Vocabulary-Free Visual Language Models

Yintao Tai

Master of Science

Computer Science

School of Informatics

University of Edinburgh

2023

Abstract

Modern language models typically rely on a finite vocabulary for both input and output, which restricts the diversity of applicable text and gives rise to a vocabulary bottleneck. Also, the large number of vocabulary embeddings brings significant computation overhead.

However, humans can robustly process langue through visual input without a defined vocabulary. Inspired by this, some existing works build language models with pure vision inputs to overcome these problems and achieve promising performance. In this research, we proposed 2 new methods for vision language models: (1) We found visual language models trained in semantic-rich latent space converge faster and conduct better downstream task performance compared with training directly on text images; (2) We proposed the Visual Adversarial GPT (VAGPT) model, which is trained with pure vision input and output. We verified the feasibility of vocabulary-free vision generative language models.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Yintao Tai*)

Acknowledgements

Many thanks to the valuable instructions and support from Dr. Antonio Vegari and Dr. Suglia, Alessandro.

Table of Contents

Chapter 1

Introduction

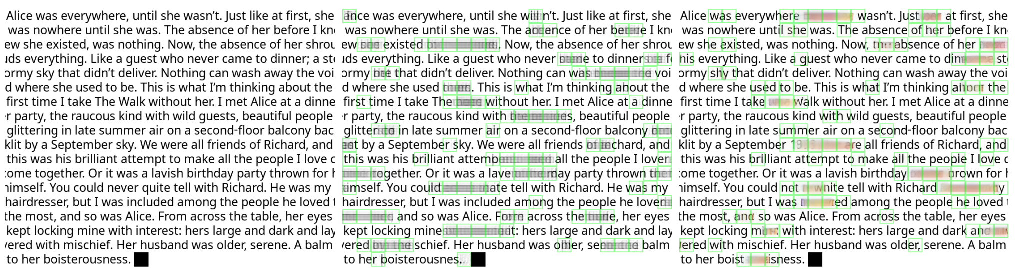
Mainstream language models rely on rule-based tokenizers, which convert input text sentences into discrete indexes within a defined vocabulary. And the model will map the indexes to embedding vectors to represent the semantic meanings. Consequently, the finite size of the vocabulary constrains the variability of the language which the model can handle and brings a vocabulary bottleneck [27]. The matrix which contains token embeddings also causes significant computation overhead during the training and the inference [15].

Subword-level tokenizers, such as Wordpiece [39] and BPE [33] tokenizers can mitigate the vocabulary bottleneck problem by encoding rare and unknown words as sequences of subword tokens. However, humans engage with linguistic information through vision or sound. Tokenizers forfeit the graphical information of text during the tokenization process, thereby creating a disparity between human language comprehension and NLP models. This discrepancy becomes even more pronounced for logographic languages like Chinese[35].

1.1 Visual Language Model in Latent Space

A natural solution is training language models with visual input as well. PIXEL [27] adapts Masked Autoencoder (MAE) [10] as a masked language model with pure vision input and achieves comparable downstream performance with BERT [5]. PIXEL also exhibits robustness to noisy inputs and generalizability to look-alike texts.

PIXEL accepts image patches containing texts as input. The training objective is to reconstruct masked text image patches. So the output is also an image. The training objective is similar to image inpainting or image generation tasks. Directly training



(a) target (b) PIXEL (c) LatentPIXEL

Figure 1.1: Reconstruction of PIXEL and LatentPIXEL. Masked patches are circled by green squares.

models to reconstruct images focus on both high-frequence details and semantic information [26]. However, we do not require the language model to perfectly reconstruct image details but to focus on high-level semantic meanings.

Thus, in this research [1], we adapted a VQGAN-based image compressor which encodes the text images into a semantic-rich space. Then we train the PIXEL on the encoded latent images. We obverse that LatentPIXEL converges faster and has better downstream task performance than the original PIXEL trained with the same setting. Figure 1.1 shows the reconstruction output of our models.

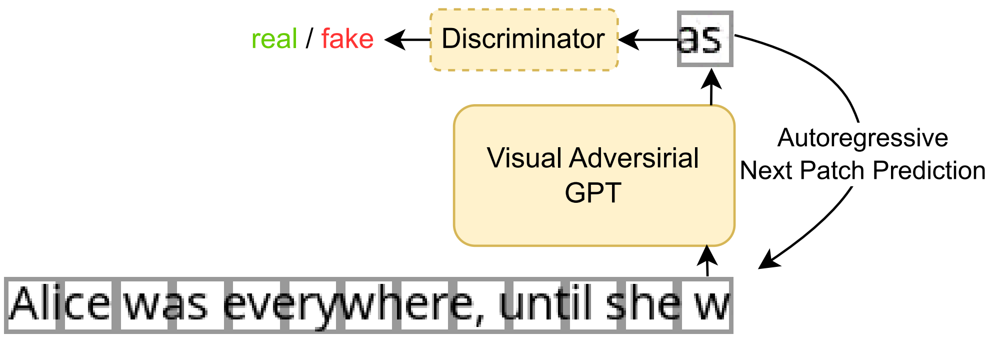
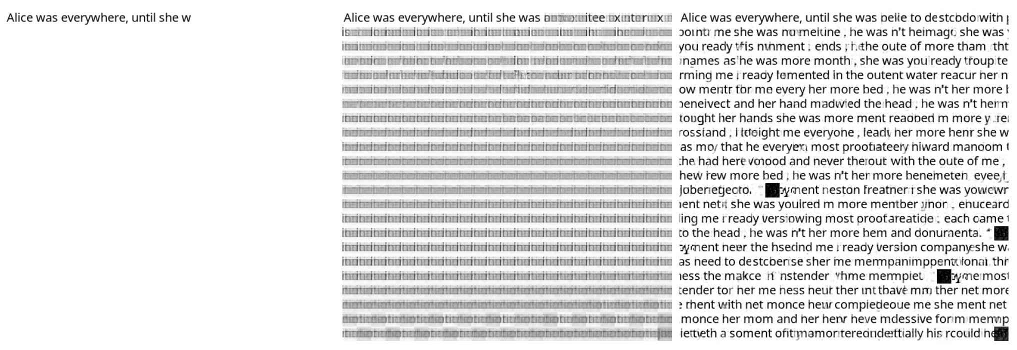

real / fake
Discriminator
as
Visual Adversirial GPT
Autoregressive Next Patch Prediction
Alice was everywhere, until she w

Figure 1.2: The Architecture of VAGPT.

1.2 Generative Visual Language Model

Similarly, [29] proposed a generative sequence-to-sequence language model with pure visual input. They append a Convolutional [23] image encoder before an encoder-

[1]Our work is accessible through GitLab. https://git.ecdf.ed.ac.uk/s1891075/msc_project



(a) prompt (b) VGPT (c) VAGPT

Figure 1.3: Text images are generated patch by patch autoregressively by the corresponding model. VGPT trained with only reconstruction loss suffers the "average patch" problem. VAGPT mitigates this problem by training with an extra adversarial fidelity loss. The quality of generated texts is not promising due to limited pretraining steps.

decoder Transformer [38] to feed rendered text images to the Transformer backbone. And the Transformer output translated texts. However, this model still suffers the vocabulary bottleneck because the output text is tokens selected from the output embedding layer.

In our implementation, we build a vocabulary-free decoder-only Transformer-based model, Visual GPT (VGPT), whose input and output are all text image patches. We exploited the GPT2 [24] as our language backbone, and linearly map image patches as embedding vectors to feed to the backbone. For the output, VGPT linearly maps the output vectors into image patches. As Figure 1.2 shows, the model generates new text in the next patch autoregressively. Also, we evaluated training in both images and in latent space.

However, VGPT suffers the "average patch" problem since predicting a non-text "average patch" has a lower loss than a wrong patch. We mitigate this problem by adding an adversarial loss. Thus, VGPT becomes Visual Adversarial GPT (VAGPT). VAGPT is capable of generating long text patches. Figure 1.3 displays the generated text images. To mitigate the mode collapse and avoid the catastrophic forgetting problem [36] of GANs [9], we applied the following measurements:

- The adversarial training is short and only applied in the final stage of pretraining.
- Linear warm up the ratio of the adversarial loss.

Chapter 2

Background

2.1 Language Model with Visual Information

2.1.1 Applications in Non-Alphabetic Languages

Language models with visual input are majorly applied for Chinese due to its loggraphic feature. Chinese, Japanese, and Korean characters have compositional parts that carry semantic meanings. [17] used a CNN-based encoding block with max-pooling to extract character-level visual representations to capture the visual semantics of the compositional parts in text classification tasks. Similarly, [34] rendered Chinese sentences into fixed-size images and thus used a CNN network to do text classifications. Models proposed by [17] and [34] solely rely on visual inputs. Researchers also explored methods to combine id-based embedding and visual information.

[20] exploited the Tianzige feature of Chinese characters, and proposed a specific Tianzige-CNN encoding layer to capture graphic information as embeddings. They combine graphic embeddings with traditional ID-based embeddings as a mixed input of RNN-based models. [4] and [35] explored using character-level visual feature for BERT-based models. [4] also applied a CNN encoder to capture visual features as Glyph Embeddings and added to ID-based embeddings as a mixed embedding. Furthermore, [35] proposed ChineseBERT which embeds each Chinese character as three embeddings, visual embedding, Pinyin (pronunciation feature) embedding, and the ID embedding as the input of the RoBERTa backbone [18]. ChineseBERT uses linear mapping to convert each character image to an embedding, rather than CNN layers.

The aforementioned works, except [34], treat each character as a token because a single Chinese character could be regarded as an individual word [35]. MacBERT [3] also uses single-character tokenization and achieved state-of-the-art performance on many Chinese NLP tasks.

2.1.2 Applications in Alphabetic Languages

Researchers also explored using visual input for alphabetic languages. [28] used visual similarity for romanization of Egyptian and Russian. [29] is a Transformer-based machine translation model that uses a CNN-based input layer to convert overlapping rendered text image patches to embeddings. [27] uses non-overlapping patches as input to the MAE backbone and is pretrained to predict the masked image patches.

2.2 Transformers in Vision

The following is based on my IPP report.

Transformer makes few architectural assumptions about the input data formats, it could be easily adapted to vision input without heavy modification. In 2021, the Vision Transformer (ViT) [6] successfully applied Transformer in vision tasks and achieved comparable performances with SOTA models on several vision benchmarks. ViT segments input images into 16-by-16 patches and maps them into vector embeddings through a linear projection. Unlike the convolutional neural networks (CNN) [23], Transformers lack the inductive bias for 2-dimensional images. ViT uses a set of learned positional embeddings to embed the position information of each image batch.

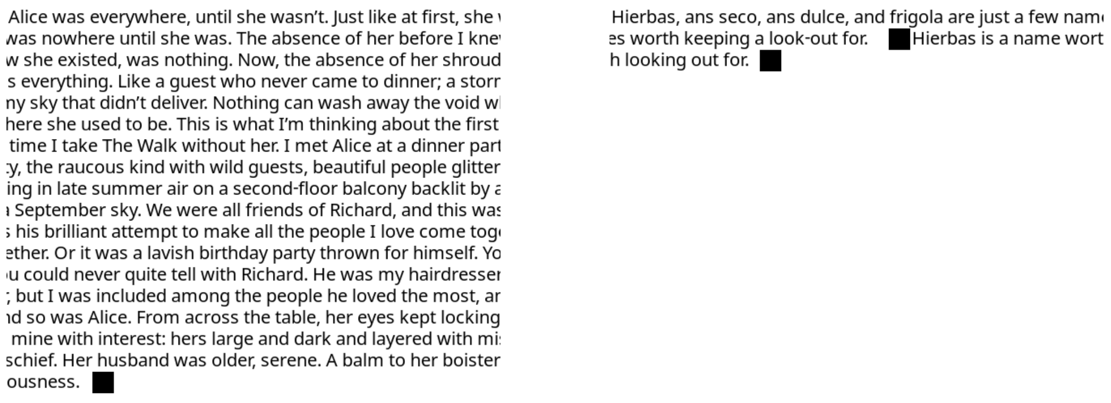
ViT is trained with the image classification objective, lack of generation ability. Inspired by the Masked Language Models (MLMs) in NLP, MAE-ViT [10] is trained by predicting a masked image patch, displays of applying NLP training objectives in vision tasks. MAE-ViT is the backbone model of PIXEL, which is also trained with a similar MLM objective.

Similarly, BEiT [2] is also trained by predicting masked image patches. The difference between BEiT and MAE-ViT is that BEiT does not directly take raw image patches as input, but encodes the image patches into a latent space by a Quantised-Variational AutoEncoder (VQVAE) [37]. Training in a latent space brings several potential benefits, such as reducing the computational cost of the Transformer block and leaving the high-frequency details to the VQVAE, therefore, the backbone can focus on semantic meanings rather than image textual details [26].

Chapter 3

Methodology

3.1 Text Render


Alice was everywhere, until she wasn't. Just like at first, she
was nowhere until she was. The absence of her before I kne
w she existed, was nothing. Now, the absence of her shroud
s everything. Like a guest who never came to dinner; a storr
ny sky that didn't deliver. Nothing can wash away the void w
here she used to be. This is what I'm thinking about the first
time I take The Walk without her. I met Alice at a dinner part
y, the raucous kind with wild guests, beautiful people glitter
ing in late summer air on a second-floor balcony backlit by a
September sky. We were all friends of Richard, and this was
his brilliant attempt to make all the people I love come tog
ether. Or it was a lavish birthday party thrown for himself. Yo
u could never quite tell with Richard. He was my hairdresser
; but I was included among the people he loved the most, ar
d so was Alice. From across the table, her eyes kept locking
mine with interest: hers large and dark and layered with mi
schief. Her husband was older, serene. A balm to her boister
ousness. ■

Hierbas, ans seco, ans dulce, and frigola are just a few nam
es worth keeping a look-out for. ■Hierbas is a name wort
h looking out for. ■

Figure 3.1: The left-hand side image is the rendered text of 1 sentence. The right-hand side is the rendered text of a pair of sentences split by a black patch. For better demonstration, we fold 1-line images into a square image.

Our model accepts images with text as input. As the figure 3.1 shows, we render a sample of text into one line as one RGB or grayscale image using the PangoCairo-TextRenderer render from PIXEL. All line breaks are ignored to keep one sample of text rendered in one line. The rendered image composes 529 patches with the same width. A text image could be represented as $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n)$, where $\mathbf{p}_i \in \mathbb{R}^{c \times w \times w}$. c is the number of channels and w is the width of an image patch. $c = 3$ when the image is rendered into RGB color. For grayscale, $c = 1$.

We render the end of sentences as black patches and padding all images into 529 patches using white patches. For sentence-pair input, we use the black patch as the splitter.

Patch width, font, font size, and DPI are configurable for different experimental setups. We will demonstrate the settings in 5. For computational efficiency, we parallelize the rendering process during training to take advantage of multi-core CPUs and pre-render the next batch of images asynchronously.

3.2 Adversarial Training

3.2.1 "Average Patch" Problem

Different from traditional generative language models which always output valid text within the vocabulary. As Figure 1.3(b) exhibits, VGPT generated only one patch text and then begin to generate non-text or unrecognizable image patches. We hypothesize this problem is due to some non-text patches, such as an ”average patch”, having lower loss compared with a wrong text patch. VGPT prunes to generate ”average patches” when the next patch is uncertain or unpredictable.

3.2.2 Fidelity Loss

To mitigate this problem, we apply an adversarial loss from a lightweight discriminator $\mathbb{D}$ which classifies whether an image patch is generated or real. The discriminator is trained to minimize the classification loss but the generator is trained to maximise it. Therefore, one of the generator's objectives is to generate image patches with high fidelity to fool the discriminator to classify them as real. We regard the inversed classification loss as fidelity loss. When fidelity loss and reconstruction loss are properly balanced, the model can generate long and valid text images rather than non-text "average patches". We refer to VGPT trained with fidelity loss as Visual Adversarial GPT (VAGPT).

We can regard the output of the generator as a distribution $p'(\hat{\mathbf{x}}_{i+1}|\mathbf{x}_1,...,\mathbf{x}_i)$. For a generated next patch $\hat{\mathbf{x}}_{i+1}$, we use the Mean Squared Error (MSE) between $\mathbf{x}_i$ as the reconstruction loss. For the discriminator, we use the Cross-Entropy between $p'(\hat{\mathbf{x}})$ and $p(\mathbf{x})$ as the loss. Therefore, the complete training objective is to find a Nash equilibrium [22] of $\theta_{\mathbb{G}}$ and $\theta_{\mathbb{D}}$ in a minimax game.

$$\arg\min_{\theta_{\mathbb{G}}}\max_{\theta_{\mathbb{D}}} \lambda[\mathbb{E}_{\hat{\mathbf{x}}\in p'}[\log \mathbb{D}(\hat{\mathbf{x}})] + \mathbb{E}_{\mathbf{x}\in p}[\log(1-\mathbb{D}(\mathbf{x}))]] +$$
$$(1-\lambda)\mathbb{E}_{\hat{\mathbf{x}}_{i+1}\in p'(\hat{\mathbf{x}}_{i+1}|\mathbf{x}_1,\ldots,\mathbf{x}_i),\mathbf{x}_{i+1}\in p(\mathbf{x}_{i+1}|\mathbf{x}_1,\ldots,\mathbf{x}_i)}[\mathrm{MSE}(\hat{\mathbf{x}}_{i+1},\mathbf{x}_{i+1})]$$

In the above equation, λ is the adversarial ratio which controls the significance of the recognizability objective.

3.2.3 Two-Stage Training

GANs often catastrophic forgetting and mode collapse problems. Our purpose of adversarial training is only to mitigate the "average patch" problem. And language understanding is harder than generate high-fidelity images. Thus, VAGPT is pretrained in two stages. The first long stage we only apply the reconstruction loss. The adversarial fidelity loss is only applied in the short second stage. Also, in this stage we linearly warm up the fidelity loss ratio from 0.

3.3 Training in the Latent Space

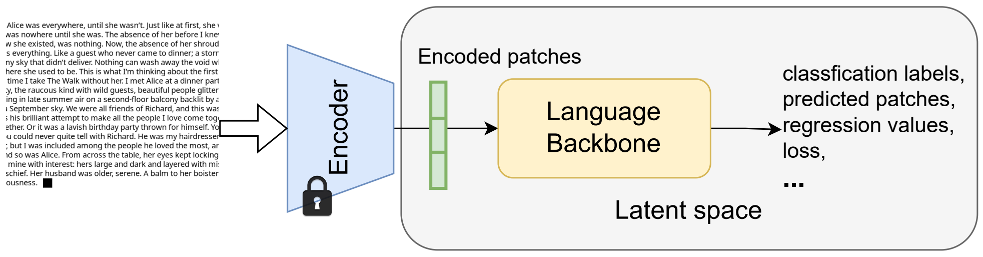


Figure 3.2: Models are trained in the latent space.

Our target is to understand or generate the text contents in images but not to reconstruct the image itself. Thus, the mismatched pixel details and mislocation of the contents are irrelevant to our objective. However, computing the loss directly on the text images will take the high-frequency details into consideration. To mitigate this problem, following other generative models in image or multi-modal tasks [26, 25, 41], we apply a VQGAN [7] as an image compressor to compress raw high-resolution text images into a low-resolution latent space where image representations are more effective [26].

As Figure 3.2 shows, when the model is trained in latent space, we first apply the encoder of the compressor to encode each image patch into a sequence of low-dimensional vectors and feed them into the language backbone. For MLM training, we randomly mask some of the encoded patches and the model is trained to reconstruct the masked encoded patches. For Causal Inference training, the objective is to predict the value of the next encoded patch at every feasible position of the input sequence.

All the training is done in the latent space thereby we do not need the decoder during the training. For most discriminative tasks, such as classification or regression, the output of the model is labels or continuous values corresponding to the task. The decoder is functionless for these tasks thus we will not load the parameters of the decoder to reduce the cost.

In our implementation, we didn't train the compressor model on our own but directly use the trained VQGAN model from Stable Diffusion [1]. We freeze all parameters of the compressor during the training.

3.4 Text Recognizability Measurement

Applications of generative language models, such as chatbots, AI translators, text summarization, etc., require the model to generate correct and human-understandable texts. Therefore, we need to consider both the correctness and the recognizability of the generated contents when measuring the quality or performance of the generative language models. However, the mainstream generative models do not suffer the recognizability problem because they only direct generate texts defined in a fixed vocabulary and these defined texts are always recognizable. Thus, traditional metrics, such as perplexity, for generative language models do not consider recognizability.

Also, the traditional metrics for image-generation tasks focus on the quality or the fidelity of generated images rather than the accuracy of texts inside and therefore do not suitable for our purpose.

To consider both the correctness and recognizability of generated text images, we propose a simple metric which is the recognized Edit-distance (RecDist). The RecDist is the character-level Edit-distance between the ground-truth text and the recognized text averaged by the length of the ground-truth text.

Suppose we have a sequence of characters $S = (s_1, s_2, ..., s_n)$ as the ground truth and an image I of predicted or reconstructed text. To measure the RecDist, we first use an OCR tool to recognize the text in the image, then calculate the Edit-distance between the ground-truth text and the recognized text. Finally, average the distance with respect

[1] Parameters are available here https://huggingface.co/stabilityai/stable-diffusion-2-1

to the number of characters of the ground-truth text. The RecDist could be defined as the below equation.

$$\text{RecDist}(S, I) = \text{EditDistance}(S, \text{OCR}(I))/n$$

In this research, we use the Tesseract OCR[2] software, which contains a lightweight LSTM-based [13] text recognition model. To calculate the Edit-distance, we use Python's "editdistance" package [3].

[2]Tesseract OCR is publically available here https://github.com/tesseract-ocr/tesseract
[3]"editdistance" package is publically available here https://pypi.org/project/editdistance/

Chapter 4

Model Architectures

4.1 Transformers

Both of our backbones apply the Transformer structure. Transformer is a deep neural network architecture proposed by [38] in 2017 and became the dominant choice for natural language processing (NLP) tasks in following years [6]. The Transformer structure was also been proven effective in vision tasks, including image classification, image recognition, [6, 10], and image generation [25] tasks. In our proposed models, we also adapt the Transformer structure. The original Transformer model is composed of an encoder and a decoder. Each layer of the encoder and the decoder contains self-attention or encoder-decoder-attention layers. The encoder-decoder-attention is also referred as cross-attention in the following works [16].

4.1.1 Self-attention Mechanism

For NLP tasks, the semantic meaning of each token relies on all other tokens in the context. To model this relationship, the self-attention accepts a sequence of vectors $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ as input, each of them represents one input token. Each input embedding is linearly mapped to 3 vectors, k, q, and v. If we write them into matrices, the procedure could be represented as

$$\mathbf{Q} = \mathbf{W^Q X}; \quad \mathbf{K} = \mathbf{W^K X}; \quad \mathbf{V} = \mathbf{W^V X},$$

where $\mathbf{W^Q}$, $\mathbf{W^K}$, and $\mathbf{W^V}$ are h by h matrices and h is the dimensionality of hidden embeddings $\mathbf{x}_i$. It's not necessary to make the $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ have the same dimension. But for simplicity, we follow the other works to use the same hidden dimension h

through the network. Thus, $\mathbf{q}_i \in \mathbb{R}^h$ represents the "query" of $\mathbf{x}_i$, $\mathbf{k}_i \in \mathbb{R}^h$ represents the "key" of $\mathbf{x}_i$, and $\mathbf{v}_i \in \mathbb{R}^h$ represents the "value" of $\mathbf{x}_i$.

The attention score from $\mathbf{x}_i$ to $\mathbf{x}_j$ is the dot production of related "query" and "key" scaled by the dimensionality.

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{h}}$$

Then, the embedding of $\mathbf{x}_i$ is the average of "values" weighted by the softmax of the attention scores.

$$\text{SelfAttention}(\mathbf{X}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{h}})\mathbf{V} \tag{4.1}$$

In our settings, we apply the multi-head self-attention, which split each vector of $\mathbf{k}$, $\mathbf{q}$, and $\mathbf{v}$ into k sets. Thus, each of them has the dimensionality h/k. Then calculate the SelfAttention for k sets and concatenate the final result to get the h dimension result. The Transformer block then applies Layer Normalizations [1] and a feed-forward layer to the result to get the final contextualized output embedding. To mitigate the gradient vanishing problem, a residual connection [11] is applied before each Layer Normalization in the original Transformer structure.

The calculation of encoder-decoder-attention, or cross-attention, is very similar to self-attention but takes embeddings from another encoder rather than from the previous layer as input. In the original Transformer model, the decoder takes the contextualized embeddings from the encoder through cross-attention.

4.1.2 Attention Masks

To improve the calculation efficiency on modern GPUs, we pack several input sequences into a batch. Therefore, we need to pad each sequence of a batch to the same length using special meaningless padding embeddings. To avoid the token embedding $\mathbf{x}_i$ attend to the padding embeddings, we can set the attention score to $-\infty$ before the softmax function by adding an attention mask $\mathbf{M}^A \in \mathbb{R}^{n \times n}$.

$$\mathbf{M}^A_{i,j} = \begin{cases} 0 & \text{if } \mathbf{x}_j \text{ is not a padding} \\ -\infty & \text{if } \mathbf{x}_j \text{ is a padding} \end{cases}$$

We thus modify the equation 4.1 to

$$\text{SelfAttention}(\mathbf{X}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{h}} + \mathbf{M}^A)\mathbf{V}. \tag{4.2}$$

Similarly, for models trained with Causal Inference (next patch prediction) task, we need to add causal attention masks $\mathbf{M}^C \in \mathbb{R}^{n \times n}$ to the attention scores to prevent embeddings attending the "future" information [24]. $\mathbf{x}_i$'s causal attention mask $\mathbf{M}_i^C$ is defined as below.

$$\mathbf{M}_{i,j}^A = \begin{cases} 0 & \text{if } j \leq i \\ -\infty & \text{if } j > i \end{cases}$$

The self-attention calculation of causal inference could be represented as the following equation.

$$\text{SelfAttention}(\mathbf{X}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{h}} + \mathbf{M}^A + \mathbf{M}^C)\mathbf{V}. \tag{4.3}$$

4.1.3 Transformer Layer

Both the PIXEL backbone and the GPT2 backbone exploit the Transformer layers as their main structure. Figure 4.1 shows the detailed structure of the Transformer layer. Firstly, the input embeddings are normalized with the layer-wise mean and variance. The Layer Normalization operation mitigates the "covariate shift" problem and stabilizes the training [1]. Then the normalized embeddings are passed to the Masked Multi-Head Self-Attention layer to calculate the attention between each embedding and output contextualized embeddings as Section 4.1.1 discussed. The masking strategy of the Self-Attention layer depends on the model.

There is also a residual connection between the input and the output of the Self-Attention layer to mitigate the gradient vanishing problem for deep neural networks. Then, the added results are normalized again by a Layer Normalization operation using the layer-wise mean and variance. The normalized results are passed to a 2-layer fully-connected neural network with the GELU [12] activation function. Normally, the intermediate dimension of the fully-connected neural network is larger than the hidden dimension h. Finally, the results of the fully-connected neural network are fed to a dropout layer which randomly set some of the values to 0 during the training. And add the output from the self-attention layer to construct another Residual Connection.

In our models, the position of 2 Layer Normalizations is different from the original Transformer model. The original Transformer performs Layer Normalizations after

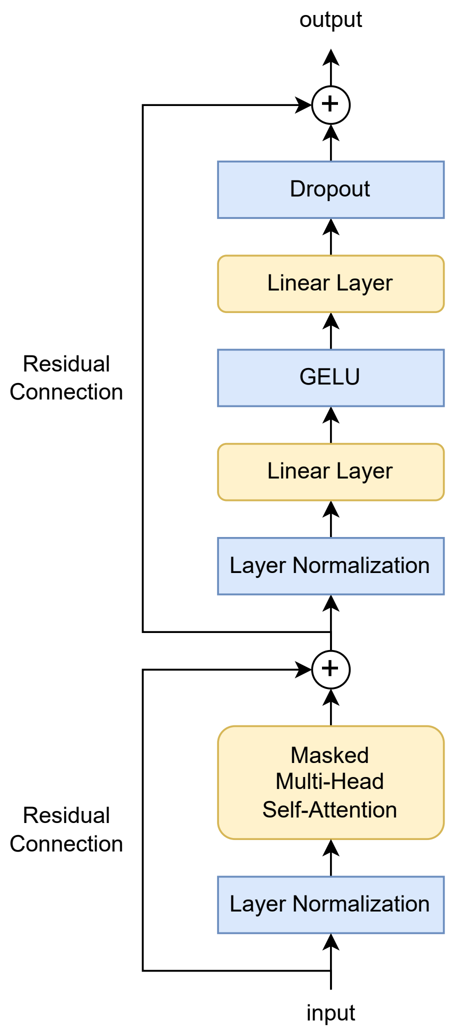


Figure 4.1: Structure of Pre-LN Transformer layer.

the Multi-Head Attention layer and the fully-connected layers in a Post-LN style [40]. Instead, our models apply Pre-LN style Layer Normalization. The Layer Normalization is applied before the Multi-Head Attention layer. Compared with Pre-LN, the Pre-LN Transformer layer is more stable during the initial stage of the training and more robust to different hyperparameter settings [40].

4.2 Patch Embedding & Transposed Patch Embedding

As we mentioned in the previous chapter, the self-attention mechanism models the relationship between every pair of input patches or tokens by calculating the attention score between every pair of keys and queries. The computational complexity of the Transformer increases quadratically with the length of the input sequence. Thereby, directly applying the Transformer structure to pixels requires dramatical computation [6]. To reduce the computation overhead, similar to other Transformer-based vision models [6, 10, 27]. Our models treat an image patch as a token rather than a pixel.

We applied the Patch Embedding as the first layer of both the PIXEL and GPT2 backbone. The functionality of the Patch Embedding layer is to firstly partition an input image into a sequence of consecutive but non-overlapping regular image patches, then map them into embedding vectors of shape h. For i_{th} patch $p_i \in \mathbb{R}^{c\times w\times w}$, the layer flattens all values into a vector $\mathbf{x}_i^{\text{in}} \in \mathbb{R}^d$, where $d = c \times w \times w$. Then conduct a dot product between x_i^{in} and a parameter matrix $\mathbf{W}^p \in \mathbb{R}^{b\times h}$ to get the embedding vector $\mathbf{x}_i = \mathbf{x}_i^{\text{in}} \cdot \mathbf{W}^p$.

The calculation could be effectively done by a non-biased Convolutional layer [23] with stride and kernel size set to w. For a Convolutional layer with stride w, kernel size $w \times w$, c input channels, and h output channels, there is a kernel tensor $\mathbf{K} \in \mathbb{R}^{c\times h\times w\times w}$. The j_{th} value of the output embedding $\mathbf{x}_i$ is the sum of the element-wise product between corresponding kernels and all input channels. The equation below precisely describes this calculation.

$$\mathbf{x}_{i,j} = \sum \sum_{k=1}^{k=c} K_{k,j} \odot p_{i,k}, \text{ where } j \in \{1, ..., h\}$$

The $\odot$ symbol represents the element-wise product between two matrices. The procedure is equivalent to the convolutional calculation when the stride length is set to w.

$$\mathbf{x}_{\cdot,j} = \sum_{k=0}^{k=c} K_{k,j} \star p_{\cdot,k}, \text{ where } j \in \{1, ..., h\}$$

The $\star$ symbol in the equation ahead is the 2D discrete cross-correlation operation with stride w.

To reconstruct the image patches from embedding vectors, we apply Transposed Patch Embedding layers at the output layer of our models. Similarly, we can effectively implement the Transposed Patch Embedding using the Transposed Convolutional layer with stride w, input channel h, output channel c, and kernel size $w \times w$. The Transposed Patche Embedding layer converts an embedding of dimensionality h into an image patch of shape $c \times w \times w$. In the original PIXEL implementation, the Transposed Patch Embedding is conducted by a series of tensor reshapes and concatenations. We implement it using the Transposed Convolutional operation to improve efficiency.

4.3 PIXEL Backbone

4.3.1 Architecture

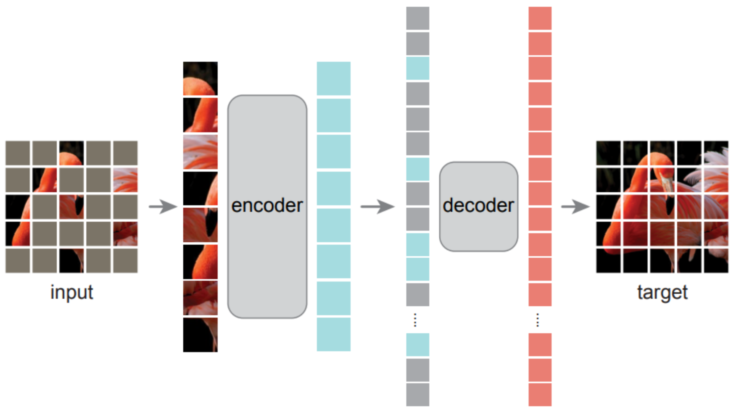


Figure 4.2: The ViT-MAE architecture from [10].

The PIXEL model is a Transformer-based masked autoencoder (MAE) [10]. It adapts the structure of ViT-MAE. As Figure 4.2 demonstrates, the model composes an encoder and a relatively small decoder. The encoder only operates on a partial of the input and others are masked [10]. When the image compressor is applied, the encoder takes the compressed image representation as input. Otherwise, the encoder directly takes the image patches as input. Then, the encoder converts each input patch into a vector embedding in the encoder's space. The decoder accepts the representations in the encoder's space as input and reconstructs the missing parts in the latent space or directly reconstructs the missing pixels depending on whether the image compressor is applied.

4.3.2 Span Masking

During the pretraining, only part of the patches are fed to the model. The ViT-MAE model selected a high masking ratio, 75%. Since the ViT-MAE is trained on natural images which are more informatically redundant compared with the human-generated text data or text images [10]. It is easier for the model to reconstruct natural images compared with text images. Thus, we select a much lower masking ratio, 25%, during the pretraining process.

Following the PIXEL, we apply the span masking strategy. We mask consecutive spans of patches according to a pre-defined distribution. We use the same masking distribution as the PIXEL, a mask may span 1, 2, 3, 4, 5, or 6 patches with respect to probability 0.2, 0.2, 0.2, 0.2, 0.1, and 0.1 for each length. Algorithm 10 represents the pseudo-code of the span masking strategy.

Algorithm 1 the Span Masking Algorithm

```
Input: #Input patches $n$, masking ratio $R$, maximum masked span length $L$, span length
    cumulative probability $P = \{p_1, p_2, ..., p_L\}$
Output: Masked patch positions $\mathcal{M}$
  $\mathcal{M} \leftarrow \emptyset$
  while $|\mathcal{M}| \leq R \cdot n$ do
      $l \leftarrow \text{randchoice}(\{1,...,L\}, P)$        ▷ Sample a span length from the distribution
      $s \leftarrow \text{randint}(0, max(0, n-l))$        ▷ Sample a starting position
      $e \leftarrow l + s$        ▷ The ending position of the span
      if $\mathcal{M} \cap \{s-l,...,s-1\} = \emptyset$ and $\mathcal{M} \cap \{e+1,...,e+l\} = \emptyset$ then
          $\mathcal{M} \leftarrow \mathcal{M} \cup \{s,...,e\}$        ▷ Make sure two spans are not too close
      end if
  end while
  Return $\mathcal{M}$
```

4.3.3 MAE Encoder

The encoder is a Vision Transformer [6, 10] but only accepts the unmasked patches as input. Thus, there is no calculation on the masking embeddings in the encoder which reduces the computation overhead.

As Figure 4.3 shows, the first layer of the encoder is a learnable patch embedding layer, which linearly maps the input patches into vectors of dimension h. In our implementation, for computation efficiency, we apply a Convolutional layer without bias to conduct the linear mapping. The stride and kernel size of the layer are set to the patch width w and the number of the output channel is set to the hidden embedding dimension h. By this setting, an image patch $\mathbf{x} \in \mathbb{R}^{c \times w \times w}]$ will produce a "single-pixel" image $\mathbf{z} \in \mathbb{R}^{h \times 1 \times 1}$ with h channels. We can easily reshape z into a vector of dimension h.

Due to Transformer layer lacks inductive bias, the model does not acquire the position information of each patch automatically like the CNN layer. Thus, we use a positional encoding layer to add position information to the patch embeddings. In our

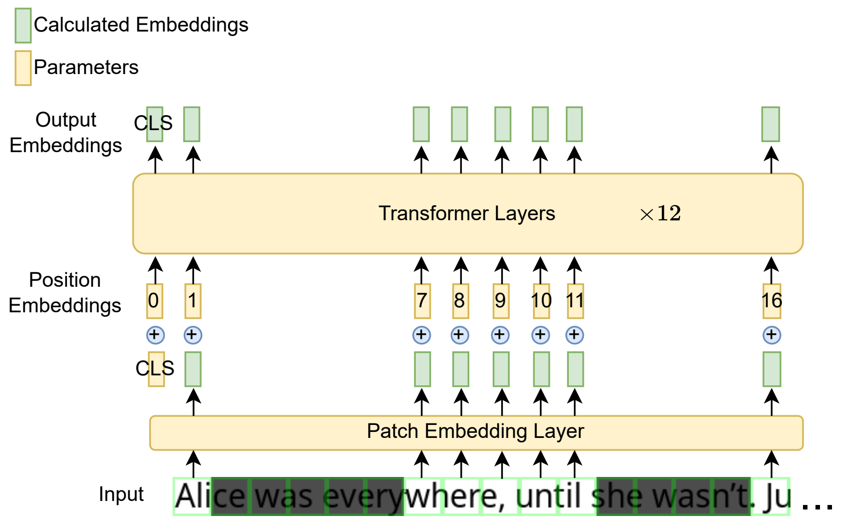


Figure 4.3: Structure of PIXEL encoder. Patches filled with black color represent the masked patches. The input is latent image patches when the image compressor is applied.

PIXEL backbone, we apply fixed position embeddings rather than relative positional embeddings. The positional embeddings are stored in a matrix $\mathbf{P} \in \mathbb{R}^{N \times h}$, where N is the maximum number of patches the model can accept. The ith vector $\mathbf{p}_i$ of the matrix represents the ith position in the input sequence. The computation of the embedding layer could be represented as Equation 4.4.

We prepend a CLS token at the start of the input sequence as Figure 3 shows. Embeddings of the CLS token are also trainable parameters. During the pretraining, the CLS token is nonfunctional. We use the output CLS embedding in downstream tasks, such as classification and regression tasks.

$$\text{PatchEmbedding}(\mathbf{x}_i) = \text{Convolutional}(\mathbf{x}_i; w, h) + \mathbf{p}_i \tag{4.4}$$

Following the patch embedding layer, there are several Pre-LN Transformer layers, as we discussed in Section 4.1.1, in the encoder. [10] refers to the Transformer layer as the ViT layer. The ViT layer has the identical structure as the Figure 4.1. Specifically, the Multi-Head Self-Attention layers are bidirectional which only applies the attention masking mechanism but not the causal masking mechanism, as Equation 4.2 demonstrates.

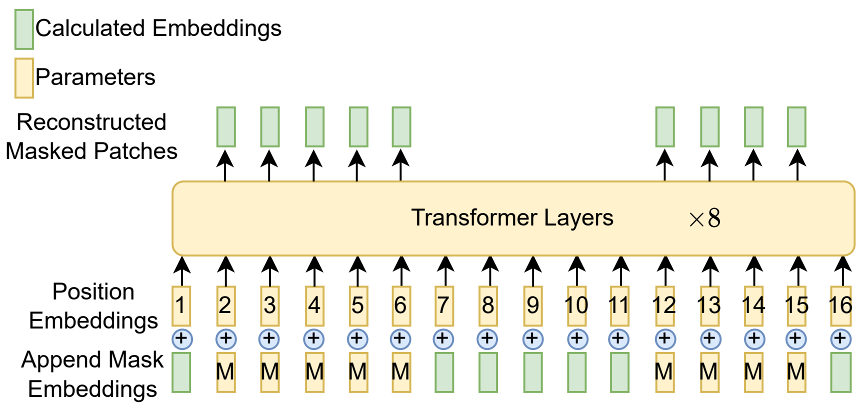


Figure 4.4: Structure of PIXEL decoder. Input embeddings are contacted with masking embeddings.

4.3.4 MAE Decoder

The decoder has a very similar structure to the ViT encoder. As Figure 4.4 shows, it also has a trainable positional embedding layer and 8 Transformer layers but with a smaller hidden dimension. The decoder takes the output of the encoder as the input and linearly maps the encoder embeddings into vectors of the decoder's hidden dimension. Because the encoder does not output the embeddings of the masked patches, the decoder appends a trainable masking embedding to the corresponding positions of the input sequence. Masking embeddings append to all positions are the same. And similarly, adds positional embeddings to each vector and feeds them into subsequent ViT layers.

The self-attention layers of the decoder also apply the calculation of Equation 4.2. Thus, the model will predict embeddings at masking positions depending on all previous and subsequent embeddings.

4.3.5 Classification & Regression

In this paper, we evaluated the model with PIXEL backbone across both regression and classification tasks. For a classification task involving C classes, we initialize a linear layer which takes the CLS embedding from the encoder as its input and outputs a vector with a dimensionality of C. Subsequently, a softmax operation is applied to this vector to derive the probabilities associated with each individual class.

Typically, in regression tasks, models are expected to generate continuous values for each sample. To facilitate this, we similarly set up a linear layer that accepts the CLS embedding from the encoder as input and generates a singular scalar value as the outcome. Following the original PIXEL model, for both regression and classification tasks, we only use the 12-layer encoder.

4.4 GPT2 Backbone

4.4.1 Architecture

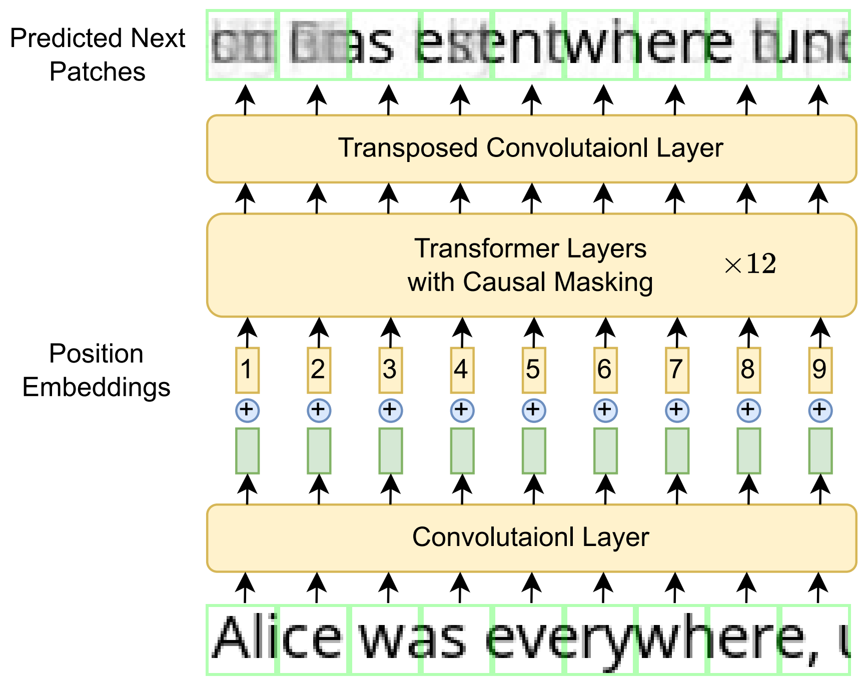


Figure 4.5: Structure of GPT2 backbone. When applying the image compressor, inputs are latent image patches.

We exploit the GPT2 model as our backbone model in generative tasks. Figure 4.5 exhibits the structure of our GPT2 backbone. The GPT2 model is a decoder-only [24] 12-layer Transformer model. The GPT2 backbone is identical to the original GPT2 model except we replace the input embedding layer as a Patch Embedding layer and the output embedding layer as a Transposed Patch Embedding layer, as we discussed in Section 4.2.

The GPT2 backbone applies the same positional embedding strategy as the PIXEL backbone. We initialize a new trainable vector of dimensionality h for each possible input position. And add the corresponding position embedding to each mapped embedding to integrate the location information. Then the embeddings of each patch are passed to the subsequent Transformer layers to predict the next patch at each position.

The GPT2 backbone also uses the Pre-LN Transfomrer layer as Figure 4.1 demonstrates. For the self-attention calculation, GPT2 attention applies both Causal Inference masking and attention masking mechanism as Equation 4.3 shows. Thus, all tokens can only attend previous tokens through uni-directional attention.

Since the uni-directional attention, the GPT2 backbone is trained to predict the next patch at all feasible possible positions and thus a generative model.

4.4.2 Discriminator

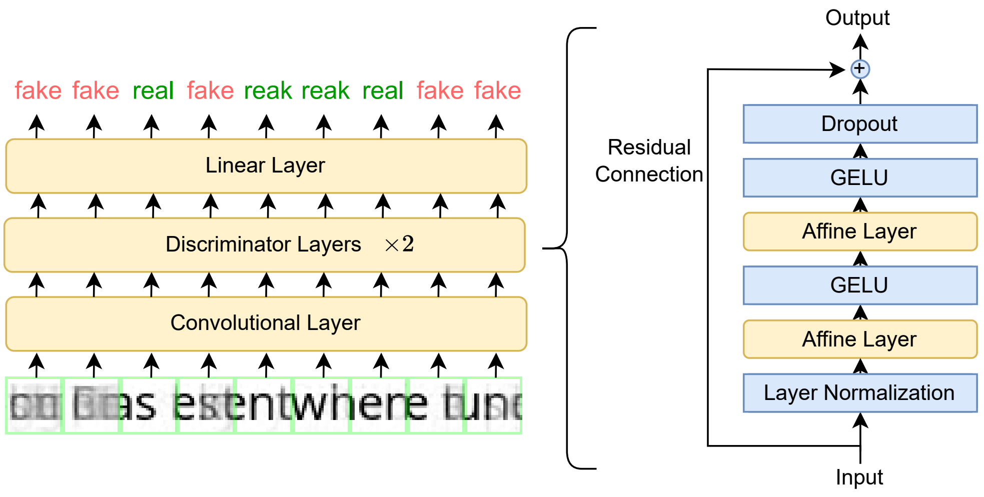


Figure 4.6: Discriminator Structure.

Figure 4.6 demonstrates the structure of the discriminator, which is trained together with the GPT2 backbone. The discriminator applies the Patch Embedding on the output of the backbone's output and judges whether a patch is generated.

Normally, the discriminator's objective is much simpler than the generator's [30]. Therefore, we design the discriminator with a lightweight structure that only has 4 Affine layers and 2 linear mapping layers. We can regard the Patch Embedding layer as 1 linear mapping layer as we discussed in Section 4.2.

4.4.3 Classification & Regression

When we apply our GPT2 backbone on classification and regression tasks, we initialize a linear mapping layer at the end of the model. The linear mapping layer maps an embedding to a distribution among C tasks or a continuous value. Since only the final embedding can attend all information from previous patches, we only input the embedding of the output embedding at the last position to the linear mapping layer.

4.5 Image Compressor

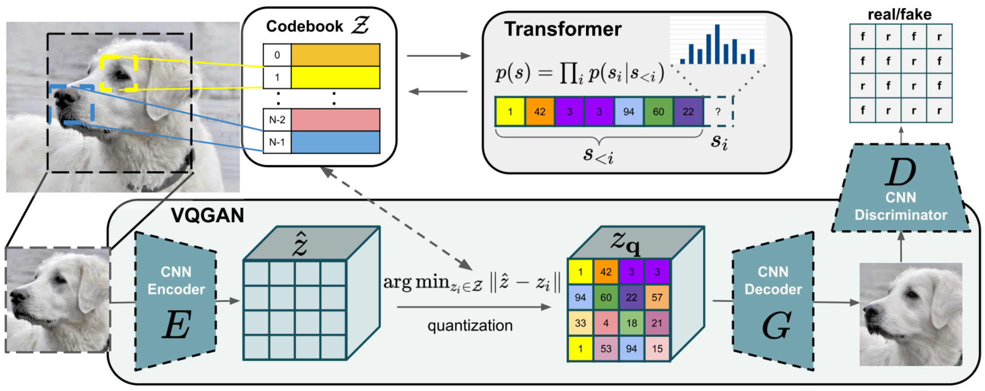


Figure 4.7: VQGAN Structure. This figure is taken from [7].

We use the VQGAN [7] structure with a compress ratio of 8 as our image compressor for our latent models. The VQGAN compresses an image patch of shape $3 \times w \times w$ into a latent patch of shape $4 \times \frac{w}{8} \times \frac{w}{8}$.

The VQGAN model is composed of CNN-based encoder and decoder, a vector quantization layer [37], and a CNN-based discriminator as Figure 4.7 shows.

During the training, the encoder converts image patches into embeddings and outputs the closest embedding in the quantization layer. Then the decoder reconstruction the image from the discrete embeddings. The discriminator is also a patch-wise discriminator trained together with other parts of the model.

However, the quantization layer is not continuous and thus not derivable. During the backward propagation, we copy the gradients of the discrete embeddings to the output embeddings of the encoder. Thus, gradients of other parts can be calculated accordingly using the chain rule.

During the inference, since the output of the encoder layer is the closest embedding of the output vector, we can skip the embedding layer and directly feed the output vector to the decoder. In our case, latent models are trained on the output vectors. We didn't train the image compressor from scratch. We directly applied the VQGAN from Stable Diffusion [26] which is trained on LAION dataset [32, 31].

Chapter 5

Experiments & Analysis

5.1 Pretraining Data

Our models are pretrained on a dataset collated from an English Wikipedia dump [8] and the BookCorpus dataset [42]. The size of the English Wikipedia and the BookCorpus dataset is 19 GB and 4.6 GB. The dataset is similar to the pretraining datasets of BERT and PIXEL. Because both Wikipedia and BookCorpus contain long documents which exceed the maximum length of the model input. We segment long documents into short documents by splitting them at line breaks. If a sample is still too long after the splitting, we only render the non-exceeding part.

Some samples are too short and require a lot of padding patches during the training. However, the padding patches do not contribute to the training but waste calculation overhead. During the pretraining, we set 400 as the minimum length for each training sample. If a sample is shorter than 400 characters, it will be concatenated to its following samples until it's longer than 400 characters.

We also balance the 2 datasets by randomly drawing samples from them with the probability proportion to their quantity. Thus, we can iterate an epoch over them at about the same time. Different from PIXE which renders all data in advance, we render all samples in real-time during the training using our parallel render software.

5.2 Recognizability Evaluation

We measured the RecDist of different rendering settings to ensure texts are recognizable after the image compressor. Table 5.1 exhibits the results with or without the image compressor. When measured with the image compressor, we first compress the image

Font	Patch Width	Width Compressed	DPI	RecDist without/with compressor
GoNotoCurrent	32	4	240	6.5e-3/6.4e-3
GoNotoCurrent	24	3	180	1.9e-2/3.8e-2
GoNotoCurrent	16	2	120	1.1e-2/0.52
GoNotoCurrent	8	1	60	0.92/1.0

Table 5.1: RecDist under different rendering settings.

and then reconstruct the image from the compressed latent embeddings. Details of the RecDist metric are discussed in Section 3.4.

When the patch width is 32 or 24, both the original image and compressed images have very low RecDist from the target text. However, when the patch width is set below 24, the RecDist of compressed images is very high and hence hard to recognize. Therefore, we trained our models with input patch width 32 or 24 in our experiments. This is different from the original PIXEL or ViT models, since they are trained with patch width 16.

5.3 Masked Language Reconstruction

We pretrained 2 models with PIXEL backbone with similar configurations and compared them on downstream tasks. We found the LatentPIXEL, which is trained in the latent space, is more robust to training hyperparameters and achieves better downstream performance.

5.3.1 Pretraining

Pretraining Hyperparameters								
Model	w	batch_size	lr	β_1	β_2	decay	steps	min_lr
PIXEL	32	196	1.5e-4	0.9	0.999	0.05	100k	1.5e-5
LatentPIXEL	32	196	1.5e-4	0.99	0.999	0.05	100k	1.5e-5
VGPT	24	196	1.5e-4	0.99	0.999	0.05	100k	1.5e-5
LatentVGPT	24	196	1.5e-4	0.99	0.999	0.05	100k	1.5e-5

Table 5.2: Pretraining hyperparameters of visual language models. Models begin with "Latent" are trained in latent space. w represents the patch width.

For both models, we conducted 100k steps of pretraining with batch size 196 and trained on text images with 32×32 patches. Following BERT and PIXEL, we applied the AdamW optimizer during the pretraining using the hyperparameters in Table 5.2. We also apply the CosineAnnealingLR [19] which decreases the learning rate from 1.5e-4 to 1.5e-5 during the pretraining. The model structure details are in Appendix A.1.

Since the pixel value varies within the range [0, 1], which is different from the input range of our image compressor. We minus each value of 0.5 and times 2 to map them into the range [-1, 1] before feeding to the image compressor. Then direct input the encoded latent value to the PIXEL backbone. When calculating the loss, we normalize the input and the output of the PIXEL backbone by per-image mean and variance. The loss is per-pixel averaged MSE between the normalized input and output. When calculating the loss, we only consider the loss on masked non-padding patches.

We also normalize the input and out of the PIXEL model. The input is normalized by per-channel mean and variance. We use the statistics from the PIXEL training scripts. The mean is [0.5, 0.5, 0.5] and the std is [0.5, 0.5, 0.5]. The output is also normalized by per-image mean and variance as we did for LatentPIXEL. The loss is also the MSE averaged on masked patches.

Figure 1.1 visually exhibits reconstructed patches after 100k training steps. When we visualize the output image, we reverse all the normalization procedures so that the output value range matches the input range. Because the parameters of the image compressor are frozen, we deleted image compressor's decoder during the pretraining. The decoder is useful only when we need to visualize the output.

5.3.1.1 Pretrain Analysis

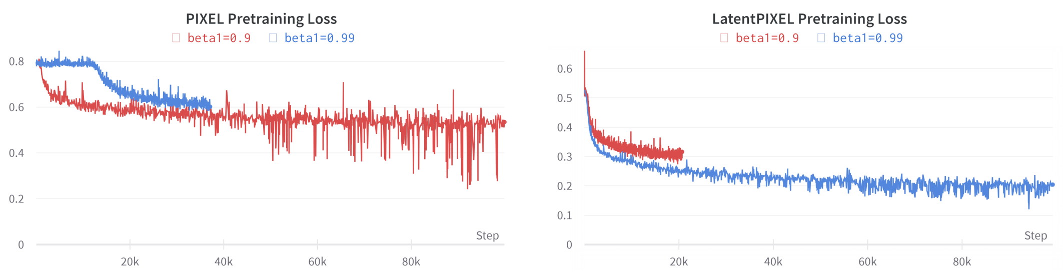


Figure 5.1: Training curves of PIXEL and LatentPIXEL.

From Figure 5.1, we can observe that when AdamW's β_1 is set to 0.99, PIXEL converges much slower than which when β_1 is 0.9 in the first 40k steps. And there are very few loss drops during the first 10k steps. However, for LatentPIXEL, the convergence speed is faster when β_1 is set to 0.99. But the loss halting problem didn't appear in the first 10k steps. Thus, we set β_1 to 0.9 and 0.99 for PIXEL and LatentPIXEL correspondingly.

Transformer-based MLM models are selective to hyperparameters [18]. However, due to limited resources, we didn't conduct exhaustive hyperparameter searching. In 100k steps, the training has not finished one epoch of the training data. Consequently, every batch during the training is unseen to the model. We can regard the training loss trend as the validation loss trend. Since it's still dropping and has not converged yet. We expect models to have better performance if more training steps are conducted.

One drawback of LatentPIXEL is the extra computation brought by the image compressor. It costs about 35 hours on 16 16GB Tesla V100-SXM2-16GB GPUs to train LatentPIXEL for 100k steps. But the same amount of training for PIXEL only takes about 16 hours on the same devices. However, the extra computation overhead could be amortized when a larger language model backbone is applied. Compared with recent large models with tens or hundreds of billions of parameters, PIXEL has only 111M parameters and the encoder of the image compressor has 34M parameters, which makes the extra computation overhead significant. The extra computation overhead will be ignorable when a much larger language backbone is applied.

5.3.2 General Language Understanding Performance

We cannot compare the performance of PIXEL and LatentPIXEL directly through losses since they are computed in different spaces and may have different scales. Hence we finetuned pretrained models on the GLUE dataset and measured the validation set performance to compare. Due to the resource limitation, we haven't done an exhaustive hyperparameter search for each task but used similar training settings as the original PIXEL paper. The 2 models are finetuned with identical hyperparameters.

5.3.2.1 GLUE Tasks

For each task, we measured the corresponding metric on the validation set according to the GLUE benchmark. We report Matthews' correlation for CoLA, Accuracy and F1 score for MRPC and QQP, Pearson's correlation and Spearman's correlation for STSB. All other metrics are accuracy.

GLUE Performance (Validation)										
Model	Patch Width	CoLA (Mat.)	SST-2 (Acc.)	MRPC (Acc./F1)	STS-B (Pea./Spe.)	QQP (Acc./F1)	MNLI-m/mm (Acc.)	QNLI (Acc.)	RTE (Acc.)	WNLI (Acc.)
PIXEL (0.1M)	32	8.2	69.2	*68.4/78.9*	17.3/16.3	77.7/65.9	50.0/50.3	*59.7*	*59.9*	**56.3**
LatentPIXEL (0.1M)	32	*9.0*	*71.6*	**64.3**/73.8	**21.9**/*20.7*	**78.7**/*71.2*	*52.9/53.8*	59.1	55.6	**56.3**
PIXEL (1M)	16	38.4	89.6	- /88.2	- /81.1	- /84.5	78.1/78.9	87.8	60.5	53.8
BERT	-	**60.3**	**92.6**	- **/90.2**	- **/88.8**	- **/87.6**	**84.0/84.2**	**91.0**	**69.5**	51.8

Table 5.3: GLUE performance of MLM models. The number beside each model indicates the number of pretraining steps. The performance of PIXEL with 1M pretraining steps and BERT are taken from [27]. For each metric, the **best overall performance is in bold**, and the *best performance of our models* is in italics.

In GLUE tasks, only STSB is a regression task and others are classification tasks. CoLA and SST2 are single-sentence tasks. MRPC, QQP, and STSB are sentence similarity or sentence paraphrase tasks with 2 sentences input for each task. For each input, we separate the two sentences with a black patch. MNLI, QNLI, RTE, and WNLI are Natural Language Inference tasks also with two-sentence input.

As we can see in Figure 5.2, the sample quantity distribution for each GLUE task is unbalanced. There are more than 100k training samples in QQP, MNLI, and QNLI tasks, but only 635 in WNLI. And the focus of each task is also different. Thus we need to set different training hyperparameters for each task. Such as, for tasks with more than 100k samples we set the total training steps as 15000 or 30000. And for WNLI we only conducted 400 steps for each model. The detailed training hyperparameters are in Appendix B. However, due to resource limitations, we didn't exhaustively search hyperparameters. We report the best validation performance in our experiments.

5.3.2.2 Performance Analysis

As Table 5.3 shows, LatentPIXEL achieves better overall performance than PIXEL when pretrained with 0.1M steps. The score is the macro-average score among all tasks. For tasks with multiple metrics, we use the averaged score as the score of that task. Also, LatentPIXEL has a higher score in the 10/14 metrics. For comparison, we also write the performance from the original PIXEL model which is pretrained with 1M steps. Although our model is not comparable with the original PIXEL because lacking

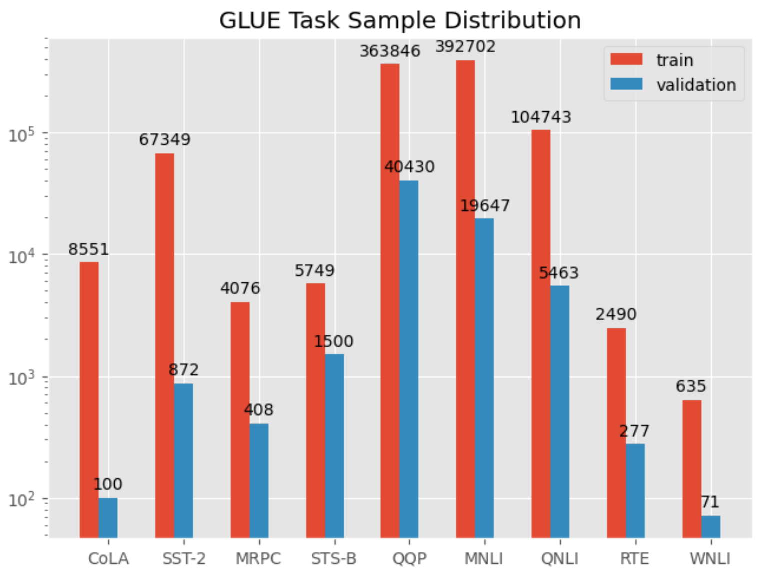


Figure 5.2: The logarithmic scale bars represent the sample counts for each GLUE task.

pretraining. Our experiments indicate training in latent space has faster convergence thus conducting better downstream task performance.

5.4 Language Generation

The performance of PIXEL and LatentPIXEL in language understanding tasks approves that vocabulary-free models with solely vision input can achieve comparable performance with traditional NLP models. In this section, we further verified the feasibility of generative language models with pure visual input and output.

5.4.1 Pretraining

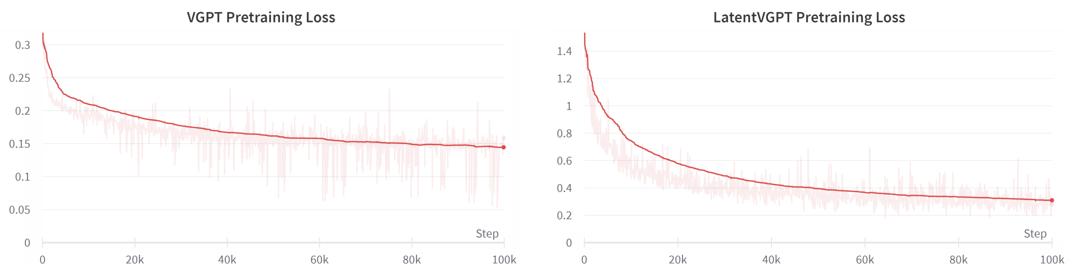


Figure 5.3: Training curves of VGPT and LatentVGPT with EMA smoothing [14].

As we did for MLM models, we also pretrained 2 generative models. One is VGPT trained with image input. Another one is LatentVGPT trained in the latent space with the VQGAN image compressor. Different from MLM models, we set the patch width as 24 to save calculation overhead. Model architecture details are in Appendix A.2.

Figure 5.3 records the pretraining losses of VGPT and LatentVGPT. We pre-trained them with identical hyperparameters for 100k steps with AdawM optimizer and CosineAnnealingLR scheduler. Table 5.2 records detailed hyperparameters.

During the training, images are normalized with channel-wise mean and variance of the corresponding dataset. For VGPT, the mean and variance are calculated among the first 4000 samples. For LatentVGPT, since it's trained in latent space, the mean and variance are calculated among the compressed latent embeddings of the first 4000 samples. Training loss is MSE calculated between the model output and the normalized input image. We regard the MSE loss as the reconstruction loss. While calculating the loss, we shift input patches left by 1 patch to correctly perform the Causal Inference task.

As we mentioned in Section 5.3.1, 100k steps do not consume a whole epoch of the pretraining dataset. Every batch is unseen to the model. We could regard the training loss trend as the validation loss trend. Figure 5.3 also indicates that the training has not converged or overfitted yet. More pretraining steps will improve the model performance further.

5.4.2 VAGPT

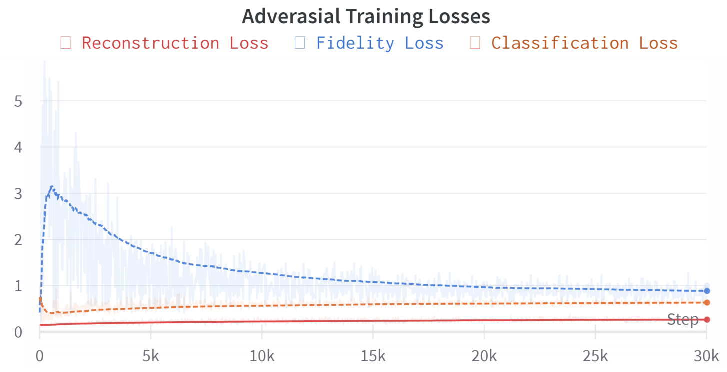


Figure 5.4: Training curves of VAGPT with EMA smoothing. The model is trained from the VGPT 100K checkpoint. We linearly warm up the fidelity loss ratio from 0 to 0.27.

As Figure 1.3(b) shows, models trained with only reconstruction loss suffer the "average patch" problem and can only generate very few text patches after the prompt.

We continued the pretraining of the VGPT for 30k adversarial steps, where the generator is trained with both reconstruction and fidelity loss and the discriminator is trained with classification loss. Thus VGPT became VAGPT. The ratio of the fidelity loss increased from 0 to 0.27 linearly during the 30k step training. The discriminator structure details are in Appendix A.3.

For each step, VAGPT is trained together with the discriminator. We first generate fake patches and update the parameters of the generator with the composited loss. Then we train the discriminator on real and generated patches for one step.

From Figure 5.4, we can observe that during the adversarial training, the reconstruction loss increases and the fidelity loss decreases. The model is focusing on minimising the fidelity loss in this stage. 1.3(c) is an example of VAGPT generation. VAGPT is capable of generating long text patch by patch, but the quality is not promising due to the limited training steps.

5.4.3 Generation Performance

RecDist on Wikitext-103				
VGPT (100k)	LatentVGPT (100k)	VAGPT (130k)	GPT2	dummy
0.77	**0.54**	0.69	0.65	0.89

Table 5.4: The number beside each model indicates the total pretraining steps. GPT2 performance is measured by the next token prediction rather than the next patch prediction. The dummy performance is the average Edit-distance between the ground truth and the sentence with the same length but filled with only the most common character.

We evaluate the next patch prediction performance on the Wikitext-103 dataset [21] using RecDist as the metric. We didn't finetune models but directly measured the RecDist on the validation set. There are 3.76k rows, or 1.1M characters, in the validation set.

When processing the data, we used a sliding window with a length of 750 characters to segment the dataset. For each partition, we first render it into image patches and predict the next patch at each feasible position using the model. Then calculate the RecDist between predicted text images and the ground truth text.

Table 5.4 exhibits the RecDist of each model. We find that although the reconstruction loss increases during the latest 30K adversarial training steps, the performance of VAGPT is still better than VGPT. We hypothesize it's due to the model stopped generating "average patches" which have lower reconstruction loss. Also, LatentVGPT pretrained in latent space has better performance than VGPT. However, we can not draw the conclusion that generative models also converge faster in the latent space because the CNN structure of the image compressor may cause information leakage between consecutive patches, which makes the next patch prediction task simpler.

The GPT2's performance is measured using the smallest GPT2. But GPT2 predicts the next token, which contains more characters than a patch, the task for GPT2 is harder. We cannot directly compare our models with GPT2. The metric is only for reference. We left the comparison between traditional generative language models to the future.

5.4.4 Calculation Overhead

Model	#Parameters	#Embedding parameters	Inference Speed
GPT2 (12-layer)	124.4M	38.6M / 31.0%	6.56 (batch/sec)
VGPT2 (12-layer)	88.5M	2.65M / 3.0%	7.52 (batch/sec)

Table 5.5: Number of parameters of the entire model and the embedding layer. For VGPT2 we count the parameters in the Patch Embedding layer and the Transposed Patch Embedding layer. The inference speed is measured on RTX3090Ti GPU with batch_size equals 16.

One advantage of vocabulary-free models is reducing the large token embeddings and the corresponding calculation. Table 5.5 shows that 31.0% parameters in the 12-layer GPT2 model are token embeddings. In our model, the input and out Patch Embedding layers only capture 3.0% parameters and thus 14.6% faster than the GPT2 model during the inference.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

In this research, we evaluated training visual language models in latent space, including BERT-style language models and GPT-style generative language models. In generation tasks, we proposed a new adversarial training to overcome the “average patch” problem. Our major contributions are:

- We found training in semantic-rich latent space converge faster and conduct better downstream task performance compared with training directly on text images;
- We proposed a new training method for generative visual language models and proposed VAGPT, which is trained with pure vision input and output. We verified the feasibility of vocabulary-free vision generative language models.

6.2 Limitations & Future Work

However, our experiments have not achieved promising downstream task results compared with modern state-of-the-art models. We expect longer pretraining and larger model scales will improve the performance. For generative models, our experiments are preliminary, the autoregressive patch-wise generation method is still under exploration. Also, compared with traditional generative language models, the patch-wise generation lacks beam-search mechanism.

In the future, we plan to train the image compressor on text images rather than natural images to have more representative visual features for text images. Also, we will conduct more training steps to improve the performance of downstream tasks. To emulate the beam-search mechanism, we will explore training language backbones on VQ embedding IDs. Our long-term goals include rendering natural images and text in the same image and exploring the multi-modality training.

Bibliography

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[2] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers, 2022.

[3] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. Revisiting pre-trained models for Chinese natural language processing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 657–668, Online, November 2020. Association for Computational Linguistics.

[4] Falcon Dai and Zheng Cai. Glyph-aware embedding of Chinese characters. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 64–69, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[7] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis, 2021.

[8] Wikimedia Foundation. Wikimedia downloads.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[10] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[12] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Frank Klinker. Exponential moving average versus moving exponential average. *Mathematische Semesterberichte*, 58(1):97–107, dec 2010.

[15] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[16] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019.

[17] Frederick Liu, Han Lu, Chieh Lo, and Graham Neubig. Learning character-level compositionality with visual features. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2059–2068, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[18] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[19] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[20] Yuxian Meng, Wei Wu, Fei Wang, Xiaoya Li, Ping Nie, Fan Yin, Muyu Li, Qinghong Han, Xiaofei Sun, and Jiwei Li. Glyce: Glyph-vectors for chinese character representations, 2020.

[21] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[22] J.F. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[23] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[24] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[25] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *CoRR*, abs/2102.12092, 2021.

[26] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[27] Phillip Rust, Jonas F. Lotz, Emanuele Bugliarello, Elizabeth Salesky, Miryam de Lhoneux, and Desmond Elliott. Language modelling with pixels, 2023.

[28] Maria Ryskina, Matthew R. Gormley, and Taylor Berg-Kirkpatrick. Phonetic and visual priors for decipherment of informal Romanization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8308–8319, Online, July 2020. Association for Computational Linguistics.

[29] Elizabeth Salesky, David Etter, and Matt Post. Robust open-vocabulary translation from visual text representations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7235–7252, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[30] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[31] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell

Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. Laion-5b: An open large-scale dataset for training next generation image-text models, 2022.

[32] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. Laion-400m: Open dataset of clip-filtered 400 million image-text pairs, 2021.

[33] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[34] Baohua Sun, Lin Yang, Patrick Dong, Wenhan Zhang, Jason Dong, and Charles Young. Super characters: A conversion from sentiment classification to image classification. In *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 309–315, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[35] Zijun Sun, Xiaoya Li, Xiaofei Sun, Yuxian Meng, Xiang Ao, Qing He, Fei Wu, and Jiwei Li. ChineseBERT: Chinese pretraining enhanced by glyph and Pinyin information. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2065–2075, Online, August 2021. Association for Computational Linguistics.

[36] Hoang Thanh-Tung and Truyen Tran. Catastrophic forgetting and mode collapse in gans. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2020.

[37] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[39] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff

Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[40] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020.

[41] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023.

[42] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Appendix A

Model Structure Configurations

A.1 PIXEL

- Number of encoder layers: 12
- Encoder hidden size: 768
- Encoder intermediate size: 3072
- Number of encoder attention heads: 12
- Encoder dropout probability: 0.1
- Number of decoder layers: 8
- Decoder hidden size: 512
- Decoder intermediate size: 2048
- Number of decoder attention heads: 16
- Decoder dropout probability: 0.1
- QKV bias: True
- Maximum input length: 529

A.2 GPT backbone

- Number of layers: 12

- Hidden size: 768
- Intermediate size: 3072
- Number of encoder attention heads: 12
- Dropout probability: 0.1
- Maximum input length: 1024

A.3 Discriminator

- Number of layers: 4
- Hidden size: 768
- Dropout probability: 0.2

Appendix B

GLUE Finetuning Hyperparameters

B.1 PIXEL with Patch Width 32

Task	lr	beta1	beta2	decay	steps	eval_freq	warmup_steps	batch_size
CoLA	2e-5	0.9	0.999	0.05	5000	50	100	64
MNLI	3e-5	0.9	0.999	0.0	30000	200	100	192
MRPC	3e-5	0.9	0.999	0.05	5000	200	100	192
QNLI	3e-5	0.9	0.999	0.05	5000	200	100	192
QQP	3e-5	0.9	0.999	0.05	15000	200	100	192
RTE	3e-5	0.9	0.999	0.05	400	20	100	256
SST-2	3e-5	0.99	0.999	0.05	15000	500	100	192
STS-B	3e-5	0.9	0.999	0.05	5000	200	100	192
WNLI	3e-5	0.9	0.999	0.05	400	5	100	192

B.2 LatentPIXEL with Patch Width 32

Task	lr	beta1	beta2	decay	steps	eval_freq	warmup_steps	batch_size
CoLA	2e-5	0.9	0.999	0.0	1000	50	200	64
MNLI	3e-5	0.9	0.999	0.0	30000	200	100	192
MRPC	3e-5	0.9	0.999	0.05	5000	200	100	192
QNLI	3e-5	0.9	0.999	0.05	5000	200	100	192
QQP	3e-5	0.9	0.999	0.05	15000	200	100	192
RTE	3e-5	0.9	0.999	0.05	400	20	100	256
SST-2	3e-5	0.99	0.999	0.05	5000	200	100	192
STS-B	1e-5	0.9	0.999	0.05	5000	50	50	64
WNLI	3e-5	0.9	0.999	0.05	400	5	100	192