Tiny MaskGIT

Vincent Arak (Youlong, Ding in Chinese)
2200094801
Contributed to the design of the VQ-VAE section
Contributed to the design of the Generation Mechanism section
2200094801@stu.pku.edu.cn

Qihua Sun
2200011641
Contributed to the design of the Transformer section
Contributed to the design of the Generation Mechanism section
2200011641@stu.pku.edu.cn

Abstract

We are confronted with the challenge of image generation, where the goal is to produce an image with semantic information related to a specified label, given either a designated label or an partially incomplete image with accompanying labels. We initially trained a VQ-VAE and, based on the VQ-VAE, developed a bidirectional Transformer. Finally, we employed the mask algorithm mentioned in [4] to accomplish image generation under specified conditions. Our VQ-VAE achieved excellent reproducibility on the tiny-imagenet-200 dataset, while the bidirectional Transformer (hereafter referred to as maskGIT), based on the VQ-VAE, generated images that exhibit recognizable semantic information on this dataset. Finally, we acquired some advanced techniques in using PyTorch based on our study of [3].

1. Introduction

The category of the problem we are addressing falls under the domain of image generation, encompassing conditional image generation where images are generated based on given conditions, specifically represented by labels describing the image category. This includes unsupervised image generation, where images are generated from unlabeled data, and the model needs to learn the latent structures and features within the data. Additionally, it involves partial image generation, where complete images are generated based on partial images or local information, also known as image restoration or completion.

The image generation task holds significant importance, and in the context of our implemented maskGIT model, it can play a crucial role in various aspects, including: Data Augmentation, image generation techniques are instrumental for data augmentation, expanding the scale of training datasets, by synthesizing images, the model can learn in a broader range of conditions, enhancing its generalization capabilities to diverse data; Image Editing and Restoration, image generation can be applied for editing and restoring images, encompassing tasks such as object removal, image enhancement, and local repairs.

We selected ten labels from the tiny-imagenet-200 dataset and initially trained a VQ-VAE, achieving a Frechet Inception Distance (FID) score of 69.73 (based on the comparison between reproduced images and those used for training). Subsequently, building upon this VQ-VAE, we further trained a maskGIT model on the tiny-imagenet-200 dataset. The maskGIT model was employed for both label-based image generation tasks (using only labels) and image completion tasks with given labels and incomplete images. The generated results exhibited semantic information recognizable by human observers.

In the maskGIT reconstruction task based on the given code and original image, a Frechet Inception Distance (FID) value of 69.71 was achieved. Although this metric may not be directly relevant to our primary task, it serves as an indicator of how well the Transformer fits the VQ-latent space.

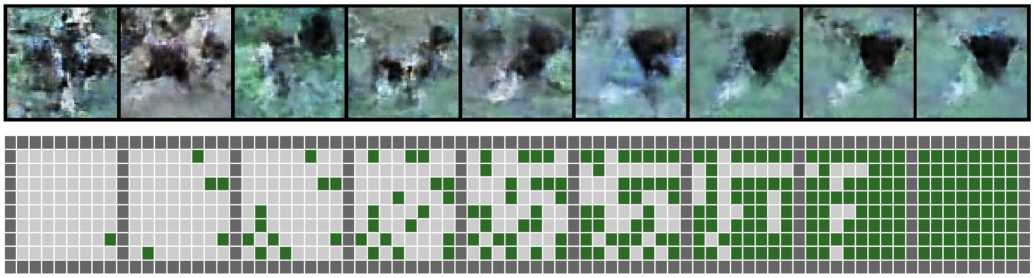

Figure 1: If you look carefully enough you may see a bear pooping in the forest

2. Related Work

2.1. Neural Discrete representation learning

In this section, we would like to focus on the differences between VQ-VAE and the typical VAE, with the most crucial distinction being the discrete representation, as introduced in [5].

1. Latent Space Representation:
   (a) VAE: In a standard VAE, the latent space is continuous and follows a Gaussian distribution. Each point in the latent space represents a set of parameters for generating an output.
   (b) VQ-VAE: In VQ-VAE, the latent space is discrete and is created using vector quantization. It means the latent space is composed of a set of discrete vectors or codes. This discrete representation allows for a more structured and controlled encoding of information.
2. Quantization and Encoding:
   (a) VAE: The encoder in VAE maps input data to continuous latent variables, and the decoder generates outputs based on samples from this continuous space.
   (b) VQ-VAE: The encoder in VQ-VAE maps input data to discrete codes (quantized vectors), and the decoder generates outputs based on these discrete codes.
3. Training Mechanism:
   (a) VAE: VAEs are trained using a combination of a reconstruction loss and a regularization term. The regularization term encourages the latent space to follow a specific distribution, typically a Gaussian distribution.
   (b) VQ-VAE: VQ-VAE uses a discrete latent space, and during training, it involves a commitment loss that encourages the encoder to map inputs to specific discrete codes, as well as a reconstruction loss for generating accurate outputs.
4. Generative Process:
   (a) VAE: In VAE, the generative process involves sampling from a continuous distribution in the latent space.
   (b) VQ-VAE: In VQ-VAE, the generative process involves selecting discrete codes from a predefined codebook in the latent space.
5. Expressiveness and Interpretability:
   (a) VAE: The continuous latent space in VAE provides a high level of expressiveness, but the interpretability of individual dimensions may be challenging.
   (b) VQ-VAE: The discrete latent space in VQ-VAE offers a more interpretable representation, as each code corresponds to a specific category or concept.

In summary, VQ-VAE introduces a discrete latent space through vector quantization, providing benefits in terms of interpretability, structured representation, and improved handling of categorical information compared to the continuous latent space of traditional VAEs.

2.2. Attention Mechanism and MaskGIT

In this section, we focus on the structure of bidirectional transformer and explain the essence of MaskGIT.

1. Attention mechanism
   The proposal of Attention mechanism, introduced by Vaswani et al. (2017)[2], lays the structural foundation of transformer model. These models have revolutionized various tasks due to their ability to capture long-range dependencies in sequential data, including language understanding, image generation, and machine

translation.
More specifically, the (self-)attention mechanism can be described by mapping an intermediate representation with three position-wise linear layers into three representations, query $Q \in \mathbb{R}^{N \times d_k}$, key $K \in \mathbb{R}^{N \times d_k}$ and value $V \in \mathbb{R}^{N \times d_v}$, to compute the output as

$$Attn(Q, K, V) = softmax(\frac{QK^t}{\sqrt{d_k}})$$

By integrating the attention module with a FFN, the attention value calculated in the formula can then further nagivate the performance of the model, especially the encoder structure.

2. MaskGIT
By extending the transformer structure into a bidirectional one, MaskGIT introduced by Chang et al.(2022)[1] has successfully revolutionized the image synthetic process by thinking out of the box of sequential generation, allowing faster and better image generation to happen.

   (a) Iterative decoding: Unlike the conventional decoding procedure, which is sequence-oriented, the decoding mechanism in MaskGIT personifies the essence of a painter's process of painting, i.e., creating some random strokes and then gradually refining in the details. That is to say, the decoding process fits in a iterative structure:
   $predict \Rightarrow sample \Rightarrow mask\ schedule \Rightarrow mask$.

   (b) Masking design: From the previous work of MaskGIT we observe that the quality of image generation is significantly affected by the masking design, and that the mask scheduling has to strictly follow a function $\gamma(r)$ ranging in $[0, 1]$. In MaskGIT, the authors tried various kinds of scheduler function including linear, concave, convex and triangular functions, and found out that the cosine scheduler outperformed the rest candidates.

   We empirically compare the above mask scheduling functions in experiment section and find the cosine function works the best in all of our experiments.

Overall, the combination of Transformer architecture, self-attention mechanism, and recent advancements such as MaskGIT has paved the way for significant improvements in (contextual) image processing tasks, demonstrating their effectiveness in capturing global information. Meanwhile, our work is also based on the above mentioned attention mechanism as well as the essence of MaskGIT. Our model has been successfully integrated into a smaller and easier-to-implement image generating network through improvement and packaging, and has played a better effect and prospect in image restoration.

3. Data

The data we are dealing with is image data, where each image has a size of (64, 64, 3), representing RGB images, along with labels used to train maskGIT. All images are sourced from the tiny-imagenet-200 dataset. The tiny-imagenet-200 dataset comprises a total of two hundred categories, and we have selected ten specific categories, namely: Egyptian cat $(n02124075)$; lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens $(n02509815)$; German shepherd, German shepherd dog, German police dog, alsatian $(n02106662)$; brown bear, bruin, Ursus arctos $(n02132136)$; goldfish, Carassius auratus $(n01443537)$; teapot $(n04398044)$; orange $(n07747607)$; teddy, teddy bear $(n04399382)$; sunglasses, dark glasses, shades $(n04356056)$; sports car, sport car $(n04285008)$. Each category consists of a total of five hundred photos for training, resulting in a total of five thousand images in our training set. We did not apply any augmentation to the images because our goal is to generate high-quality images, and the learning process requires accurate representations of high quality.

4. Method

4.1. pre-Training

At the training stage, the process is divided into two phases. In the first phase, we train a VQ-VAE, and in the second phase, we train a MaskGIT based on the trained VQ-VAE.

4.1.1 VQ-VAE

Network Architecture The VQ-VAE consists of three components: an Encoder, a Decoder, and a Vector Quantizer. The encoder includes convolutional layers to reduce the image size. Following the convolutional layers, numerous ResNet layers are connected, with an Attention layer between every two ResNet layers. The output is a tensor with a shape of (batchsize, patchsize, patchsize, embeddingDim), denoted as z.
The Vector Quantizer contains an embedding layer that stores all vectors in the entire latent space. It supports retrieving the corresponding vector through integer indexing. After obtaining z, we search for the nearest neighbor z_q in the latent space, which serves as the output and is fed into the decoder.
The decoder starts with several deconvolutional layers, followed by multiple ResNet layers. Similarly, Attention lay-

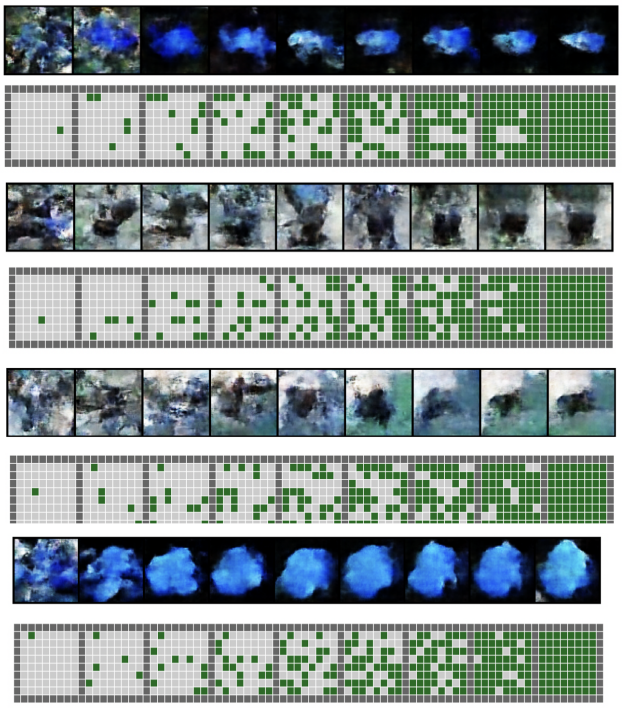

Figure 2: Generation process of the transformer model. Upper: Fish in shape of a rocket; middle: a bear standing; lower: a giant goldfish

ers are placed between these ResNet layers. Ultimately, the output is mapped through a tanh activation function, resulting in an image with dimensions [batchsize, 3, 64, 64] within the range of -1 to 1.

Training Process Each time we input a set of images into the VQ-VAE and generate an image, gradient backpropagation is performed through the following loss function. The overall loss function of the entire network consists of two components:

$$loss = \alpha_1 loss_{rec} + \alpha_2 loss_{vq}$$

$$loss_{rec} = \frac{1}{N}\sum(\hat{y_{ij}} - y_{ij})^2$$

$$loss_{vq} = log(p(x|z_q(x))) + \|sg[z_e(x)] - e\|^2 + \beta\|z_e(x) - sg[e]\|^2$$

where sg stands for the stopgradient operator, $z_e(x)$ represents z, e represents z_q, β represents commitment loss ratio, $\hat{y}$ represents generated image, y represents original image.

4.1.2 MaskGIT

Network Architecture The transformer structure in this model is mainly adapted from BERT(Bidirectional Encoder Representation from Transformers), taking tokens in both

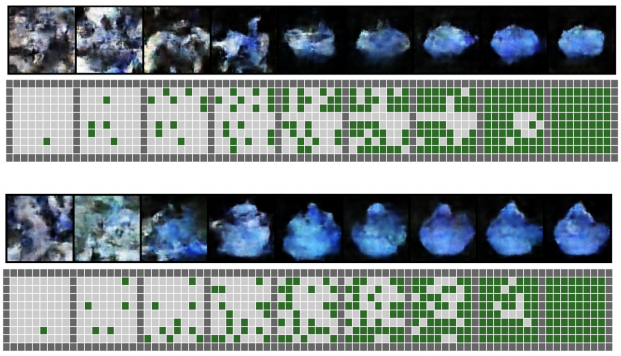

Figure 3: Generation process of the transformer model. Upper panel: teddy-fish too rare to be spotted in the world; Lower panel: teddy-teapot, masterpiece of magic

directions into consideration. The bidirectional transformer is composed of N transformer layers, which contain an encoder and a FFN respectively.
Since the transformer is insensitive to both absolute and relative position, passing through a specific pair of embedding layers before inputting is required so that the original input can carry positional information as well. Transformer's encoder relies mainly on a self-attention mechanism, specifically, the encoder passes the input tensor through a multi-head attention module to obtain the relevant attention score, and subsequently passes the obtained attention score further forward through a feed-forward neural network.
In addition, encoders can be connected in series to form a longitudinal structure - the input of the first encoder is the tensor embedded with position information, while the output of the last encoder is the attention score that needs to be provided to the feed-forward neural network, improving the performance of the bidirectional transformer.

Training Process In the training process, we take "tokenized" images created by the pretrained VQVAE network as inputs, the transformer then embeds the pictures, feed forward along the structure and then get a prediction for the probability for each masked token. We choose cross entropy loss as the criterion and back propagate the loss to rectify the performance of the model. One of the most crucial features in the training session is getting masked codes.

Get Masked Codes: The handling of the mask is a crucial part of the model building process, so the implementation of the get_mask_code function during the training process is the essential part to focus on. The function takes encoded code by VQVAE as inputs and the value taken when the mask is overwritten. By selecting a

specific masking function, it is useful for determining the ratio of the mask.

$$val\_to\_mask = \frac{2 \arccos}{\pi}$$

By converting the mask scale tensor to the same shape as the code and comparing it to the random number tensor mask, a boolean tensor is obtained where the element value True or False, indicating whether or not the element at the corresponding position is masked. The mask of the transformer model in this state is therefore obtained by the get_mask_code operation.

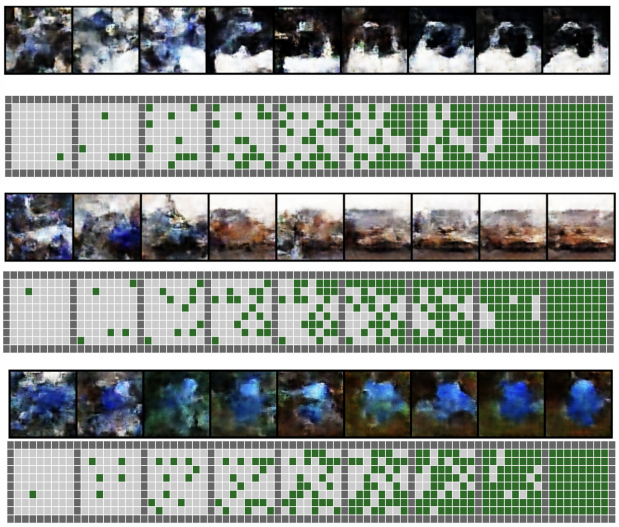

Figure 4: Generation process of the transformer model. Upper: a man with his sunglasses on; middle: an "orange" car; lower: a teddy bear walking on the street

4.2. Generation Mechanism

We have a sampling algorithm, with the most crucial inputs being the initial code and labels, corresponding to the tasks of image completion and specified-label image generation, respectively. The size of the initial code is (batchsize, patchsize, patchsize).

At the outset, we obtain an initial code and mask based on the provided initial code. If an initial code is present, we perform masking at the positions where the initial code values match the mask value. The mask value is a hyperparameter indicating the value used for masking, typically with a choose of codebooksize+1. If no initial code is available, we generate an initial code entirely masked.

Afterwards, we generate a schedule S, where $S[i]$ for $i = 0, \cdots, T-1$ expresses how many masked positions should be unveiled at each iteration. Here, T denotes the number of iterations. We employ a function $\tau(t)$ to create S, where $\tau(0) = 0$, $\tau(T-1) = 1$, and $\tau(i)N$ represents the cumulative number of unveiled positions at each step. S can then be constructed based on $\tau(i)N$. Various choices exist for $\tau(t)$, including but not limited to cosine, linear, arccos, square, root. During training, we observed that the arccos pattern for S exhibited the best performance.

At each step, we probabilistically predict the code based on the current code using a Transformer. The probability prediction involves a softmax operation, comprising a total of codebooksize+1 categories (where the last category is used to represent being masked). Subsequently, we employ the Classifier Free Guidance (CFG) method, generating a category distribution based on this probability distribution. We then randomly sample from this category distribution to obtain our predicted code for the current step. Following this, using the predicted code and the category distribution, we deduce the probability of each position under the probability distribution, serving as the current confidence score (hereafter referred to as 'conf'). We subsequently set the conf of the already unveiled mask positions to $-\infty$, select the top $S[i]$ positions from the conf tensor, unveil the mask at those positions, and proceed to the next iteration. This process continues until the final iteration, at which point we feed the current predicted code into the VQ-VAE for image generation.

4.3. Discussion on Correctness and Alternative Approaches

VQ-VAE VQ-VAE plays a pivotal role in this project by serving as a high-quality image generator (decoder) for maskGIT. In this functional role, the VQ-VAE architecture is sufficient for the task. Regarding alternatives to VQ-VAE, two directions were considered. One approach was to replace VQ-VAE with VQ-GAN, but due to the challenges faced in training GANs, such as mode collapse, and the tendency to learn the easiest-to-imitate distribution (on the MNIST handwritten dataset, GANs tend to exhibit a preference for learning the most easily imitable distribution), this option was dismissed. Another direction involved using a more advanced architecture on the Vector Quantizer, with Gumbel-VQ-VAE differing from the standard VQ-VAE by employing Gumbel-Softmax sampling. Compared to the hard sampling method in traditional VQ-VAE, Gumbel-Softmax in Gumbel-VQ-VAE enhances training stability and effectiveness. The reason for not adopting Gumbel-VQ-VAE was a lack of understanding of its underlying principles, and by the time we encountered the concept, our VQ-VAE had already achieved excellent results. However, in retrospect, considering the entire generation mechanism, adopting Gumbel-VQ-VAE could have led to better results

in the context of CFG. This is a regrettable aspect of the project.

MaskGIT Undoubtedly, the bidirectional transformer is the key focus of this programme since it is what makes the masked image generation model unique. Through the experimental results we can confirm the correctness and validity of the transformer model, but there are still questionable aspects in the choice of the model architecture as well as the treatment of the class labels. For example, the model still has a certain generative performance when no dropout is set, but the performance trend of the model under different dropout ratios is worth more attention as well.

Generation Mechanism A typical Transformer processes patches sequentially, predicting the next patch based on the adjacent ones. While this sequential approach is highly effective for strongly time-correlated data like voice and text, images possess a spatial structure that does not adhere to such temporal patterns. Instead, akin to a painter working on a canvas, the entire image space should be considered as one, and predictions should be made in a non-sequential, more jumps-and-skips manner. This fundamental difference is why our maskGIT outperforms conventional GIT in essence. It serves as a validation of the correctness of the mask mechanism.

After establishing the basic framework mentioned above, the next question is how the 'painter' (model) should choose where to make the next stroke. This involves the design of the confident score, and the current design we use is actually a replacement. Here, I will discuss our original approach. The initial idea was to randomly generate a code, predict the probability distribution using softmax at each step, employ argmax to generate the predicted code for each patch, and use the maximum predicted probability value for each patch as the confident score. This is based on an intuitive notion – when a patch has a high probability value for its prediction, it indicates a high level of confidence in its prediction, and we prioritize selecting such patches. However, based on our experimental observations, the results produced using this approach were not as good as those achieved with CFG. Thus, the validation of correctness concludes here, and the decision to replace it with CFG was made based on experimental observations.

4.4. Innovations

With the current architecture, there is still a lot of room for label expansion. The current label for maskGIT is merely one used for image classification. We can expand it to include language-related descriptions and then use a Transformer to learn from these language tokens. Connecting this language Transformer with maskGIT allows for more flexible generation tasks and can improve the learning effectiveness of maskGIT on labels.

Figure 5: reconstruction from cropped image with the transformer: our model is able to help complete cropped images with their original labels

5. Experiments

In the previous Method section, we provided a detailed description of the implementation mechanism of the entire project. Therefore, in this section, we will not delve into the specific mechanics of the experiments but instead showcase our experimental results and discuss the challenges we faced before achieving the final outcome.

5.1. VQ-VAE

Our final presentation results of VQ-VAE are shown in Figure 6. As seen, it perfectly reproduces the original images at a macro level. However, at a detailed level, there is a somewhat Gaussian-blur-like feeling, indicating a degree of blurriness. In the initial stages of the experiment, the model exhibited even more pronounced blurriness. Initially, we attributed this blurriness to the network not being large enough. Therefore, we increased the size of the network in terms of hidden dimensions and the number of hidden layers, leading to significant improvement. However, it still wasn't as good as the current state, showing clear content loss at some boundaries.

To address this, we introduced Attention layers between ResNet layers, resulting in a substantial improvement and

achieving the effect shown in Figure 6.

Nevertheless, the images still exhibit a degree of blurriness. After analysis, we believe this blurriness is closely related to the choice of the loss function. In the $loss_{rec}$ component, we used MSE loss, which does not effectively capture the structural features of the images. It merely measures the similarity between two images at the pixel level. Therefore, for further optimization, a loss function that better captures structural features needs to be chosen.

We conducted testing using FID to compare the original image set with the generated image set, ultimately obtaining a score of 69.71.

MaskGIT The final results for the task of generating images for specified labels are shown in Figure 7. The generated results for completing incomplete images (specified labels) are displayed in Figure 5. As observed, our experimental results have achieved a level of recognizability to the human eye.

However, it is evident that there is still room for optimization in our experiments. In the following analysis, I will delve into the reasons why we did not achieve a more satisfactory outcome.

Firstly, regarding the training process of maskGIT, it is trained based on being masked to fit a prediction close to the original code. However, our design approach for the interaction between categories and the model is somewhat primitive. We directly added the nclass dimensions of the category to the codesize dimension. While this is a feasible method that allows the model to be sensitive to categories, it has some essensial issues. Firstly, the training sensitivity of the model to categories is based on the softmax of codesize+1 dimensions, obtained through cross-entropy with the original code after backward propagation. It is a byproduct rather than a direct category-based training method. Moreover, it is structurally predetermined as something akin to codebooksize, which is not the case. The former is an image token, while the latter is a semantic token. They should not be confused in this way.

Figure 6: generated image of VQVAE: the left panel is the ground truth image while the right panel marks the generation result

Data Selection The issues mentioned above are structural problems. Next, we will discuss the issue related to data selection. We chose only 10 classes from tiny-imagenet-200 for training. While VQ-VAE initially demonstrated perfect results, it does not imply that maskGIT has equally low requirements for the number of classes and samples. It is highly likely that we need more classes, more samples, and more features available for learning to allow maskGIT to reach a threshold of true understanding of the world. Therefore, if further research is to be conducted to optimize this issue, we should select more diverse categories and increase the number of samples.

Next is the issue related to the correlation between the Generation Mechanism and patch size. In the original research context, they dealt with images of sizes 256×256

Figure 7: generated image given fixed label by the transformer: all the 10 classes included

and 512×512, and after passing through the encoder of VQ-VAE, the patch size of the image eventually became 16×16. In contrast, we are dealing with a problem in the context of 64×64, where the final patch size is 8×8. This significant difference in scale may lead to substantial differences in the essence of the Generation Mechanism. Therefore, we may need to explore entirely different methods for generating schedules and masks.

Next is the issue of the mismatch between the Vector Quantizer of VQ-VAE and the CFG in the Generation Mechanism. If we adopt a more advanced Gumble-VQ-VAE, we are likely to achieve better results.

Finally, some insights on hyperparameter tuning: the schedule under arccos performs the best, it is preferable to have a smaller value for the w parameter (such as 1.0) during sampling, and setting the total number of steps to 9 yields optimal results.

6. Conclusion

We have learned a new way to perceive neural networks. Before this project, our understanding of neural networks was mechanical, and we did not grasp the principles behind some design aspects. We were merely engaged in mechanical replication without understanding the underlying principles. However, after this learning experience, we gained insights into many design aspects of neural networks, such as the design philosophy of loss functions, the essence of loss functions, and where models might encounter problems due to the structure itself. We also explored how to surpass these issues. Although we did not achieve that breakthrough, the awareness of the problem, the clarity of the problem's emergence, and why the problem ultimately becomes challenging to solve constitute a complete learning process. In this project, we experienced a holistic learning process, providing a profound experience at the soul level.

References

[1] A. Alpher and J. P. N. Fotheringham-Smythe. Frobnication revisited. *Journal of Foo*, 13(1):234–778, 2003.

[2] A. Alpher, J. P. N. Fotheringham-Smythe, and G. Gamow. Can a machine frobnicate? *Journal of Foo*, 14(1):234–778, 2004.

[3] V. Besnier and M. Chen. A pytorch reproduction of masked generative image transformer, 2023.

[4] H. Chang et al. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[5] A. Van Den Oord and O. Vinyals. Neural discrete representation learning. In *Advances in neural information processing systems 30*, 2017.