



# Text to Image Generation

**Team:**

Lena ABALO

Vladyslav HERASYMENKO

Caio LANG

Pedro ROSA

Bruno SANCHES

Mars 2023

# Contents

# 1 Introduction

## 1.1 Text Conditioned Image Generation

The task of text-to-image is an evolution of the image generation task: from a given text prompt in natural language, the challenge is to generate an acceptable visual rendition for it.

One of the greatest breakthroughs in the domain was DALL-E [1], showing both scientific relevance and popular success. DALL-E's approach to text-conditioned image generation can be split in two sub-tasks: (1) convert (or "translate") from text to a latent *("hidden" or "compressed", see A.2.1 for detail)* image representation; and (2) generating an image from its latent representation.

In DALL-E, the generation sub-task is achieved with the use of a discrete Variational AutoEncoder (dVAE) [2], which is used to both (1) create a discrete latent representation of a given image; and to (2) recreate the image from its latent representation. The translation sub-task is done with an autoregressive Transformer [3], which is trained to predict image tokens (the unitary components of an image's discrete latent representation) from text tokens *(the text tokenization strategy depends on the Transformer implementation).*

The use of Transformers is at the core of DALL-E's success: as a highly scalable architecture, it made feasible the use of a 250 million image-text training dataset, obtained in great part by web crawling, ingested by a 12-billion parameter Transformer model.

Some key developments in the Variational AutoEncoder (VAE) [4] family of architectures are also key to DALL-E's success, namely the discretization (or quantization) of the latent space, present in dVAE and VQ-VAE [5] architectures. These developments and the use of Transformers for the translation sub-task are further discussed in sections B and A, respectively.

Efforts in a different line of research have discarded the Transformer-based translation of text to discrete image tokens, and have achieved state-of-the-art in the text-to-image task with the use of diffusion models [6] (GLIDE, [7]), paired with the exploitation of pre-trained joint text-image embeddings such as CLIP [8] (DALL-E 2, [9]), or with pre-trained Transformer-based text embeddings and super-resolution models (Imagen, [10]). Nonetheless, further work using Transformer-based translation and discrete image tokens has reached state-of-the-art performance by scaling the models even more and exploiting more sophisticated training strategies (Parti, [11]).

Therefore, the study of DALL-E's architecture is relevant not only for the state-of-the-art results achieved by follow-up work, but also because the fundamental elements of the architecture (discrete latent image representation, Transformer-based translation...) are seen on much of the subsequent research on the domain.

Section C presents more detail on these state-of-the-art models.

## 1.2 Contributions

The contributions brought by this project can be summarized as:

- **Bibliographic review** on the task of text-to-image, not only on the **state-of-the art models** but also on the mathematical development of some of the main used components – the **AE family of architectures** (section A) and **Transformers** (section B);
- **Explanation** of the components of a DALL-E inspired architecture in the light of the text-to-image problem;
- **Discussion** on the use of different Transformer architectures for text-to-image translation from the perspective of **inductive bias** (see B.4);
- **Experiments** in an attempt to reproduce the training of the different components of a DALL-E inspired architecture, with very limited data (MS COCO [12] Caption 2017) and computing resources (Less than 200 USD in Google Computing Platform credits).

# 2 Experiments

## 2.1 Generation: VAE and VQ-VAE implementation

One important part of the conducted experiments was image reconstruction and generation. For this purpose, we implemented first a Variational Auto-Encoder, then a Vector Quantized Variational Auto-Encoder. The aim of these experiments was the training of the models on differents datasets, some being more complex than others. We used mainly three datasets: **MNIST** dataset, **CIFAR10** dataset and **COCO-Captions** dataset.

Various architectures were implemented and tested for each of these datasets. Being more on the simple side, we started with **MNIST** dataset and obtained quite good results even with a pretty unsophisticated VAE network. We then moved to **CIFAR10**, which was more demanding in terms of processing and volume, so we had to change the network architecture. Finally, we once again modified our architecture after setteling for **COCO-Captions** dataset. We delve into more details in the following sections.

### 2.1.1 VAE implementation

Starting off with a network for **MNIST** (cf. Figure 14), it is comprised of only two convolutional encoder layers and two similar "deconvolutional" ConvTranspose2d layers. Even though the architecture itself was obviously quite simple, we still got very good results in terms of reconstruction and generation capacities of the network (cf. results section).

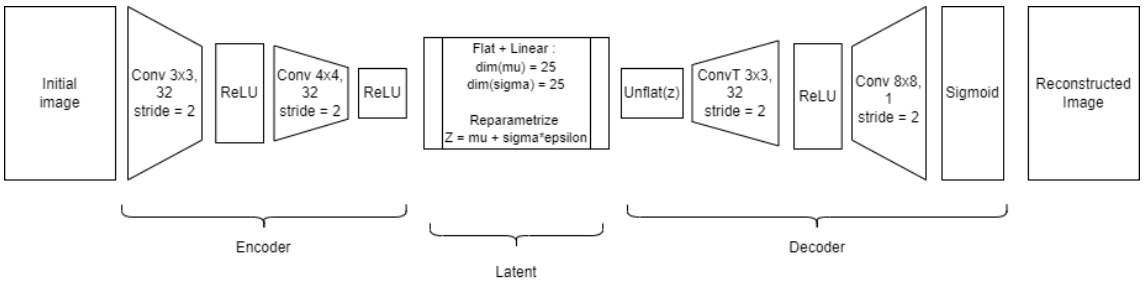



Figure 1

It's worth noting though, that the relatively good results are probably explained more by the uncomplicated nature of the dataset itself, rather than the capabilities of this particular VAE, as after trying it with **CIFAR10** the results were quite unsatisfactory.

Thus, we had to modify our architecture to represent the more sophisticated nature of **CIFAR10**.

This time (cf. Figure 15), we decided to take a pretrained resnet50 as a base for our encoder and only train the decoder from scratch (as well as the "transitional" fully connected layers). This way, we exploit the resnet to extract the important features in

encoder, transfer these features through two fully connected layers into the reparameterization kernel and then infer the reconstructed image in the decoder.

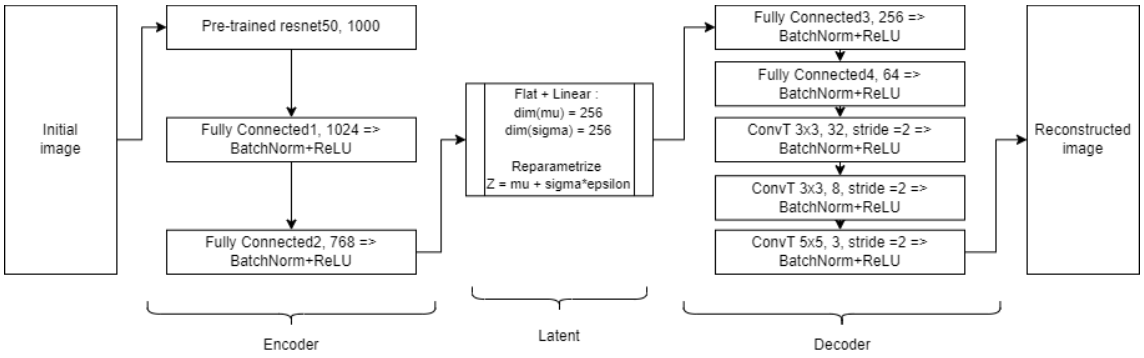



Figure 2

### 2.1.2 Resnet VQ-VAE implementation

In a second step, we moved to a little more complex architecture: a Resnet VQ-VAE. For implementing this, we decided to use a pretrained Resnet101 network as an encoder. Our Resnet VQ-VAE was made of:

- For encoding, a pretrained Resnet101 network followed by a convolution layer and a relu layer
- For decoding, one convolution layer followed by three ConvTranspose2d layers.

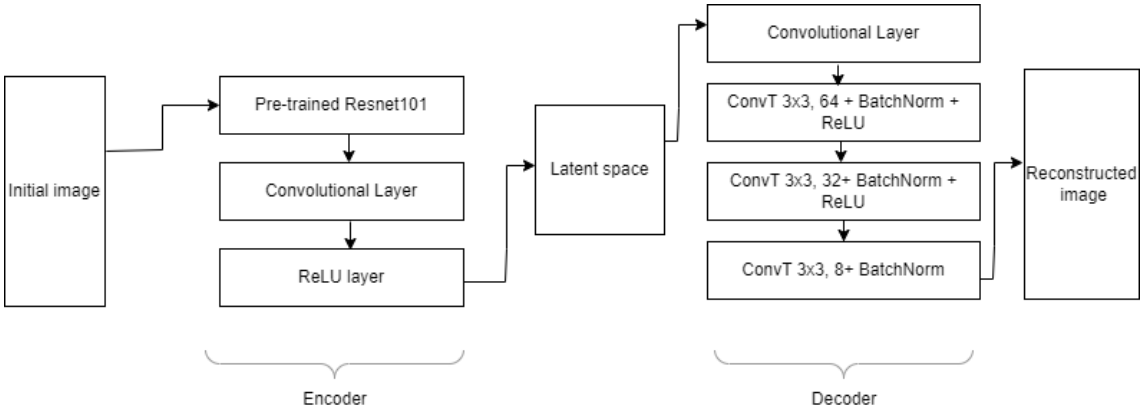



Figure 3: VQ-VAE architecture

For the loss function, we used the Mean Squared Error loss combined to the reconstruction loss.

We first trained this model on the **MNIST** dataset, then modified it and trained it on the **CIFAR10** dataset, and finally on the **COCO-Captions** dataset.

## 2.2 Translation: Encoder-Decoder Transformer

As described previously, the text-to-image translation task can be achieved by transforming a text sequence into a sequence of discrete image tokens, that encode the necessary information to regenerate the images. For this task, we use an encoder-decoder Transformer architecture that takes as input the text tokens and outputs the image tokens, which will be subsequently used as input to another decoder that transforms these tokens into images. Refer to section B.2.3 for more details.

For this task, we tried to profit from pre-trained architectures that are widely available on platforms like HuggingFace, which were trained on large text corpus to perform tasks such as text translation and generation. This has been a successful approach in DALL-E inspired projects such as [13].

### 2.2.1 Pre-trained dVAE

For the creation of the latent space and decoding, we used the discrete variational auto-encoder described in [1], that was made available in https://github.com/openai/DALL-E. This auto-encoder is used to compress the **MS-COCO** dataset images into 1024 token tensors, representing a 32x32 matrix, each token belonging to a vocabulary of 8192 different tokens. Refer to section A for more details on VAEs.

### 2.2.2 Encoder-Decoder Transformer architecture

For the translation from English text tokens to image tokens, we used a **BERT2GPT** architecture, which pairs a **BERT** [14] Encoder with the **GPT2** [15] autoregressive decoder. The text is processed by the `BertTokenizer` HuggingFace class, which corresponds to a pre-trained WordPiece tokenizer with a 30522 words vocabulary. The processed tokens are then passed to the encoder which will extract the text information and pass it to the autoregressive decoder, which then has the task to predict each token from the previous ones. This pipeline is illustrated in Figure 4.

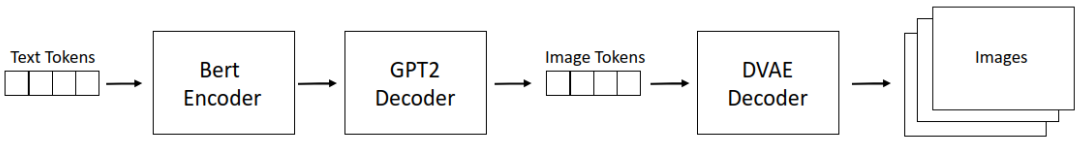



Figure 4: Text-to-image translation pipeline

During training, the image tokens are passed to the decoder, so it can infer the next token using the correct previous ones in a parallelized way. The decoder was configured to have a vocabulary of 8193 tokens, which include the image tokens to be processed by the dVAE and a beginning of sequence token to indicate the start of a sentence.

Both the **BERT** and **GPT2** models were pre-trained and available at the HuggingFace API. We fine-tuned these models for 50.000 steps with a unique NVIDIA T4 GPU which can allocate a batch size of 8 examples, we used 3 gradient accumulation steps, resulting in an effective batch size of 24. We used the AdamW algorithm for the weights optimization with the default parameters of the HuggingFace Trainer API. We used 10 warm-up steps to increase the learning rate from 0 to 0.003 and applied a linear schedule until 10.000 steps, from where we increased the learning rate to 0.0018 until 25.000 steps where it was increased to 0.0015 again, still applying the linear schedule as it can be seen in Figure 5.


Figure 5: Learning Rate scheduler during training. The different colors are the result of interrupting the training to perform tests and change training parameters such as the learning rate, the training is resumed at the saved checkpoint.

# 3 Results

In this section, we present the results of our study in terms of generation capacity of the used models, namely VAE, VQ-VAE and Text-To-Image pipeline itself. We analyze the performance of these models on different datasets such as MNIST, CIFAR, and COCO. The following subsections provide a more detailed look at the results obtained from our experiments.

## 3.1 Image reconstruction

### 3.1.1 VAE with MNIST

As discussed in the section 5.1.1, the first and the most simple model implemented, was a VAE, trained on the MNIST dataset. The reconstructional capabilities thereof are presented on the figure 22.

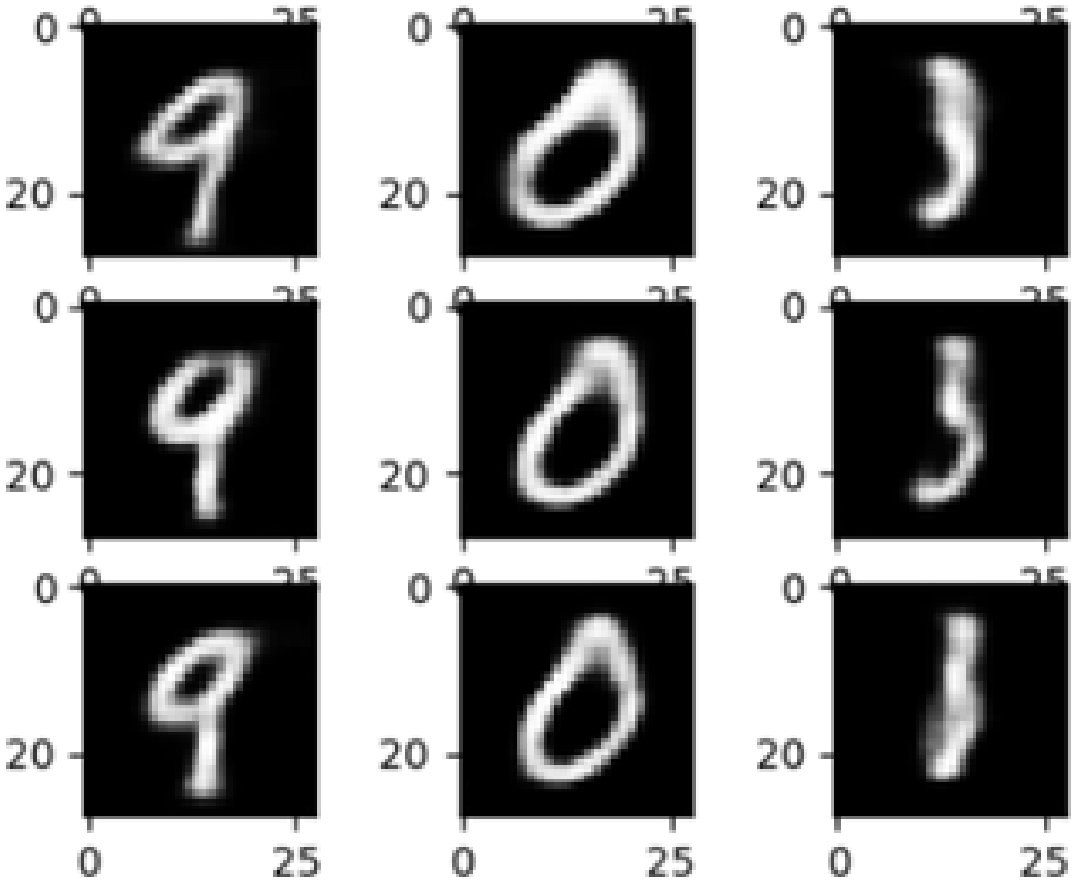



Figure 6

### 3.1.2 VAE with CIFAR10

As we saw in the previous subsection, the reconstructed images for MNIST dataset were pretty good, thus we decided to train the same network on CIFAR10 dataset. The results

are presented on the figure 23.

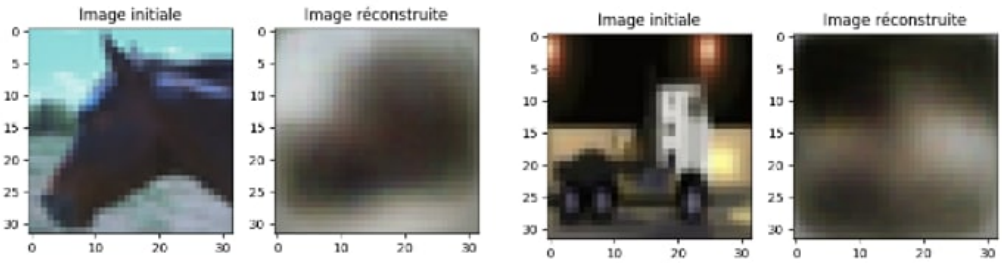



Figure 7: Reconstructed images with VAE on the CIFAR10 dataset

This time around, we observe much worse output in terms of reconstruction, as the resulting images are very noisy and hardly recognizable. This implies that the network is probably unable to capture all the different aspects of CIFAR10, which isn't surprising, considering how simple the model is.

It was thus beneficial to change the model (cf. section 5 for more details).

### 3.1.3 VQ-VAE with MNIST

In order to start with a simple base, we firstly trained our Resnet VQ-VAE on the **MNIST** dataset. We noticed that the loss was already stabilizing after 4 epochs. Here is an example of a reconstructed image with our architecture:

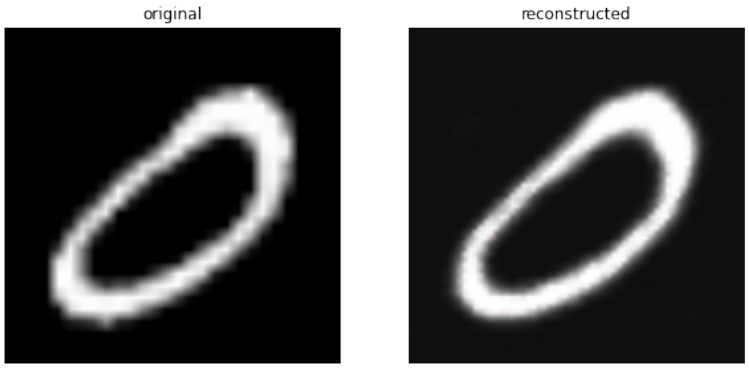



Figure 8: Reconstructed image after 4 epochs of the VQ-VAE on the MNIST dataset

Since the **MNIST** dataset is quite simple, it was important to observe the performances of our model on more complex datasets.

### 3.1.4 VQ-VAE with CIFAR10

As mentioned before, we also tested our Resnet VQ-VAE architecture on the **CIFAR10** dataset. After the good results observed on the **MNIST** dataset, the goal was to challenge

the model with a more complex dataset. After 17 epochs, the loss started to stabilize and we obtained the following reconstructions:

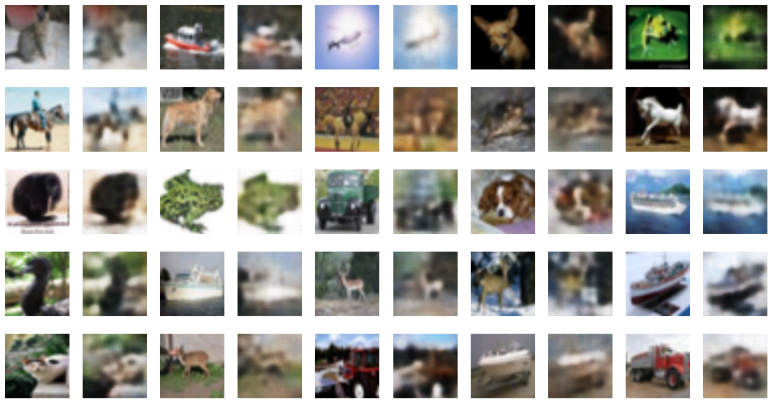


Figure 9: Reconstructed images after 17 epochs of the VQ-VAE on the CIFAR10 dataset

We computed the FID score on this generated batch and obtained a score of 7.98. This score seems too good for the reality since our images are quite blurry, but it can be explained by the fact that the score is computed on a very small batch of images.

### 3.1.5 VQ-VAE with COCO-Captions

After the **CIFAR10** dataset, we moved to a more complex dataset, and very important since it is used in every state-of-the-art models paper that we studied: the **COCO-Captions** dataset. We trained our model on the 2017 version of the dataset, and the training was obviously slower than it was for previous datasets. After 20 epochs, we obtained the following reconstructions:

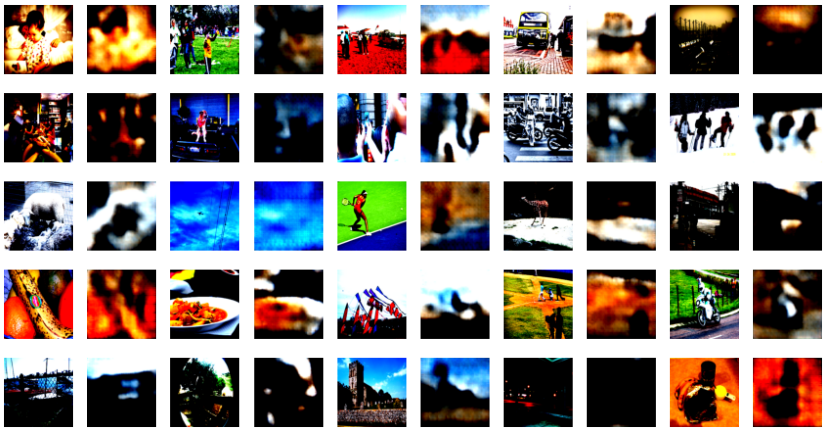


Figure 10: Reconstructed images after 20 epochs of the VQ-VAE on the Coco-Captions dataset

We also computed the FID score on this generated batch and obtained a score of

30.92. As for the previous case, the score seems too high for the quality of generated pictures; it must be also linked to the fact that the score is computed on a very small batch of images, but we struggled computing the FID score for the whole dataset because of memory allocation issues.

## 3.2 Text-to-image translation

During training, we observed a very unstable loss with very slow decay, as can be seen in Figure 11, the instability is mainly caused by the small batch size. The slow decay may be due to multiple reasons, we tested different learning rates, for smaller ones the decay is even slower and greater ones may increase the instability of the training. We evaluated every 1000 update steps, which produced the validation loss decay seen in Figure 12 showing that despite the model training, its improvement is very small.

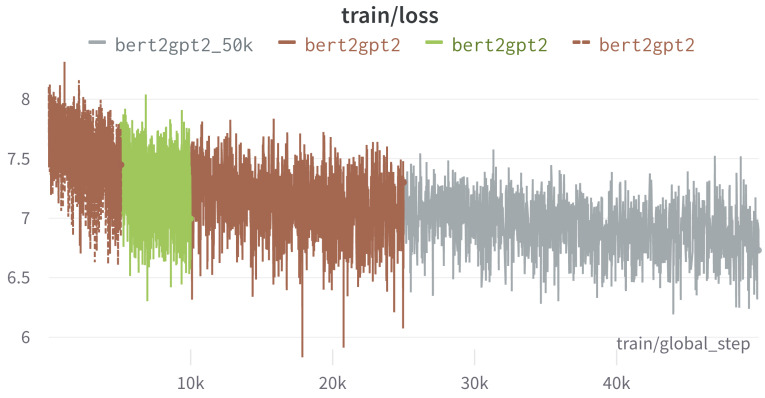



Figure 11: Training loss. The different colors are the result of interrupting the training to perform tests and change training parameters such as the learning rate, the training is resumed at the saved checkpoint.

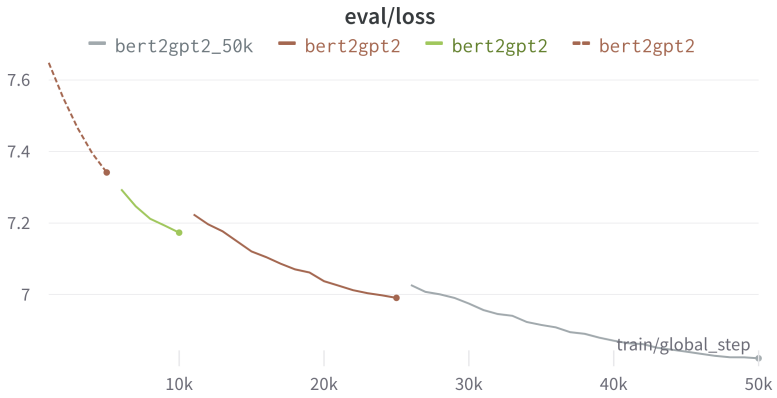



Figure 12: Training loss. The different colors are the result of interrupting the training to perform tests and change training parameters such as the learning rate, the training is resumed at the saved checkpoint.

We tested the images generated in two ways, the first one using the forward method of the network, for which we pass the text and image tokens so it can use them to predict the next tokens based on the correct previous ones, for this one we have a result that resembles the original image as in Figure 13. The second method uses the greedy search generate method which only passes the text tokens and expects the model to generate all the image tokens, taking the most probable ones, unfortunately, this method doesn't yield a meaningful image, generating repeated tokens which results in a meaningless monochromatic image. The divergence in results shows that despite the model learning, it can't yet correctly generate new images, more training may be needed so the model can learn the joint distribution of the image tokens, but the problem with the slow pace of training needs to be resolved so the model can be trained for longer periods of time and achieve a good performance.

Figure 13: The output image using the forward method.

# 4 Conclusions

With limited resources, we tried to leverage pre-trained models both for Generation and Translation, but lacked data and computing power to train well-performant models.

The conclusions for each of the two stages can be summarized as:

**Generation**

- VQ-VAE on MS COCO and CIFAR10 showed relatively structured reconstructions, with approximately correct color in many cases, even if the generated images were always blurry.
- This could also be treated with architectures such as VQ-VAE2, which try to improve resolution of generated images with different levels of codebook granularity, or with super-resolution models, such as the approach taken by Imagen.

**Translation**

- Due to the size of the pretrained encoder-decoder models, fine-tuning has proven to be a challenge, and the training process was rather unstable.
- This may be related to the fact that both BERT and GPT2 (used as pre-trained encoder and decoder, respectively) were originally pretrained on natural language, and here we have attempted to abstract this and force the models to output sequences of custom tokens, coming from the discrete image latent codebook.
- The trained encoder-decoder transformer was not able to generate feasible image tokens in an autoregressive manner (feeding the generated tokens as input) - nevertheless, when given the original tokens until a given token $t$, it seems to be able to recreate the token $t$. One possibility is that we were not able to train the Transformer enough for it to be robust to the errors in its own generated tokens.

Nonetheless, we have gained great knowledge on the task of text-to-image translation, its challenges, the state-of-the-art and the mathematical background of some of the fundamental pieces of these architectures.

# Appendices

## A Image Generation

### A.1 Introduction

There are various neural network architectures with the name *Autoencoder* (AE): AE, DAE, SAE, VAE, VQ-VAE, etc [16]. These can be seen as a *family* of architectures, and they share a common framework:

1. Input data (usually as a high-dimensionality tensor) into a specialized neural network called **Encoder**, which encodes the data into a **compressed latent space**;
2. The compressed representation of data goes through a **Decoder** network, which attempts to reconstruct the input data.

The latent space (also called "bottleneck") is of particular interest, not only for the compression of input data, but also – as will be developed further in this document – to create new data, different from what has been used for training.

In this project, we're particularly interested in the architectures of Variational Autoencoders (VAEs) and Vector-Quantized VAEs (VQ-VAEs) – and we'll also talk about Autoencoders (AEs), which is at the origin of the autoencoder family of architectures. The main ideas behind these architectures, as well as some mathematical intuition, their limits and advantages are developed in the next sections.

### A.2 AutoEncoder (AE)

#### A.2.1 Latent Spaces

Before going further in detail in the architectures of the Autoencoder family, it is important to understand what we'll call *latent space*, and why it is so important.

The term "latent" refers to a "hidden" representation of the data, usually in a space of reduced dimensionality. The latent space $Z$ can be seen as being originated from a non-linear transformation of the input data $X$ such that $Z = f(X)$, where $X \in R^n, Z \in R^m, (m < n)$ – meaning the input data $X$ encodes the same information than $Z$, but does so in a space of greater dimensionality.

Note that we could also see the input as being recreated from the non-linear transformation $X = f^{-1}(Z)$, which will be useful going further.

An example of such $X = f^{-1}(Z)$ could be a simple linear transform $X = AZ + \epsilon$, where $A \in R^{n+m}$ and $\epsilon$ is a gaussian noise of dimension $n$. In this case, the inverse transform $Z = f(X)$ could be found with dimensionality reduction methods such as Principal Component Analysis (PCA). However, if the $f^{-1}$ transform is more complex,

we'll need other approaches – and some good candidates can be found in the Autoencoder family of architectures.

### A.2.2 Architecture

As seen previously, Autoencoders are neural network architectures with the goal of learning an identity function for the input data in a non-supervised fashion. By learning this function, we'll try to create a latent representation $Z$ that encodes a compressed version of the initial data.

The model is composed of two networks (see Fig 14 for reference):

- An **Encoder** $f$, that reduces the input data's $(X)$ dimensionality, creating a representation on the latent space $Z$
- A **Decoder** $g$, whose aim is to reconstruct the input from its latent representation, having as output $X'$

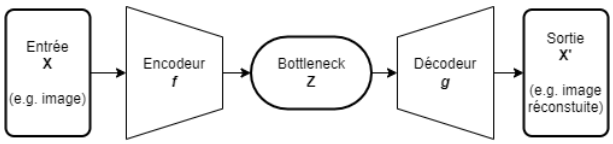



Figure 14: Scheme Autoencoder

Even though "vanilla" Autoencoders are still used and are powerful tools, they have a certain number of caveats. Firstly, the dimensionality reduction while keeping the capacity to reconstruct the input comes at a price: the latent space is not structured – making the task of new data generation infeasible. This is shown on the Fig 15.

Also, it is often necessary to manually control the AEs' networks' depth and the dimensionality of the latent space to each different task, which can be an obstacle.

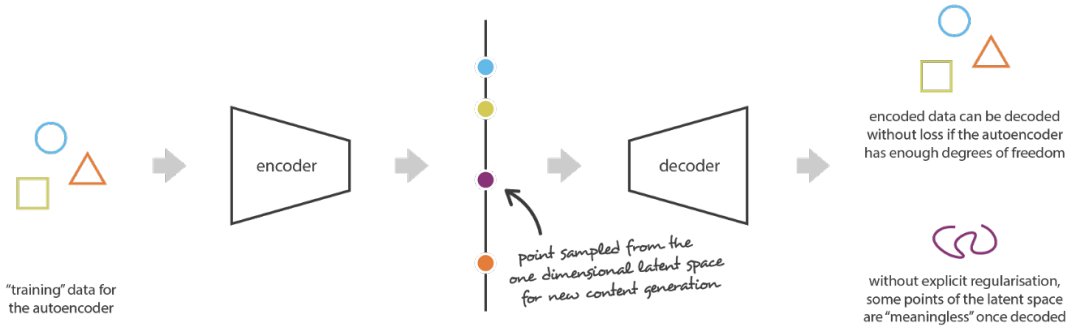



Figure 15: Irregular latent space prevents the use of Autoencoders for new content generation. [17]

## A.3 Variational AutoEncoder (VAE)

In 2014, Kingma and Welling [4] combined the basic architecture of autoencoders with Bayesian variational methods in order to correct the aforementioned shortcomings. Therefore, we can define VAEs as autoencoders that encode inputs as distributions rather than points, and whose latent space "organisation" is regularized by requiring the encoder's output distributions to resemble a typical Gaussian. In order to present a robust mathematical perspective on VAEs, we will establish a clear probabilistic framework and specifically use the **variational inference** method.

### A.3.1 Probabilistic Framework

To start, let's define a probabilistic graphical model for our data. Assuming that $x$ is produced from a latent variable $z$ (the encoded representation), which is not directly observable, we denote the variable that reflects our data as $x$. As a result, the two-step generating process described below is assumed:

- First, a latent representation $z$ is sampled from the prior distribution $p(z)$
- Second, the data $x$ is sampled from the conditional likelihood distribution $p(x|z)$

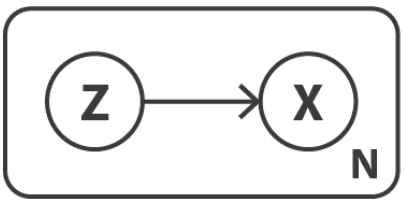



Figure 16: Graphical model of the data generation process. [17]

We can, therefore, reframe our ideas of encoder and decoder in light of such a probabilistic paradigm. Contrary to a straightforward autoencoder, which only takes into account deterministic encoders and decoders, we will now take into account probabilistic versions of these two objects. The probabilistic decoder is naturally defined by the expression $p(x|z)$, which represents the distribution of the decoded variable given the encoded one, whereas the probabilistic encoder explains the distribution of the encoded variable given the decoded one; $p(z|x)$.

Next, we make the assumption that $p(z)$ is a standard Gaussian distribution $\mathcal{N}(0, 1)$; and that $p(x|z)$ is a multivariate Gaussian distribution with a diagonal covariance structure $\mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$. Therefore, knowing $p(z)$ (**Prior**) and $p(x|z)$ (**Likelihood**) one could treat the problem of finding $p(z|x)$ (**Posterior**) as a classical Bayesian inference problem:

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)} \tag{1}$$

Where $p_\theta(x)$ is the **Evidence**, which can be expressed as $\int p_\theta(x|z)p_\theta(z)dz$ is a continuous formalism. However, for a high dimensional spaces this computation is intractable. Given this, we can use variational inference as a resource to obtain an approximate for the **Posterior**, that we will define as $q_\phi(z|x)$.

### A.3.2 Variational Inference

The goal is therefore to obtain $q_\phi(z|x) \approx p_\theta(z|x)$. Using the **variational inference** formalism, we do this through an optimization procedure, minimizing a metric that estimates the similarity between two distributions: the KL-Divergence.

$$D_{KL}(q_\phi||p_\theta) = \mathbb{E}_{q_\phi}\left[\log\frac{q_\phi(z|x)}{p_\theta(z|x)}\right] \tag{2}$$

However, in this form, the denominator has the intractable posterior $p_\theta(z|x)$. So we resort to some mathematical manipulations:

$$\begin{aligned}
&D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \\
&= \mathbb{E}_{q_\phi}\log\, q_\phi(z|x) - \mathbb{E}_{q_\phi}\log\, p_\theta(z|x) \\
&= \mathbb{E}_{q_\phi}\log\, q_\phi(z|x) - \mathbb{E}_{q_\phi}\left[\log\, \frac{p_\theta(z,x)}{p_\theta(x)}\right] \\
&= \mathbb{E}_{q_\phi}\log\, q_\phi(z|x) - \mathbb{E}_{q_\phi}\log\, p_\theta(z,x) + \mathbb{E}_{q_\phi}\log\, p_\theta(x) \\
&= \mathbb{E}_{q_\phi}\log\, q_\phi(z|x) - \mathbb{E}_{q_\phi}\log\, p_\theta(z,x) + \int q_\phi(z|x)\log\, p_\theta(x)dz \\
&= \mathbb{E}_{q_\phi}\log\, q_\phi(z|x) - \mathbb{E}_{q_\phi}\log\, p_\theta(z,x) + \log\, p_\theta(x)
\end{aligned} \tag{3}$$

Now, we can see that our equation presents the **Marginal Log Likelihood** (Log Evidence) $\log\, p_\theta(x)$. Defining the equation as a balance of this term: Now, we can see that now our equation presents the **Marginal Log Likelihood** (Log Evidence) $\log\, p_\theta(x)$. Defining the equation as a balance of this term:

$$\log\, p_\theta(x) = \underbrace{-\mathbb{E}_{q_\phi}\log\, q_\phi(z|x) + \mathbb{E}_{q_\phi}\log\, p_\theta(z,x)}_{\text{Component I}} + \underbrace{D_{KL}(q_\phi||p_\theta)}_{\text{KL-Divergence}} \tag{4}$$

And using the property that the KL-Divergence is never negative: $D_{KL}(q_\phi||p_\theta) \geq 0$

$$\log\, p_\theta(x) \geq \underbrace{-\mathbb{E}_{q_\phi}\log\, q_\phi(z|x) + \mathbb{E}_{q_\phi}\log\, p_\theta(z,x)}_{\text{Component I = Evidence Lower Bound}} \tag{5}$$

Therefore, with this formulation, we obtain an obligatory **Lower Bound** for our **Evidence** $p_\theta(x)$. And notably, as the Evidence Lower Bound is linked to the KL-Divergence,

by **maximizing** this term we will be **minimizing the KL-Divergence**; and thus, obtaining $q_\phi(z|x) \approx p_\theta(z|x)$.

So finally, we can the define the ELBO Loss function, used for VAE as:

$$\begin{aligned}
&\text{ELBO} = -\mathbb{E}_{q_\phi} \log\, q_\phi(z|x) + \mathbb{E}_{q_\phi} \log\, p_\theta(z, x) \\
&= -\mathbb{E}_{q_\phi} \log\, q_\phi(z|x) + \mathbb{E}_{q_\phi} \log\, p_\theta(x|z) + \mathbb{E}_{q_\phi} \log\, p_\theta(z) \\
&= \underbrace{\mathbb{E}_{q_\phi} \log\, p_\theta(x|z)}_{\text{Reconstruction Error}} \underbrace{- \mathbb{E}_{q_\phi} \left[ \log\, \frac{q_\phi(z|x)}{p_\theta(z)} \right]}_{\text{KL-Divergence}}
\end{aligned} \tag{6}$$

Therefore, the ELBO Loss - differently from the loss used for classic autoencoders - presents not only the expected reconstruction error, but also the KL-Divergence between the approximate posterior and the prior.

### A.3.3 Architecture

Now, with the aforementioned framework one can understand and implement a Variational Autoencoder. However, under a practical perspective, there is a problem when computing the ELBO loss with respect to the latent variable $z$: it is an stochastic variable, and therefore, cannot be backpropagated.

We can better analyze this problem starting with the ELBO loss function derived in the previous subsection:

$$\mathcal{L}_{\phi,\theta}(x) = \underbrace{\mathbb{E}_{q_\phi} \log\, p_\theta(x|z)}_{\text{Reconstruction Error}} \underbrace{- \mathbb{E}_{q_\phi} \left[ \log\, \frac{q_\phi(z|x)}{p_\theta(z)} \right]}_{\text{KL-Divergence}} \tag{7}$$

We note that the only term that we truly know for the moment is the **Prior** $p_\theta(z)$, which is defined as following a gaussian distribution $\mathcal{N}(0, \boldsymbol{I})$ (where $\theta = \{\mu = 0, \sigma = 1\}$, for $p_\theta(z)$). And developing the rest:

$$\begin{aligned}
\mathcal{L}_{\phi,\theta}(x) &= \underbrace{\mathbb{E}_{q_\phi} \log\, p_\theta(x|z)}_{\text{Reconstruction Error}} \underbrace{- \mathbb{E}_{q_\phi} \left[ \log\, \frac{q_\phi(z|x)}{p_\theta(z)} \right]}_{\text{KL-Divergence}} \\
&= \mathbb{E}_{q_\phi} \left[ \log\, p_\theta(x|z) + \log\, p_\theta(z) - \log\, q_\phi(z|x) \right] \\
&= \mathbb{E}_{q_\phi} \left[ \log\, p_\theta(x, z) - \log\, q_\phi(z|x) \right]
\end{aligned} \tag{8}$$

And now, as we want to minimize the loss, we most compute its gradient with respect to $\theta$:

$$\nabla_\theta \mathcal{L}_{\phi,\theta}(x) = \nabla_\theta (\mathbb{E}_{q_\phi} \left[ \log\, p_\theta(x, z) - \log\, q_\phi(z|x) \right]) \tag{9}$$

And using Leibniz Integral Rule, the fact that the second term does not depend on $\theta$, and computing the expectation approximately using one sample, we have:

$$\nabla_\theta \mathcal{L}_{\phi,\theta}(x) = \nabla_\theta \log\, p_\theta(x,z) \tag{10}$$

And with respect to $\phi$:

$$\nabla_\phi \mathcal{L}_{\phi,\theta}(x) = \underbrace{\nabla_\phi(\mathbb{E}_{q_\phi}}_{\text{Trouble}}[\log\, p_\theta(x,z) - \log\, q_\phi(z|x)]) \tag{11}$$

But now, we have a problem. The Leibniz Integral Rule can only be entertained as long as the support of the integral is not a function of a variable that is being derived. So we cannot interchange the gradient and the expectation as done before.

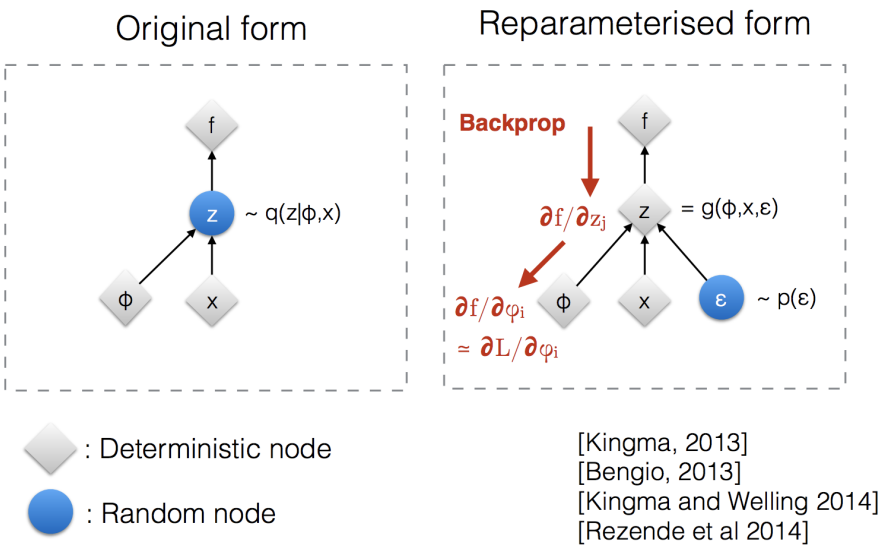



Figure 17: Illustration of how the reparameterization trick makes the sampling process trainable. [18]

In order to circumvent this problem, a transformation $\mathcal{T}_\phi$ can be defined as:

$$\begin{aligned} &\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) \\ &\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I}) \quad \text{; Change of Variables.} \end{aligned} \tag{12}$$

This transformation is also called **Location-Scale Transformation**, and this removes $\theta$ as the source of randomness! This is a first key idea for the implementation of Variational Autoencoders. The second key idea is the Law of The Unconscious Statistican (LOTUS). Let's suppose we have a random variable $X$ from the distribution $p_\theta(x)$; and so the expectation is:

$$\mathbb{E}_{p_\theta(x)}[X] = \int_{\mathbb{R}} x \cdot p_\theta(x) dx \tag{13}$$

And now, supposing we have a function $Y = f(X)$, so that $Y$ now follows the distribution $p_\phi(y)$; its expectation would be:

$$\mathbb{E}_{p_\phi(y)}[Y] = \int_{\mathbb{R}} y \cdot p_\phi(y) dy \tag{14}$$

However, the considering the dependence between $Y$ and $X$, the LOTUS gives us an interesting property about the behavior of expectations:

$$\mathbb{E}_{p_\phi(y)}[Y] = \mathbb{E}_{p_\theta(x)}[f(X)] = \int_{\mathbb{R}} f(x) \cdot p_\theta(x) dx \tag{15}$$

Therefore, to get the expectation of the function of a random variable, we can still use the base distribution! So, using these two key ideas we will be able to compute the expectation in the ELBO Loss in a fashion that the support of the integral will not be a function of $\theta$. Therefore, we will be able to use the Leibniz Integral Rule!

$$\nabla_\phi \mathcal{L}_{\phi,\theta}(x) = \mathbb{E}_{p(\epsilon)}(\nabla_\phi [\log\, p_\theta(x,z) - \log\, q_\phi(z|x)]) \tag{16}$$

This is the **Reparameterization Trick**: the use of a change of variables and the LOTUS property to enable the application of the Leibniz Integral Rule, so that the computation of the gradient can be done. This method existed before and is also known as Partwise Estimator or Pushing Gradient.

Given this, we can see the Variational Autoencoder in a framework that is much more similar to previous developments:

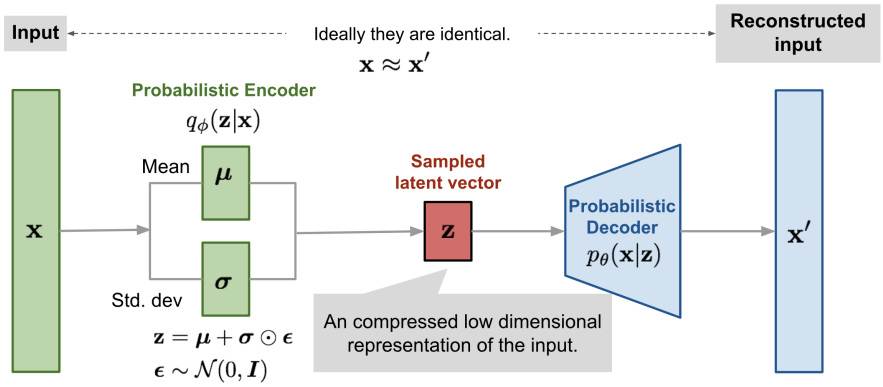



Figure 18: Illustration of variational autoencoder model with the multivariate Gaussian assumption. [18]

### A.3.4 Posterior Collapse

Posterior collapse is perhaps the most common issue presented by VAE architectures.

It describes the phenomenon in which the variational distribution $q_\phi(z_i|x)$ collapses towards the uninformative prior $p(z_i)$ for a subset $S$ of latent variables:

$$\exists i \in S \;\; s.t. \;\; \forall x \;\; q_\phi(z_i|x) \approx p(z_i) \tag{17}$$

This results in a generative model that ignores some variables of the latent space, reducing the generative power of the network. It has been shown that posterior collapse happens due to the $KL$ term on the $ELBO$ loss (refer to eq. 7), and its effect can be alleviated – though not eliminated – by the use of meta-heuristics. [19]

## A.4 Vector Quantized VAE (VQ-VAE)

While previous work on AEs and VAEs was focused on learning continuous representations of data on a continuous latent space, the Vector Quantized Variational AutoEncoder (VQ-VAE) architecture is motivated by the problem of unsupervised **discrete** representation learning [5]. This is of great interest for domains such as language – which is inherently discrete –, speech – which can be split into discrete symbols (phonemes) – and even images, since they can be described by language.

VQ-VAE differs from VAEs in two key points: the encoder network outputs discrete, rather than continuous, codes; and the prior is learned – while VAEs assumed the prior $p_\theta(z)$ to be a static centered gaussian, as seen on section A.3.1.

Also, the vector quantization method avoids the posterior collapse problem (see section A.3.4), observed on VAEs.

VQ-VAEs have been shown to generate high quality results in different media, such as images, videos and speech, showing the versatility of the architecture and the quality of the learned discrete representations obtained.

### A.4.1 Discrete Codebook

The general structure of VQ-VAEs is the same as that of VAEs, except for the fact that we replace the continuous latent space **Z** with what we call a **"Codebook"**. This new element is basically a list of vectors with numerical indices associated to each vector (thus, a Codebook is simply an Embedding layer just like the ones we use, for example, in NLP). This Codebook is used to **quantize** the latent space: the vector output of the encoder is compared to each vector in the Codebook and the closest vector in the Euclidean direction is then propagated to the decoder (hence the name "vector-quantized VAEs")

### A.4.2 Generating New Images: Learned Prior

With a VAE, one could create new images by sampling a point from the continuous latent space. On the VQ-VAE, however, it is non-trivial to do so: after all, the codes obtained from the training dataset follow a certain non-uniform prior. Indeed, if one were to generate images assuming a uniform prior for the codes, the results are meaningless:

To solve this problem, we'll have an accessory model (authors use a PixelCNN) to learn the training data's codes' prior. One could do so by encoding the training dataset, and then training the PixelCNN on the training codes.

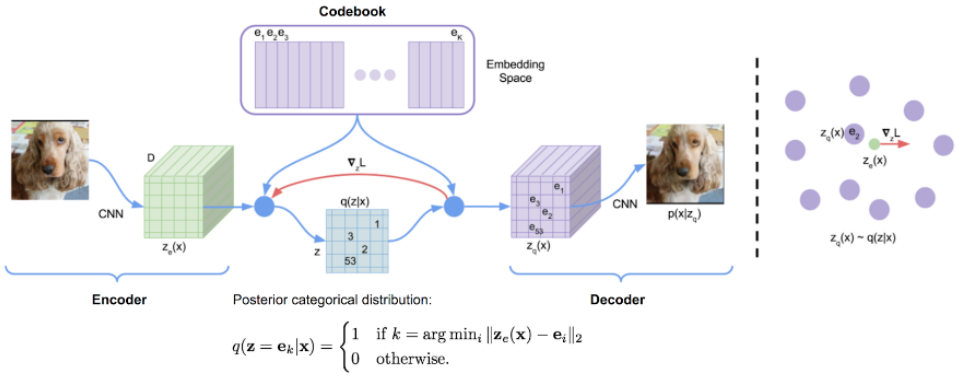



Figure 19: The architecture of VQ-VAE. [5]

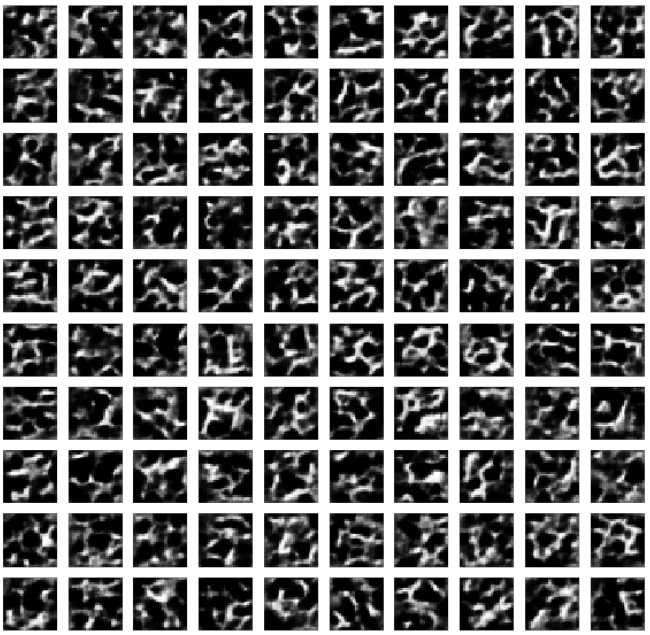


Figure 20: Results obtained on MNIST assuming a uniform prior for latent codes. [20]

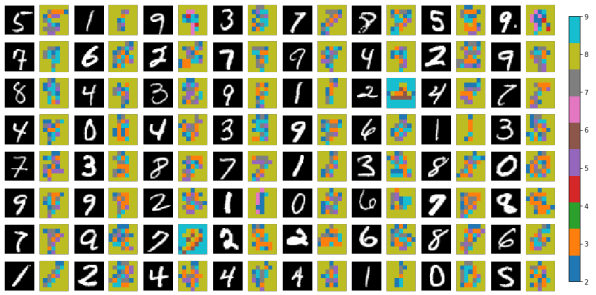


Figure 21: Training images and the respective latent codes, used to train the prior. [20]

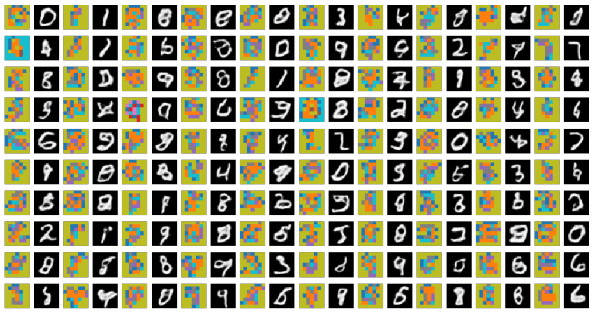


Figure 22: Codes generated from the trained PixelCNN and the respective resulting images, generated by decoding the latent codes. [20]

# B Domain Translation Architectures

## B.1 Introduction

As mentioned in the introduction, the text-to-image generation task is composed of two steps. The first, presented in section A consists of models with an architecture that allows the formation of a regularized latent space over training. In this way, the decoder component of these models can be used for the generation of new images, starting from points not previously traversed in our reduced dimensional space.

This section focuses on the second step: the text-to-image translation. The following discussion aims at presenting how to start from a textual prompt to the generation of an image corresponding to the caption description. To this end, we will first briefly introduce the transformer architecture that is commonly used in this task. This will be followed by a general presentation of different transformer types that can be used for this task. And finally, we will support our choice of architecture with a discussion about the inductive bias of this class of models.

## B.2 Transformer Architecture

The context in which the Transformer architecture emerged was in the area of NLP, notably based on the development of seq2seq (REF) models that made use of the attention mechanism (REF2). However, unlike the state-of-the-art algorithms developed so far, this architecture proposed an approach that was not based on recurrent network units. What made this breakthrough possible was a clever use of multi-head self-attention.

### B.2.1 Attention in the Transformer

The transformer views the encoded representation of the input as a set of key-value pairs, $(\mathbf{K}, \mathbf{V})$, both of dimension $n$ (input sequence length. Both the keys and values are the encoder hidden states. In the decoder, the previous output is compressed into a query ($\mathbf{Q}$ of dimension $m$) and the next output is produced by mapping this query and the set of keys and values.

Many attention mechanisms were suggested in recent years such as additive [REF3], location-based [REF4] and content-based [REF5] attention. The transformer adopts the scaled dot-product attention. The output is a weighted sum of the values, where the weight assigned to each value is determined by the dot-product of the query with all the keys:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{n}}\right)\mathbf{V} \tag{18}$$

The multi-head mechanism processes the scaled dot-product attention numerous times in parallel as opposed to merely computing it once. Simply concatenating and linearly

transforming the separate attention outputs into the expected dimensions. One can assume that the motivation behind this is because ensembling always helps.

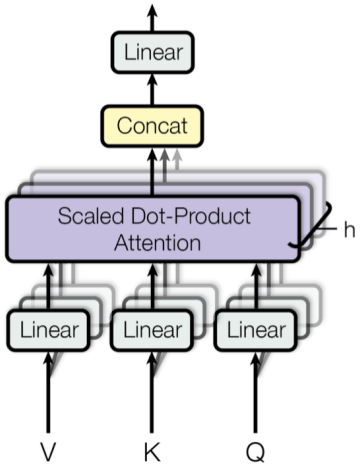
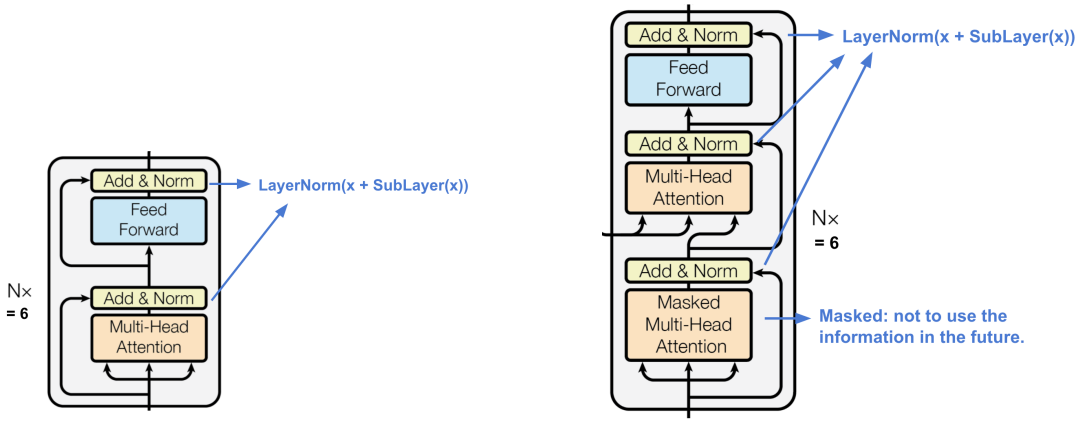



Figure 23: Multi-head scaled dot-product attention mechanism. (Fig 2 in Vaswani, et al., 2017)

### B.2.2 The Original Transformer

The original architecture is composed of an encoder and a decoder component. The encoder generates an attention-based representation with capability to locate a specific piece of information from a large context. It consists of a stack of 6 identity modules, each containing two submodules, a multi-head self-attention layer and a point-wise fully connected feed-forward network. By point-wise, it means that it applies the same linear transformation (with same weights) to each element in the sequence.

On the other hand, the goal of transformer decoder is to retrieve information from the encoded representation. The architecture is quite similar to the encoder, except that the decoder contains two multi-head attention submodules instead of one in each




(a) Transformer encoder component

(b) Transformer decoder component

Figure 24: Full transformer architecture proposed by Vaswani et al.

identical repeating module. The first multi-head attention submodule is masked to prevent positions from attending to the future.

### B.2.3 Transformer Variations

Taking into account the different tasks that can be addressed by models such as transformers, several variations of the original architecture have been proposed. Notably, many developments have been made with encoder only (autoencoder) and decoder only (autoregressive transformers) architectures; in addition to the models that followed with the Seq2Seq approach. Figure 25 shows a brief taxonomy of these recent developments.

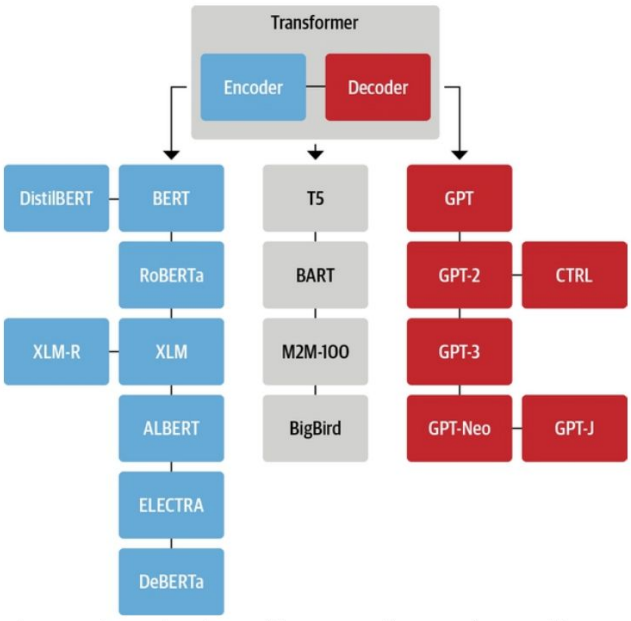



*Figure 3-8. An overview of some of the most prominent transformer architectures*

Figure 25: Taxonomy of prominent transformer architectures. [21]

First we will focus on encoder-only models. During training, the encoder takes input sequences and generates a fixed-size representation of the input, which is usually provided as a sequence of tokens, such as words or subwords, and each token is embedded into a high-dimensional vector space. These embedded tokens are then processed by multiple layers of self-attention and feed-forward neural networks to generate the final fixed-size representation. Additionally, encoder-only models tend to use the Masked Language Modeling (MLM) objective to randomly mask some tokens in the input sequence and train the model to predict the original masked tokens based on the context provided by the other unmasked tokens in the sequence. This encourages the model to learn bidirectional representations of the input sequence, as it has to consider both the left and right context of each masked token in order to predict it accurately.

On the other hand, decoder-only models generates a sequence of tokens autoregressively, where the output of the previous time step is used as input to the current time step.

This means that it is trained to predict the next token in a sequence given the context of the previous tokens. Furthermore, in the context of text generation, this is achieved by training the model to predict the probability distribution over the vocabulary of possible next tokens given the context of the previous tokens.

## B.3 Text-to-Image Translation

Now we will deal definitively with the problem of text-to-image translation. First, it is necessary to understand how many of the advances proposed for this task have arisen from developments in the field of NLP, focused purely on text translation.

With this in mind, we can think of the approaches suggested here as abstractions in the way of treating the image that bring it closer to the way a text translation task would be treated classically. At this point, we recover the concept of image tokens that makes up the codebook of the VQ-VAE architecture, from subsection A.4.

Taking as an example the translation proposed in the DALL-E paper [1], and based on a dataset of captions and corresponding images (COCO), the following process is proposed:

- The captions are encoded by Byte Pair Encoding (BPE), with dimension 256. The images, on the other hand, are encoded by the already trained VQ-VAE encoder, with final dimension 1024.
- The text and image tokens are coupled into a single vector, with special start, end and separation tokens between each sequence.
- This vector is fed to an autoregressive transformer (decoder-only), which is trained to predict the image tokens, given the text tokens.
- Once the transformer is trained, it is able to predict 1024 image tokens from any text.

These steps completely summarize the translation task. For image generation, we return to the VQ-VAE architecture and input into the decoder the 1024 image tokens produced by the autoregressive transformer. This way, we have a complete overview of how DALL-E works.

One might also wonder why a decoder-only model was chosen, and not a Seq2Seq model that is commonly used for translation tasks. In the literature it is possible to find other text-based image generation frameworks that propose the encoder-decoder approach, such as Parti, from Google [11].

While doing this work we were faced with different questions regarding the architecture of the translation transformer. What would be the ideal one? And why architectures that were not necessarily ideal worked well for the task? We propose answers to these questions

based on a discussion of the inductive biases of the transformer architecture as a whole, and of the differences in inductive biases between transformer models themselves.

## B.4 Inductive Bias

In the majority of machine learning tasks, our objective is to build a generalization based on a small group of observations (samples). Moreover, we want our generalization to hold true for brand-new, untested data. In other words, we want to infer a general rule from a small sample subset that applies to the entire population of samples.

As a result, we have certain observations and some assumptions that may be drawn from those findings. The collection of observations serves as our data, and the set of hypotheses consists of ML algorithms with all the potential learning parameters. Any model may explain training data, but it will provide noticeably different results when applied to fresh, unused data.

An inductive bias is the priority of some hypotheses (limitation of hypothesis space). So, the model favors a certain set of hypotheses. Based on some prior knowledge of the data, one can select a linear model for the preceding case and give linear generalization priority [22].

Therefore, selecting the appropriate induction bias for the model improves generalization, particularly in a low data environment. For the model to generalize successfully, the inductive bias should be larger the less training data we have. To prevent any induction bias, however, and allow the model to be less confined and freely explore the hypothesis space may be better in a rich data scenario.

Compared to CNNs, RNNs and MLPs; Transformers have no strong inductive biases, so they are more flexible and more data-hungry models. Absence of strong bias puts no additional constraints on a model. As a result, it can find better optimum if enough data is provided. The drawback is that such models perform worse in a low data setting.

The primary focus of transformer architecture is to allow efficient and holistic computation of inputs, without adding structures that make the model perform better in a given task. And a natural consequence of this fact is the ability of these models to be able to process massive amounts of data, operating them in parallel and allowing an intake of information that would be much more costly in other architectures [23].

Thus, both decoder-only approaches (like DALL-E) and Seq2Seq (like Parti) will allow a generalization of the objective function that corresponds to the expected behavior.

However, in an analysis restricted to the scope of the transformers family, it is notable that the Seq2Seq models naturally present inductive bias appropriate for translation tasks. Therefore, within the scope of the transformer architecture, they perform better than decoder-only models. This result was observed by Yu et al. [11]

# C State-of-the-Art Models

## C.1 Introduction

Parti and Imagen are complementary and explore two different families of generative models: diffusion models for Imagen and auto-regressive models for Parti.

## C.2 Imagen

Imagen is a text-to-image diffusion model, which combines transformer language models and diffusion models. It has several components.

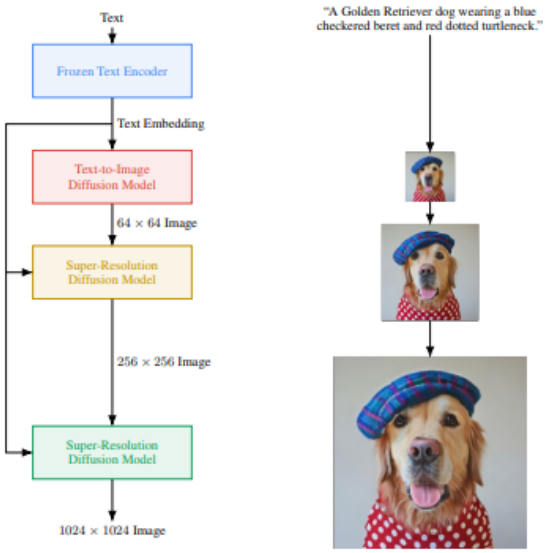



Figure 26: Imagen components

**Pre-trained Text Encoders**

Text-to-image models need powerful semantic text encoders to capture the complexity and compositionality of arbitrary natural language text inputs. Text encoders trained on paired image-text data are standard in current text-to-image models; they can be trained from scratch or pretrained on image-text data. The image-text training objectives suggest that these text encoders may encode visually semantic and meaningful representations especially relevant for the text-to-image generation task. Large language models can be another models of choice to encode text for text-to-image generation. Language models are trained on text only corpus significantly larger than paired image-text data, thus being exposed to a very rich and wide distribution of text. These models are also generally much larger than text encoders in current image-text models. It thus becomes natural to explore both families of text encoders for the text-to-image task. Imagen explores pretrained text encoders: BERT, T5 and CLIP.

**Diffusion models**

Diffusion models are generative models that convert a Gaussian noise into a sample of learned data distribution via an iterative denoising process.

**Classifier-free Guidance**

Classifier guidance is a technique to improve sample quality while reducing diversity in conditional diffusion models using gradients from a pretrained model during sampling. Classifier-free guidance is an alternative technique that avoids this pretrained model by instead jointly training a single diffusion model on conditional and unconditional objectives. In this setup, a generative model G is trained to be able to perform unconditional generation $G(z)$ (where $z$ represents random noise) and conditional generation $G(z, c)$ (where $c$ represents some condition, such as language descriptions). It is implemented as randomly dropping out the conditional vector (masking out or switching to a learned embedding) with some probability. During the inference process, sampling of an output I is done by using a linear combination of the unconditional and conditional predictions:

$$I = G(z) + \lambda(G(z, c)G(z)) \tag{19}$$

where $\lambda$ is a hyperparameter representing the weight of classifier-free guidance. Intuitively, it decreases the unconditional likelihood of the sample while increasing the conditional likelihood, which can be viewed as encouraging alignment between the generated sample and the text condition.

**Cascaded diffusion models**

Imagen utilizes a pipeline of a base 64×64 model, and two text-conditional super-resolution diffusion models to upsample a 64 × 64 generated image into a 256 × 256 image, and then to 1024 × 1024 image. Cascaded diffusion models with noise conditioning augmentation have been extremely effective in progressively generating high-fidelity images. Furthermore, making the super-resolution models aware of the amount of noise added, via noise level conditioning, significantly improves the sample quality and helps improving the robustness of the super-resolution models to handle artifacts generated by lower resolution models. Imagen uses noise conditioning augmentation for both the super-resolution models.

## C.3 Parti

Parti is a sequence-to-sequence model based on the Transformer architecture. It takes text tokens as inputs to an encoder and predicts discrete image tokens produced by a Transformer-based image tokenizer ( viT-VQGAN, better than VQ-VAE or VQ-GAN)

Parti is a Two-stage model, composed of an image tokenizer and an autoregressive model.

**Image Tokenizer**

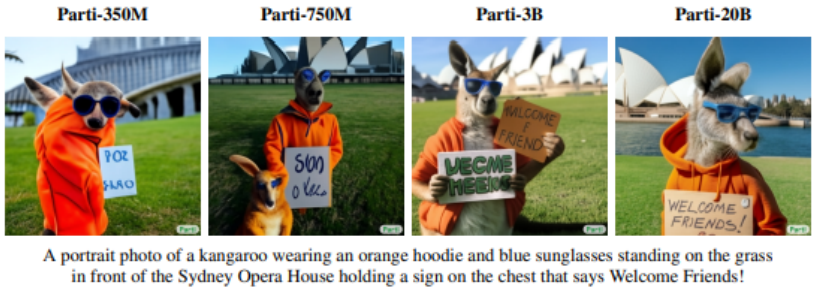



Figure 27: Parti components

Autoregressive text-to-image models must linearize 2D images into 1D sequences of patch representations. In the limit, these are just pixels, but this requires modeling very long sequences even for small images (e.g., a 256×256 × 3 RGB image leads to 196,608 rasterized values). Instead of learning representations that can take any value in the latent space, a visual codebook is learned that maps a patch embedding to its nearest codebook entry, which is a learned and indexable location in the latent space. These entries can be thought of as visual word types, and the appearance of any of these words in a patch in a given image is thus an image token.

To be most useful for the second stage model, the image tokenizer needs to learn an effective visual codebook that supports balanced usage of its entries across a broad range of images. It also must support reconstruction of a sequence of visual tokens as a high-quality output image. The ViT-VQGAN were trained with techniques including l2-normalization codes and factorized codes, which contribute to training stability, reconstruction quality and codebook usage.

Finally, while images of resolution 256×256 capture most of the contents, structures and textures, higher-resolution images have greater visual impact. To this end, a super-resolution module was put on top of the image tokenizer, shown in the picture below. It is learned with the same losses of ViT-VQGAN, mapping from reconstructed images to higher-resolution reconstructed images.

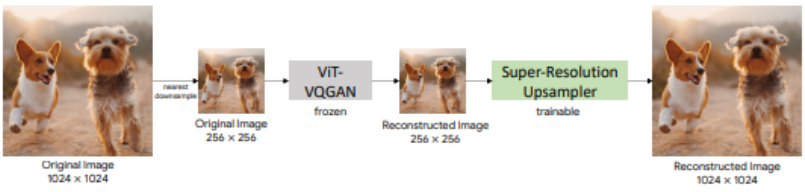



Figure 28

**Encoder-Decoder**

At the second stage, a standard encoder-decoder Transformer model is trained, by treating text-to-image as a sequence-to-sequence modeling problem. The model takes text as input and is trained using next-token prediction of rasterized image latent codes

generated from the first stage image tokenizer. For text encoding, a sentence-piece model of vocabulary size 16,000 was built on a sampled text corpus from the training data. Image tokens are produced by the learned ViT-VQGAN image tokenizer from the first stage. At inference time, the model samples image tokens autoregressively, which are later decoded into pixels using the ViT-VQGAN decoder.

**Classifier-free guidance**

As we saw for diffusion models with Imagen, Classifier-free guidance is critical in the context of improving the sample quality of diffusion models. In this context, it has been similarly applied in the context of autoregressive models for text-to-image generation to great effect. During inference, tokens are sampled from a linear combination of logits sampled from an unconditional model and a conditional model on a text prompt. CF-guidance was applied in Parti, and it has a significant improvement on the output image-text alignment, especially on challenging text prompts.

**Parti versions**

Four Parti versions were trained with data parallelism: 350M, 750M, 3B and 20B models; The results quality is increasing with the number of parameters.


A portrait photo of a kangaroo wearing an orange hoodie and blue sunglasses standing on the grass in front of the Sydney Opera House holding a sign on the chest that says Welcome Friends!

Figure 29

# References

[1] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[2] Arash Vahdat, Evgeny Andriyash, and William Macready. Dvae#: Discrete variational autoencoders with relaxed boltzmann priors. *Advances in Neural Information Processing Systems*, 31, 2018.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[7] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

[8] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[9] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[10] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

[11] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, Ben Hutchinson, Wei Han, Zarana Parekh, Xin Li, Han Zhang, Jason Baldridge, and Yonghui Wu. Scaling autoregressive models for content-rich text-to-image generation, 2022.

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[13] Boris Dayma, Suraj Patil, Pedro Cuenca, Khalid Saifullah, Tanishq Abraham, Phúc Lê Khc, Luke Melas, and Ritobrata Ghosh. Dall · e mini, 7 2021.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[15] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[16] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[17] Joseph Rocca and Baptiste Rocca. Understanding variational autoencoders (vaes), 2019.

[18] Lilian Weng. From autoencoder to beta-vae. *lilianweng.github.io*, 2018.

[19] James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. Understanding posterior collapse in generative latent variable models. 2019.

[20] Learning a prior over the latent space. `https://www.kaggle.com/code/ameroyer/keras-vq-vae-for-image-generation/notebook#Learning-a-prior-over-the-latent-space`. Accessed: 2022-11-16.

[21] Aman Chadha. Autoregressive vs. autoencoder models. *Distilled AI*, 2020. `https://aman.ai`.

[22] David Haussler. Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial intelligence*, 36(2):177–221, 1988.

[23] Eugene Kharitonov and Rahma Chaabouni. What they do when in doubt: a study of inductive biases in seq2seq learners. *arXiv preprint arXiv:2006.14953*, 2020.