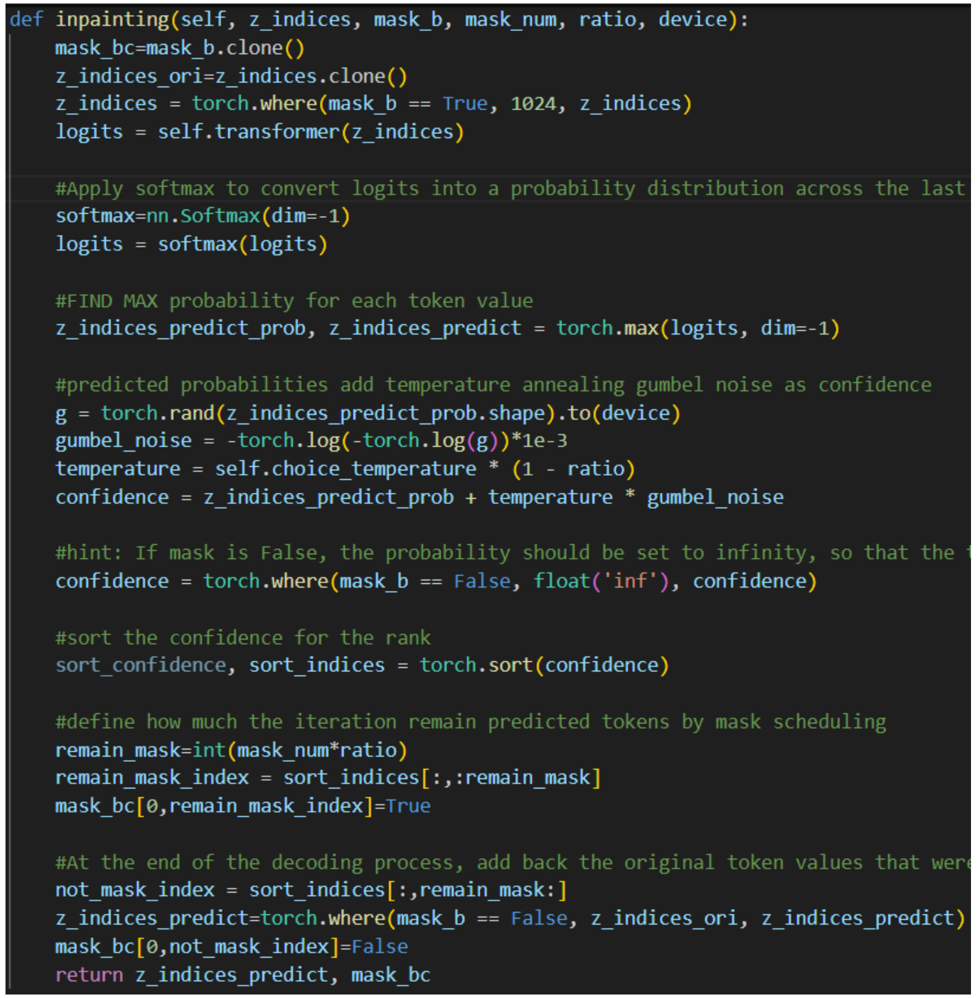
B. The details of your stage2 training (MVTM, forward, loss)



```
def train_one_epoch(self, train_loader, device):
    self.model.to(device)
    self.model.vqgan.eval()
    self.model.transformer.train()
    criterion = nn.CrossEntropyLoss()
    total_loss=0
    accum=0
    for img in tqdm(train_loader):
        img = img.to(device)
        logits, z_indices = self.model.forward(img)
        logits_flat = logits.view(-1, 1025)
        z_indices_flat = z_indices.view(-1)
        loss = criterion(logits_flat, z_indices_flat)
        total_loss+=loss.item()
        loss=loss/args.accum_grad
        loss.backward()
        accum+=1
        if accum % args.accum_grad == 0:
            self.optim.step()
            self.optim.zero_grad()

    print(f"Total Loss: {total_loss}")
```

- 讓vqgan保持eval狀態，避免更動到，只把transformer設成train狀態。
- 把logits跟z indices轉為CrossEntropyLoss可接受的形狀。
- 根據accm grad來更新權重。

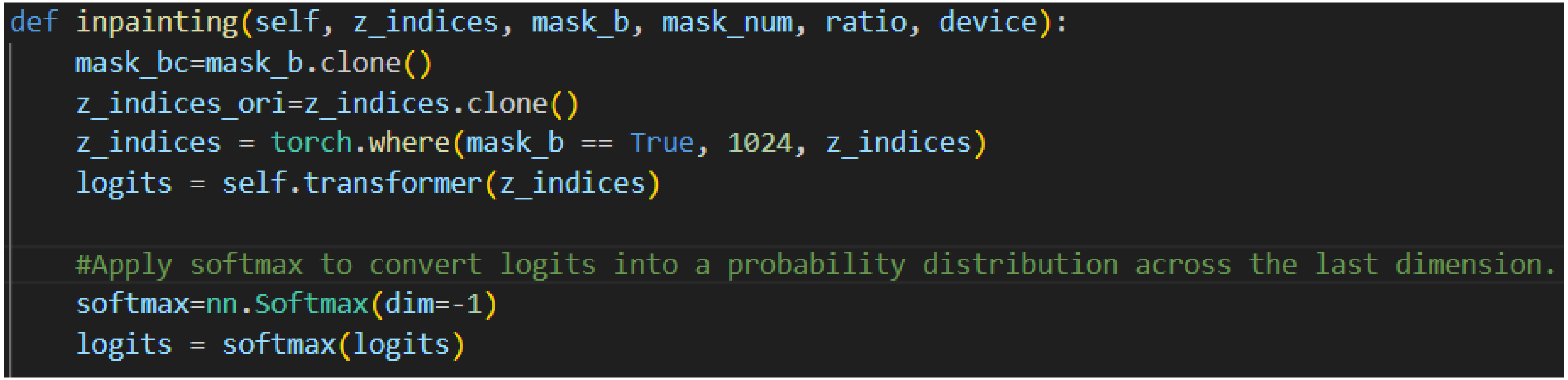
C. The details of your inference for inpainting task (iterative decoding)



```python
def inpainting(self, z_indices, mask_b, mask_num, ratio, device):
    mask_bc=mask_b.clone()
    z_indices_ori=z_indices.clone()
    z_indices = torch.where(mask_b == True, 1024, z_indices)
    logits = self.transformer(z_indices)

    #Apply softmax to convert logits into a probability distribution across the last
    softmax=nn.Softmax(dim=-1)
    logits = softmax(logits)

    #FIND MAX probability for each token value
    z_indices_predict_prob, z_indices_predict = torch.max(logits, dim=-1)

    #predicted probabilities add temperature annealing gumbel noise as confidence
    g = torch.rand(z_indices_predict_prob.shape).to(device)
    gumbel_noise = -torch.log(-torch.log(g))*1e-3
    temperature = self.choice_temperature * (1 - ratio)
    confidence = z_indices_predict_prob + temperature * gumbel_noise

    #hint: If mask is False, the probability should be set to infinity, so that the
    confidence = torch.where(mask_b == False, float('inf'), confidence)

    #sort the confidence for the rank
    sort_confidence, sort_indices = torch.sort(confidence)

    #define how much the iteration remain predicted tokens by mask scheduling
    remain_mask=int(mask_num*ratio)
    remain_mask_index = sort_indices[:,:remain_mask]
    mask_bc[0,remain_mask_index]=True

    #At the end of the decoding process, add back the original token values that were
    not_mask_index = sort_indices[:,remain_mask:]
    z_indices_predict=torch.where(mask_b == False, z_indices_ori, z_indices_predict)
    mask_bc[0,not_mask_index]=False
    return z_indices_predict, mask_bc
```

VQGAN_Transformer.py中的單次inpainting。

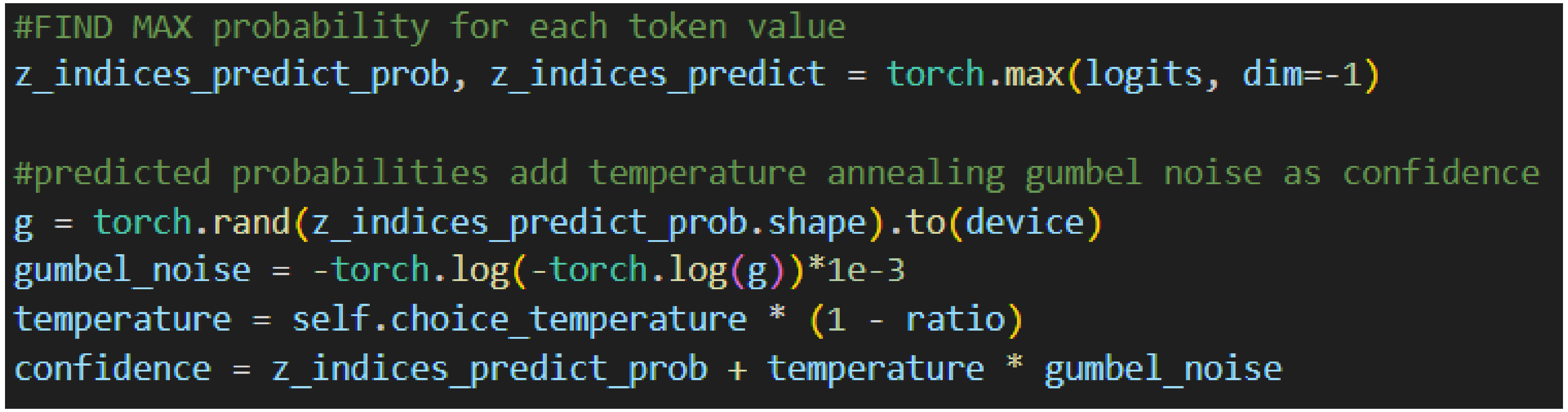
C. The details of your inference for inpainting task (iterative decoding)



```
def inpainting(self, z_indices, mask_b, mask_num, ratio, device):
    mask_bc=mask_b.clone()
    z_indices_ori=z_indices.clone()
    z_indices = torch.where(mask_b == True, 1024, z_indices)
    logits = self.transformer(z_indices)

    #Apply softmax to convert logits into a probability distribution across the last dimension.
    softmax=nn.Softmax(dim=-1)
    logits = softmax(logits)
```

- 複製一份z indices ori與mask bc，為了在最後把原本的token value加回未mask的token。
- 把z indices中，mask b為True的相對應位置設成1024。
- 通過transformer與softmax，得到logits。

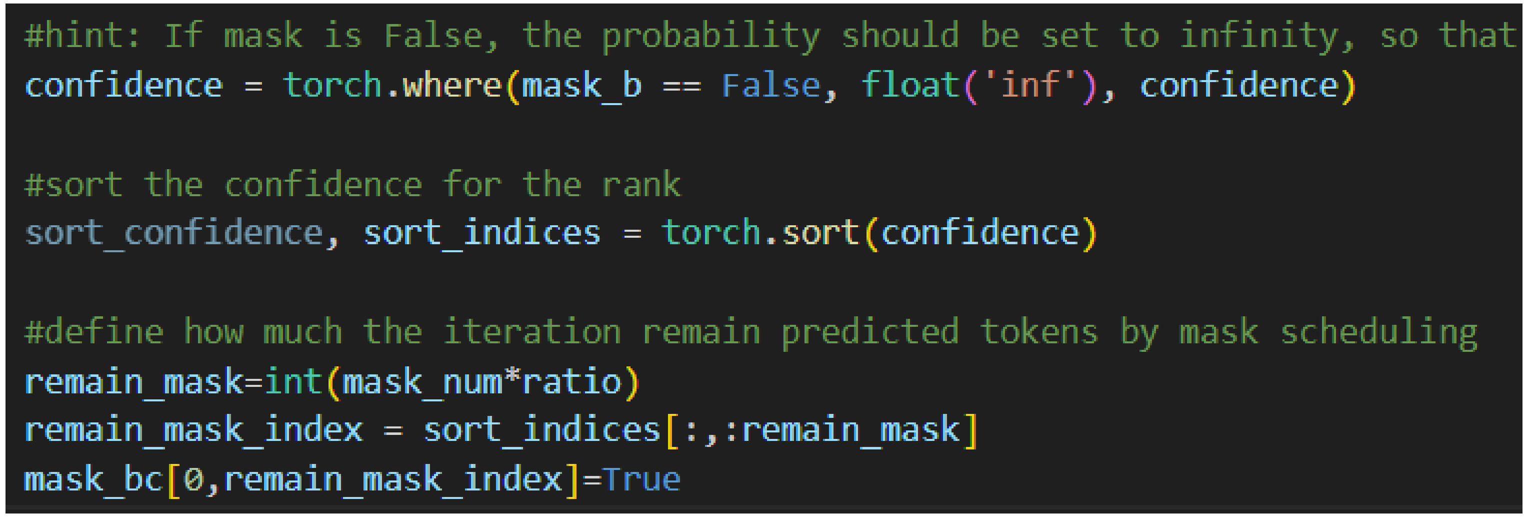
C. The details of your inference for inpainting task (iterative decoding)



```
#FIND MAX probability for each token value
z_indices_predict_prob, z_indices_predict = torch.max(logits, dim=-1)

#predicted probabilities add temperature annealing gumbel noise as confidence
g = torch.rand(z_indices_predict_prob.shape).to(device)
gumbel_noise = -torch.log(-torch.log(g))*1e-3
temperature = self.choice_temperature * (1 - ratio)
confidence = z_indices_predict_prob + temperature * gumbel_noise
```

- 把logits中最大機率的值找出來。
- 建立gumbel noise，乘上temperature，加上z indices predict prob得到confidence。
- 由於z indices predict prob大概在1e-1~1e-2之間，所以gumbel noise乘上1e-3，讓他只有些微影響。

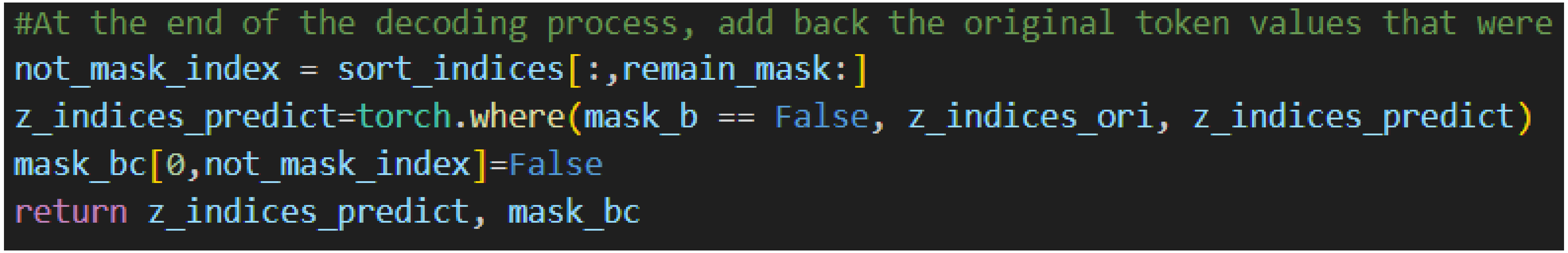
C. The details of your inference for inpainting task (iterative decoding)



```
#hint: If mask is False, the probability should be set to infinity, so that
confidence = torch.where(mask_b == False, float('inf'), confidence)

#sort the confidence for the rank
sort_confidence, sort_indices = torch.sort(confidence)

#define how much the iteration remain predicted tokens by mask scheduling
remain_mask=int(mask_num*ratio)
remain_mask_index = sort_indices[:,:remain_mask]
mask_bc[0,remain_mask_index]=True
```

- 把沒被遮住的值設為infinity，並排序。
- 算出有多少token應該要維持被遮住，並把那些token的index找出來。
- 在mask bc中，把那些index設為True。

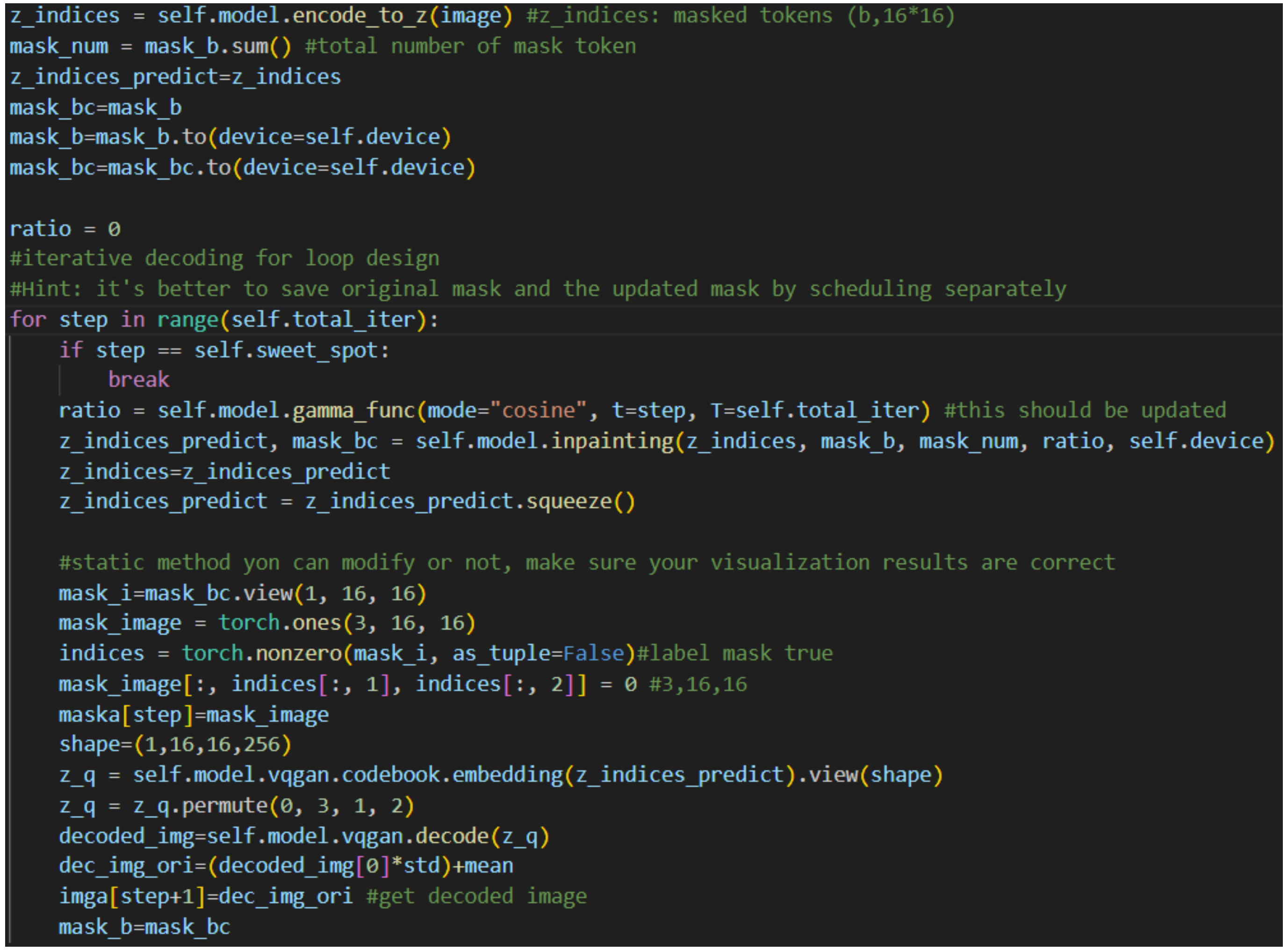
C. The details of your inference for inpainting task (iterative decoding)



```
#At the end of the decoding process, add back the original token values that were
not_mask_index = sort_indices[:,remain_mask:]
z_indices_predict=torch.where(mask_b == False, z_indices_ori, z_indices_predict)
mask_bc[0,not_mask_index]=False
return z_indices_predict, mask_bc
```

- 找出不用被遮住的index，將z indices ori的值設回去。
- 把mask bc中不用被遮住的位置設為False。
- 回傳decoding一次的z indices predict與mask bc。

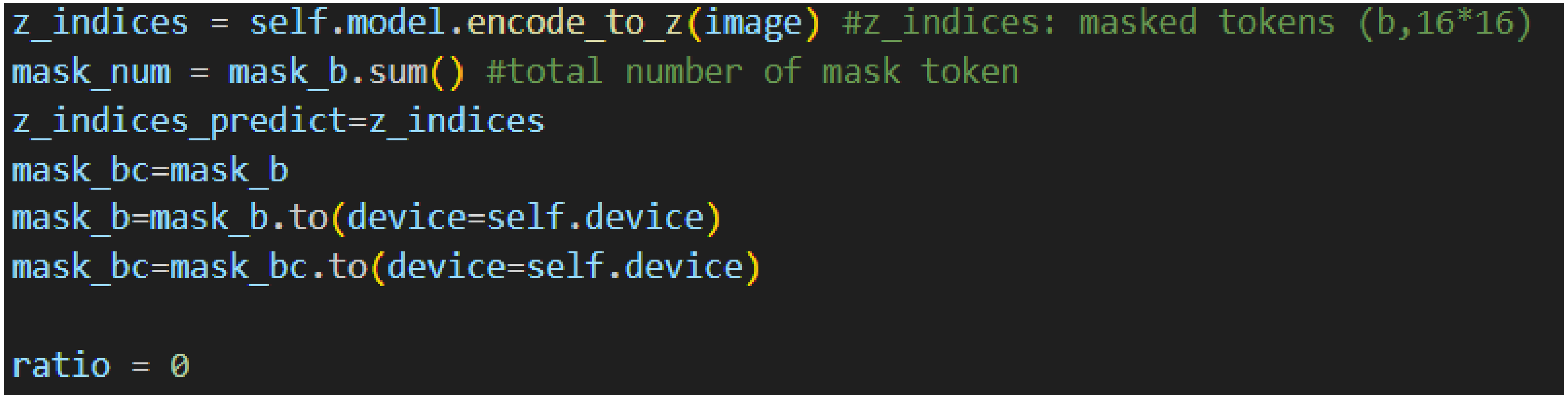
C. The details of your inference for inpainting task (iterative decoding)



```
z_indices = self.model.encode_to_z(image) #z_indices: masked tokens (b,16*16)
mask_num = mask_b.sum() #total number of mask token
z_indices_predict=z_indices
mask_bc=mask_b
mask_b=mask_b.to(device=self.device)
mask_bc=mask_bc.to(device=self.device)

ratio = 0
#iterative decoding for loop design
#Hint: it's better to save original mask and the updated mask by scheduling separately
for step in range(self.total_iter):
    if step == self.sweet_spot:
        break
    ratio = self.model.gamma_func(mode="cosine", t=step, T=self.total_iter) #this should be updated
    z_indices_predict, mask_bc = self.model.inpainting(z_indices, mask_b, mask_num, ratio, self.device)
    z_indices=z_indices_predict
    z_indices_predict = z_indices_predict.squeeze()

    #static method yon can modify or not, make sure your visualization results are correct
    mask_i=mask_bc.view(1, 16, 16)
    mask_image = torch.ones(3, 16, 16)
    indices = torch.nonzero(mask_i, as_tuple=False)#label mask true
    mask_image[:, indices[:, 1], indices[:, 2]] = 0 #3,16,16
    maska[step]=mask_image
    shape=(1,16,16,256)
    z_q = self.model.vqgan.codebook.embedding(z_indices_predict).view(shape)
    z_q = z_q.permute(0, 3, 1, 2)
    decoded_img=self.model.vqgan.decode(z_q)
    dec_img_ori=(decoded_img[0]*std)+mean
    imga[step+1]=dec_img_ori #get decoded image
    mask_b=mask_bc
```

inpainting.py中的
inpainting。

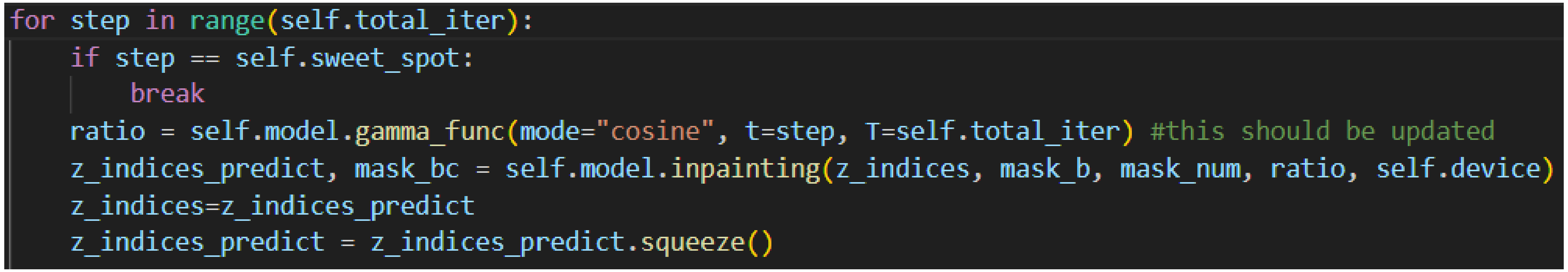
C. The details of your inference for inpainting task (iterative decoding)



```
z_indices = self.model.encode_to_z(image) #z_indices: masked tokens (b,16*16)
mask_num = mask_b.sum() #total number of mask token
z_indices_predict=z_indices
mask_bc=mask_b
mask_b=mask_b.to(device=self.device)
mask_bc=mask_bc.to(device=self.device)

ratio = 0
```

- 先把image轉成z indices。
- 由於iterative decoding是對z indices做預測，代表encode to z只要做一次，所以放在迴圈外。

C. The details of your inference for inpainting task (iterative decoding)

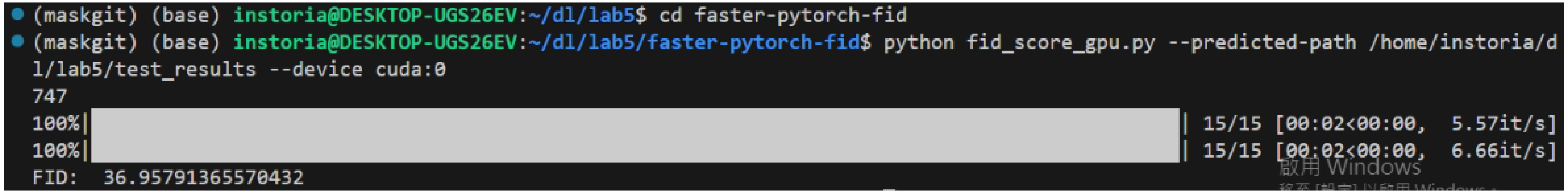


```
for step in range(self.total_iter):
    if step == self.sweet_spot:
        break
    ratio = self.model.gamma_func(mode="cosine", t=step, T=self.total_iter) #this should be updated
    z_indices_predict, mask_bc = self.model.inpainting(z_indices, mask_b, mask_num, ratio, self.device)
    z_indices=z_indices_predict
    z_indices_predict = z_indices_predict.squeeze()
```

- 根據t與T決定ratio。
- 把所有東西透過inpainting完成一次decoding，得到預測後的 z indices predict，與被遮住更多的mask bc。
- 由於z indices predict的形狀是[batch size, token length]=[1,256]，所以squeeze把形狀變[256]，以利後續embedding計算。

3. Experimental results

A. The best testing fid



```
● (maskgit) (base) instoria@DESKTOP-UGS26EV:~/dl/lab5$ cd faster-pytorch-fid
● (maskgit) (base) instoria@DESKTOP-UGS26EV:~/dl/lab5/faster-pytorch-fid$ python fid_score_gpu.py --predicted-path /home/instoria/d
  l/lab5/test_results --device cuda:0
  747
  100%|██████████████████████████████████████████████████████████████| 15/15 [00:02<00:00,  5.57it/s]
  100%|██████████████████████████████████████████████████████████████| 15/15 [00:02<00:00,  6.66it/s]
  FID:  36.95791365570432
```

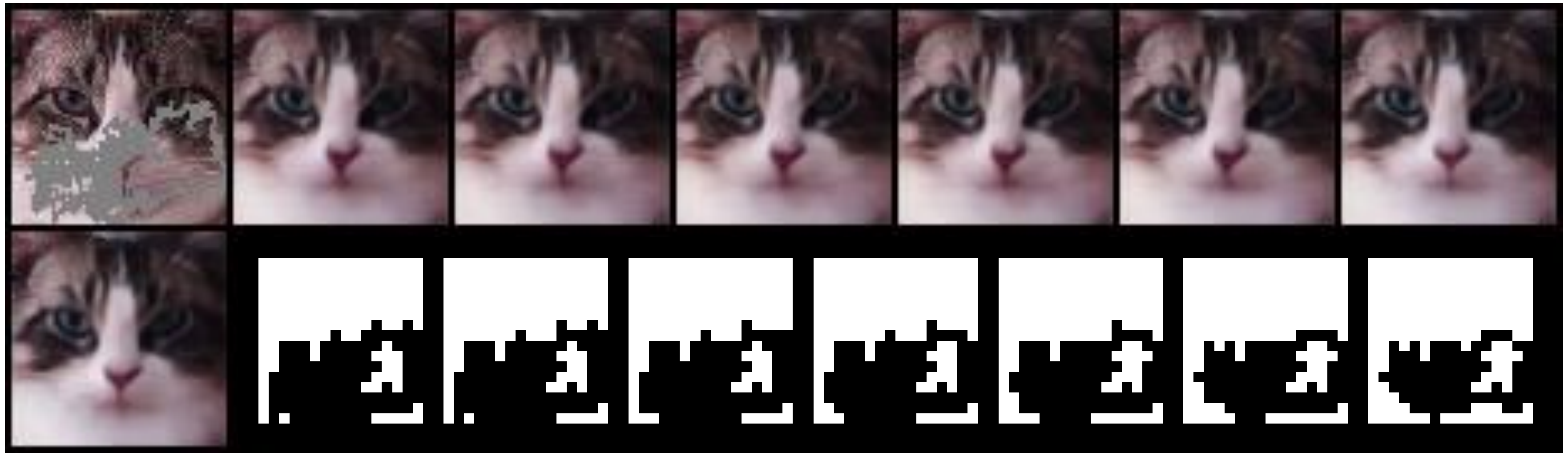
training strategy:

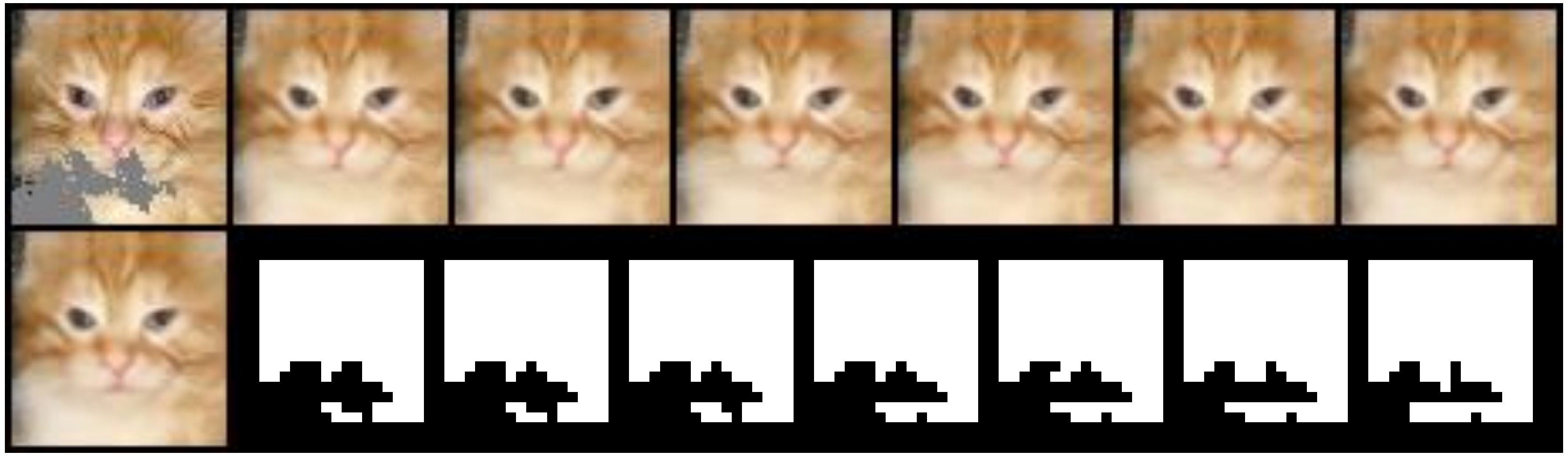
- batch-size = 10
- accum-grad = 10
- epochs = 40
- learning-rate = 0.0001
- optimizer = Adam

mask scheduling parameters :

- sweet-spot = 7
- total-iter = 12

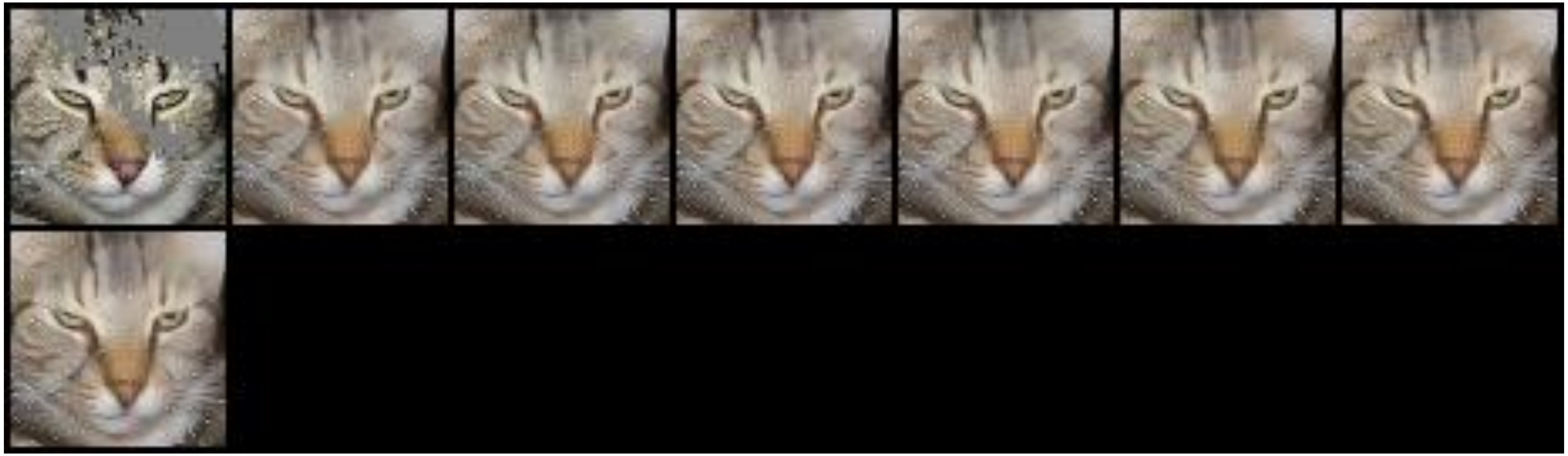
A. The best testing image





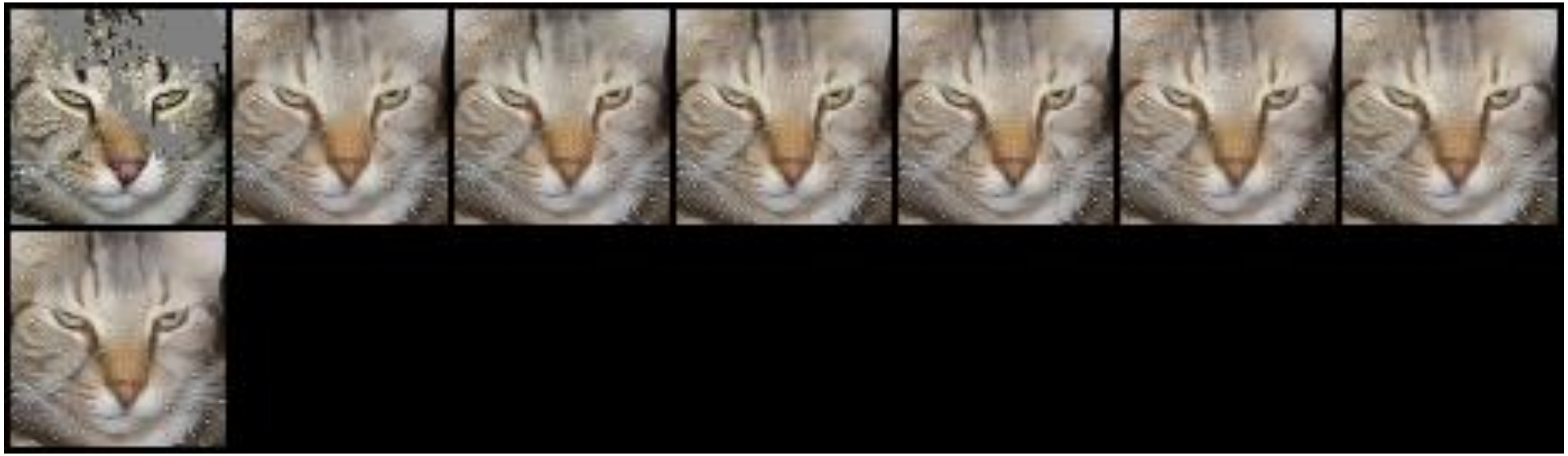
B. Comparison figures with different mask scheduling parameters setting

cosine



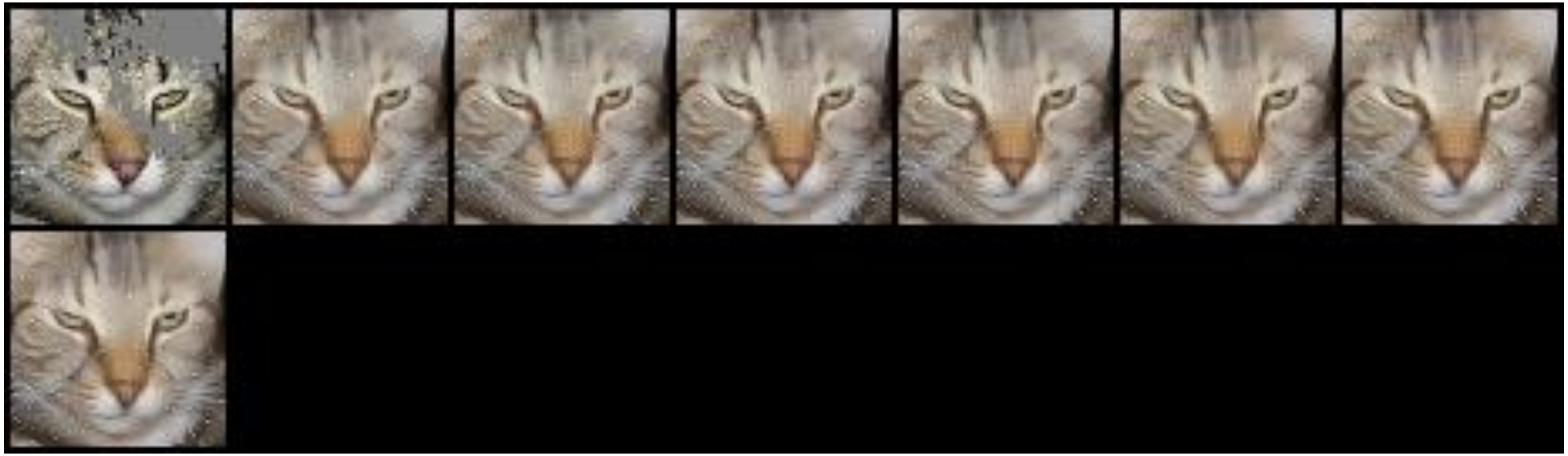
- total iteration=12
- sweet spot=7
- fid=37.168754649528466

linear



- total iteration=12
- sweet spot=7
- fid=39.408128545653824

square



- total iteration=12
- sweet spot=7
- fid=36.95791365570432

4. Discussion

- 我寫好所有TODO後，發現inpainting後的圖片模糊不清，但是嘗試了多種方法，檢查了好幾遍，花了一個禮拜都找不到原因。
- 最後發現在inpainting時，應該是要把image餵給encode to z，我餵成標準化過後的ori。
- 讀code比寫code還難。