Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Implementing OpenAI's DALL-E 2 From Scratch

Author:
Haopeng Luo

Supervisor:
Dr. Edward Johns

Second Marker:
Dr. Antoine Cully

June 21, 2023

Submitted in partial fulfillment of the requirements for the MEng Mathematics and Computer Science of Imperial College London

Abstract

We delve into the intricacies of OpenAI's DALL-E 2 (unCLIP) model, which represents a significant advancement in the realm of text-to-image synthesis. The motivation for this project stems from the lack of a complete guide or source code from OpenAI, leaving a gap in the literature that this work aims to fill. The objective is to demystify this complex model by providing a comprehensive guide to its architecture and implementation through experiments.

Despite the daunting task of reimplementing the original DALL-E 2 model, manifested by its 5.5 billion parameters and its extensive training data, we present a number of architectural modifications that make it feasible to train on commercial GPUs with less training data. The size of our model is reduced to 3% of the original DALL-E 2 model. We have conducted a series of experiments, including showing the effect of the classifier-free guidance scale on sample fidelity and diversity. We have also investigated the properties of the extensively used CLIP embeddings, finding that they encode intuitive and interpretable information of both images and texts. Our evaluations of the individual constituent models and the entire DALL-E 2 pipeline have shown that, despite occasional distortions and artefacts, our model generates plausible results most of the time and demonstrates good generalisability across different zero-shot generation tasks.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Edward Johns, for his invaluable guidance and constructive feedback throughout the course of this project. His expertise and insights have been instrumental in shaping this thesis, and his encouragement has motivated me to strive for excellence. I am also immensely grateful to my second marker, Dr. Antoine Cully, for his critical review and helpful suggestions for the interim report.

I would like to extend my heartfelt thanks to my family for their constant love, understanding, and encouragement. Their faith in my abilities and their unwavering support, even in the most challenging times, have been a source of strength and inspiration.

My sincere appreciation also goes to my friends, who have been a constant source of support and motivation throughout this journey. Their companionship and insightful discussions have made this journey more enjoyable and rewarding.

Finally, I would like to acknowledge all those who have contributed to this work in one way or another. Every constructive critique and word of encouragement has played a part in shaping this thesis.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In recent years, the field of artificial intelligence has witnessed significant advancements, particularly in the realm of deep generative models. These models, powered by deep learning techniques, have shown remarkable capabilities in generating realistic, high-quality data. They have been applied in various domains, including image synthesis, text generation, and more. Among the different types of deep generative models, diffusion models [1] have emerged as powerful paradigms, replacing previous generative models such as variational autoencoders (VAEs) [4] and generative adversarial networks (GANs) [5].

Diffusion models have been successfully applied in the field of text-to-image generation, where they have shown the ability to generate high-quality, realistic images from user-provided textual prompts [6; 7]. Recent state-of-the-art text-to-image models such as DALL-E 2 [3], Imagen [8] and Stable Diffusion [9] are all based on diffusion models. Diffusion models work by starting with a random noise image and gradually refining it over a series of steps, guided by a learned transition operation. The result is an image that closely matches the target distribution, whether it be natural images, paintings, or any other type of visual data.

The motivation for this project stems from DALL-E 2 [3], also known as unCLIP, developed by OpenAI. DALL-E 2 represents a significant advancement in the field of text-to-image synthesis, offering the ability to generate high-quality images from complex textual prompts. DALL-E 2 makes use of CLIP (Contrastive Language–Image Pretraining) [2], which is a contrastive model that connects images and texts in a unified latent embedding space, and it is trained on a large number of internet text and image pairs. CLIP has demonstrated impressive performance in zero-shot image classification tasks, where it can accurately classify images based on a set of textual captions that it has never seen during training.

With the success of DALL-E 2, the need for its reimplementation becomes apparent, as this can provide a deeper understanding of the model's functioning and can lead to discovery of improvements over the original model. In addition, it allows for customization and optimization of the model to suit specific needs, such as adapting the model to work with different datasets for downstream tasks or modifying it to run on specific hardware configurations.

However, reimplementing DALL-E 2 can be a daunting task, as the original model contains 6.5 billion parameters and is trained on approximately 250 million images [3]. Furthermore, while OpenAI has released some limited implementation details for DALL-E 2, they have not provided its source code nor a complete guide to implementing the model from scratch. This leaves a gap in the literature that this project aims to fill.

The objective of this project is to demystify the DALL-E 2 model by providing a detailed explanation of its architecture and implementation and by presenting experimental results. We aim to provide a comprehensive guide that covers all aspects of the implementation, from understanding the underlying theory to the practical details of coding the model. We hope that our project can serve as a useful resource for researchers and practitioners who are interested in understanding and using DALL-E 2 in their work to advance the field of text-to-image synthesis.

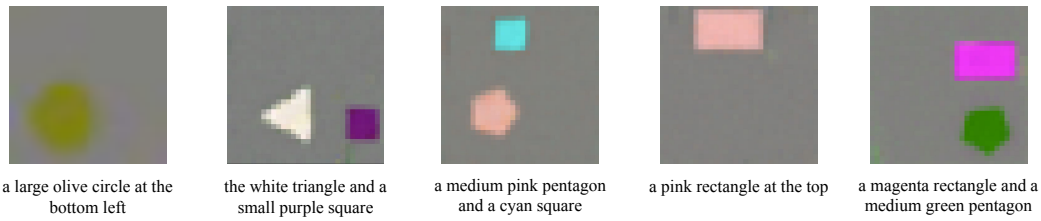


Figure 1.1: Some generated images using our DALL-E 2 model.

We believe the following are the main contributions of this project:

- We provide a detailed explanation of the DALL-E 2 model, including its architecture and the principles behind its operation. We present a number of architectural modifications to make DALL-E 2 trainable on commercial GPUs with less training data; the number of parameters in our model is only 3% of that in the original DALL-E 2 model. We also thoroughly discuss the training process, including the data requirements, the training procedure and the sampling procedure. We provide an open-source implementation of DALL-E 2 [1].

- We train our DALL-E 2 models on a customised dataset of simple geometric objects and present a series of experiments to explore the model's capabilities. Figure 1.1 shows some example generated images. We demonstrate the effect of the classifier-free guidance scale on the sample fidelity/diversity trade-off. We leverage the CLIP representation to study the CLIP latent space and to perform image variation and manipulation. We also explore text-guided image manipulation using the joint embedding space of CLIP. Finally, through experiments, we verify the hypothesis that the text captions retain more attribute-binding information than CLIP embeddings.

[1] Available at: https://github.com/RodgersLuo/open-dalle-2

- We conduct a series of quantitative and qualitative evaluations on our DALL-E 2 model to assess its performance and understand its limitations. Using a customised dataset, we demonstrate that a downsized DALL-E 2 model can produce plausible results in domain-specific tasks. Finally, we experiment and show that DALL-E 2 completes a number of zero-shot generation tasks.

This thesis is structured as follows. In chapter 2, we introduce the background and the necessary technical details behind DALL-E 2. In chapter 3, we present the architecture as well as the training and sampling process of DALL-E 2 in detail. In chapter 4, we conduct a series of experiments and evaluations of our DALL-E 2 model. And finally in chapter 5, we conclude this thesis.

Chapter 2

Background

The field of text-to-image generation has seen significant advancements over the past decade, with a variety of models and techniques being proposed to tackle this task. The goal of these models is to generate a realistic image that closely matches a given textual description, a task that requires a deep understanding of both the textual and visual domains.

The early approaches to text-to-image generation were based on Variational Autoencoders (VAEs) [4; 10] and Generative Adversarial Networks (GANs) [5], with GANs overtaking VAEs in terms of sample quality. In the context of text-to-image generation, they often use a conditioning variable, such as a text embedding, to guide the image generation process. Models like StackGAN [11] and AttnGAN [12] are examples of this approach, demonstrating the ability to generate high-quality images from textual descriptions.

However, GAN-based models have certain limitations. They often require complex training procedures and can suffer from issues such as mode collapse, where the model fails to capture the full diversity of the data distribution. Furthermore, GANs typically generate images in a single step, which can make it difficult to control the generation process and produce images that closely match the textual description.

Diffusion models [1] offer a different approach to text-to-image generation. Instead of generating an image in a single step, diffusion models start with a random noise image and gradually refine it over a series of steps, guided by a learned transition operation. This process allows diffusion models to generate high-quality images that closely match the target distribution, and their iterative nature can provide more control over the generation process.

Models such as DALL-E 2 (unCLIP) [3], Imagen [8] and Stable Diffusion [9] have demonstrated the potential of diffusion models in text-to-image generation by showing their ability to generate high-fidelity, diverse images from complex textual prompts.

We will be focusing on DALL-E 2. DALL-E 2 makes use of Contrastive Language-

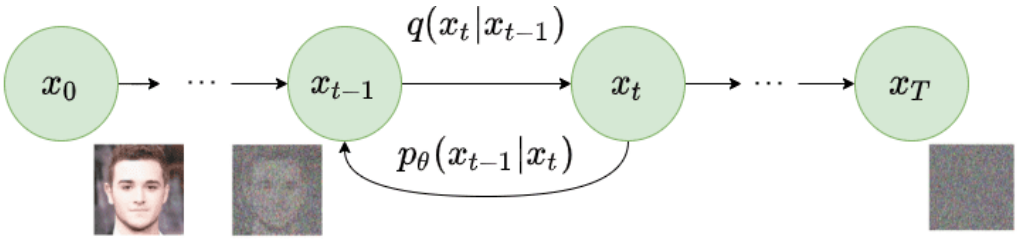


Figure 2.1: The diffusion and reverse process as presented by Ho et al. [1].

Image Pre-training (CLIP) [2], which is a method of pre-training a neural network to understand the relationship between language and images. These models make use of CLIP embeddings to bridge the gap between the textual and visual domains, allowing them to generate images that not only look realistic but also closely match the semantic content of the textual description. CLIP builds on the transformer architecture [13], which has been used in many state-of-the-art models such as BERT [14] and RoBERTa [15].

We begin this chapter by explaining the technical details of diffusion models, including some novel guidance methods for these models. Then, we introduce CLIP and describe the benefits of its shared embedding space. We conclude this chapter by introducing DALL-E 2 and explaining how CLIP embeddings are used.

2.1 Diffusion models

Recently diffusion models have emerged as the new state-of-the-art generative model in deep learning. They have broken the dominance of Generative Adversarial Networks (GANs) [5] in fields such as computer vision and image synthesis. Diffusion model works by progressively perturbing data with intensifying random noise (called the "diffusion" process), then successively remove noise to generate new data samples (called the "reverse" process). There are various formulations of diffusion models; in particular, we will examine the Denoising Diffusion Probabilistic Models (DDPM) [1] proposed by Ho et al..

2.1.1 Diffusion process

Figure 2.1 shows the diffusion process which is a Markov process. Given the input image x_0 sampled from the distribution $q(x)$, i.e. $x_o \sim q(x)$. At each step t of the Markov process, a Gaussian noise with variance β_t is added to the previous variable x_{t-1}. This produces a new variable x_t conditioned on x_{t-1} with distribution $q(x_t|x_{t-1})$. This can be formulated as

$$q(x_1, \ldots, x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.1}$$

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \mu_t = \sqrt{1-\beta_t}x_{t-1}, \Sigma_t = \beta_t \boldsymbol{I}) \tag{2.2}$$

x_t is also normally distributed with mean $\mu_t = \sqrt{1-\beta_t}x_{t-1}$ and variance $\Sigma_t = \beta_t \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix. If the total number of diffusion timesteps T is sufficiently large and the variance schedule β_t is well-behaved, the final image x_T will have an isotropic Gaussian distribution, i.e. $x_T \sim \mathcal{N}(0, \boldsymbol{I})$.

If we define $\alpha_t = 1 - \beta_t$ and the noises $\epsilon_0, \epsilon_1, \ldots, \epsilon_{t-1} \sim N(0, I)$. Then we can rewrite (2.2) as

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \tag{2.3}$$

Instead of applying q repeatedly to sample x_T, we can re-parameterise x_t in terms of x_0 by applying equation 2.3 recursively:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_0 \tag{2.4}$$

where $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. This closed-form equation enables us to perform one-step forward sampling of x_t for an arbitrary timestep t, drastically reducing the number of computations.

The variance parameter β_t can be varied. Ho et al. [1] used linearly increasing schedule. Nichol et al. [6] showed that using a cosine schedule achieves better performance.

2.1.2 Reverse process

We want to learn the reverse distribution $q(x_{t-1}|x_t)$, so that we can sample x_T from $N(0, I)$ and gradually reduce the noise in the sequence $x_{T-1}, \ldots, x_1, x_0$ during inference. x_0 would be a novel data point from the original data distribution. However, $q(x_{t-1}|x_t)$ is intractable in practice. We learn a model p_θ to approximate the reverse distribution (see Figure 2.1):

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{2.5}$$

where the mean $\mu_\theta(x_t, t)$ is approximated by a neural network. We can derive from equation (2.5) that

$$x_{t-1} = \mu_\theta(x_t, t) + \Sigma_\theta(x_t, t) \cdot z, \; z \sim \mathcal{N}(0, \boldsymbol{I}) \tag{2.6}$$

Although the reverse distribution $q(x_{t-1}|x_t)$ is intractable, if $\Sigma_\theta(x_t, t) = \tilde{\beta}_t$ [1], one can derive $q(x_{t-1}|x_t, x_0)$ using Bayes theorem:

$$q(x_{t-1}|x_t, x_0) = N(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t \boldsymbol{I}) \tag{2.7}$$

$$\tilde{\mu}_t(x_t, x_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t \tag{2.8}$$

$$\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{2.9}$$

We can optimise the following evidence lower bound (ELBO) to train the neural network p_θ:

$$L_{\text{ELBO}} = L_0 + L_T + \sum_{t=2}^{T} L_{t-1} \tag{2.10}$$

where

$$L_0 := -\log p_\theta\left(x_0|x_1\right) \tag{2.11}$$
$$L_{t-1} := D_{KL}\left(q\left(x_{t-1}|x_t, x_0\right) \parallel p_\theta\left(x_{t-1}|x_t\right)\right) \tag{2.12}$$
$$L_T := D_{KL}\left(q\left(x_T|x_0\right) \parallel p\left(x_T\right)\right) \tag{2.13}$$

Where the Kullback-Leibler (KL) Divergence $D_{KL}(P\|Q)$ measures how much the probability distribution P diverges from the reference distribution Q. Note that all KL divergences in equation (2.10) only involve Gaussian distributions, so they can be easily evaluated in closed-form equations.

Instead of predicting $\mu_\theta(x_t, t)$, i.e. the image mean directly, Ho et al. [1] also experimented with a model $\epsilon_\theta(x_t, t)$ that predicts the diffusion noise ϵ_0 from equation (2.4). The new loss is defined as follows:

$$L_{\text{simple}} = \mathop{\mathbb{E}}_{x_0,t,\epsilon}\left[||\epsilon - \epsilon_\theta(\sqrt{\bar{a}_t}x_0 + \sqrt{1-\bar{a}_t}\epsilon, t)||^2\right] \tag{2.14}$$

where $\epsilon \sim N(0, I)$. L_{simple} is the mean-squared error between the noise added to the initial image in the forward diffusion process and the noise predicted by the model in the reverse process at timestep t. The authors found that the simplified objective led to better sample quality at the cost of a lower log likelihood [1]. Using equation (2.4) and (2.8) we can derive

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) \tag{2.15}$$

However, L_{simple} provides no learning signals for the variance $\Sigma_\theta(x_t, t)$ in equation (2.5). Ho et al. [1] fixed the variance to constant values, e.g. $\tilde{\beta}_t$. To make the variance learnable with a neural network, Nichol and Dhariwal [6] proposed a hybrid loss $L_{\text{hybrid}} = L_{\text{simple}} + \lambda L_{\text{ELBO}}$, in which the variance is learned by optimising L_{ELBO} , and the authors showed that the new hybrid loss improves the log likelihood.

2.1.3 Conditional diffusion model

We introduced the standard diffusion model in the previous sections, where the evolution of the system over time is driven by a free-floating random process. In the context of image generation, the generated image will be a datapoint sampled from the entire data distribution; it is not possible to generate an image that corresponds to a specific class or text caption.

To address this issue, a conditional diffusion model could be employed. A conditional diffusion model is a type of generative model that uses the principles of

diffusion processes, but with a twist: it incorporates additional conditioning information to guide the generation process (reverse process). In contrast to the unconditional model $\epsilon_\theta(x_t, t)$, the conditional model is $\epsilon_\theta(x_t, t|y)$ where y is the conditional information. In image generation, the conditional information is usually a class label or text caption.

The core idea behind these models is to start from a simple initial Gaussian noise x_T, and then gradually transform this noise into an image x_0 that resembles the images from the target distribution. The transformation process is guided by a series of diffusion steps, which are designed to gradually shape the initial image into the target distribution. For example, if the goal is to generate images of a particular class, the diffusion steps are designed to gradually transform the initial noisy image into images of that class. Conditioning allows for a high degree of control over the image generation process. By changing the conditioning information, one can guide the model to generate specific types of images.

Supplementary guidance methods can also be applied alongside conditioning to improve sample quality. Two of the most prominent guidance methods are classifier guidance and classifier-free guidance, which we will discuss below.

Classifier guidance

Classifier guidance is a guidance method proposed by Dhariwal and Nichol [7] that uses a separate image classifier $p_\phi(y|x_t)$ to guide the reverse diffusion process. The idea is to use the gradients from the classifier to adjust the noise tensor at each step of the reverse process. The classifier is trained to predict the probability of the image x_t belonging to the class or corresponding to the text prompt y. During the reverse process, the noise tensor is adjusted in the direction that makes it more likely to be classified as belonging to the target distribution. At each step, a conditional diffusion model with mean $\mu_\theta(x_t, t|y)$ and variance $\Sigma_\theta(x_t, t|y)$ is perturbed by the gradient of the log-probability $\log p_\phi(y|x_t)$, yielding a new perturbed mean $\hat{\mu}_\theta(x_t, t|y)$:

$$\hat{\mu}_\theta(x_t, t|y) = \mu_\theta(x_t, t|y) + s \cdot \Sigma_\theta(x_t, t|y)\nabla_{x_t} \log p_\phi(y|x_t) \tag{2.16}$$

The coefficient s is the guidance scale, which determines how far the generation moves in the direction of the target class.

Classifier-free guidance

One disadvantage of classifier guidance is that a separate classifier needs to be trained and maintained. To address this issue, Ho and Salimans proposed classifier-free guidance [16], which does not rely on a separate classifier and uses a single model for conditioning and guidance. Classifier-free guidance employs a conditional diffusion model, $\epsilon_\theta(x_t, t|y)$. During training, the label/caption y is replaced by a null label/caption $\emptyset$ with a fixed probability. This results in two models: a conditional model $\epsilon_\theta(x_t, t|y)$ and an unconditional model $\epsilon_\theta(x_t, t|\emptyset)$. During predic-

tion, the guided output $\hat{\epsilon}_\theta(x_t, t|y)$ moves further in the direction of the conditional output and away from the unconditional output. The update is defined as follows:

$$\hat{\epsilon}_\theta(x_t, t|y) = \epsilon_\theta(x_t, t|\emptyset) + s \cdot (\epsilon_\theta(x_t, t|y) - \epsilon_\theta(x_t, t|\emptyset)) \tag{2.17}$$

where s is the classifier-free guidance scale. The effect of the guidance scale on the generated images is investigated in section 4.2.

Classifier-free guidance methods have been shown to produce more accurate and realistic samples than classifier guidance methods [17]. In addition, classifier-free guidance is simpler to implement: it requires just a single line of code modification during training, which involves random dropout of the conditioning, and during sampling, where the conditional and unconditional model outputs are mixed. Classifier guidance, in contrast, adds complexity to the training pipeline as it requires the training of an additional classifier. This classifier needs to be trained on noisy images, thereby making it unfeasible to incorporate a standard pre-trained classifier.

However, classifier-free guidance can be slower than classifier guidance during sampling. Generally, classifiers can be smaller and faster than generative diffusion models. During sampling, classifier guidance only requires one forward pass of the diffusion model and one forward pass of the classifier model while classifier-free guidance requires two forward passes of the diffusion model.

2.2 CLIP

One challenge commonly observed in conventional vision models is their difficulty in efficiently adapting to new tasks or domains that were not initially anticipated. Even models that demonstrate strong performance on standard benchmarks often struggle when subjected to stress testing [18]. To address this issue, Radford et al. [2] presented the Contrastive Language-Image Pretraining (CLIP) model. CLIP is a zero-shot model that learns a shared, multimodal latent representation of text captions and images. The term "zero-shot" means that the model is capable of predicting the most relevant text caption for an image given a set of text captions, even if the text captions are not observed during training.

Unlike standard image classification models that use a feature extraction network and a final linear classification network, CLIP uses an image encoder and a text encoder to obtain pairs of shared embeddings of images and texts in the latent space. In essence, CLIP tries to align image embedding vectors and the corresponding text embedding vectors while reducing the alignment of unassociated image-caption pairs during training.

2.2.1 Image encoder

Radford et al. [2] compared two different architectures to be used as image encoder for CLIP. For the first, ResNet-50 [19] is used as the base architecture for the

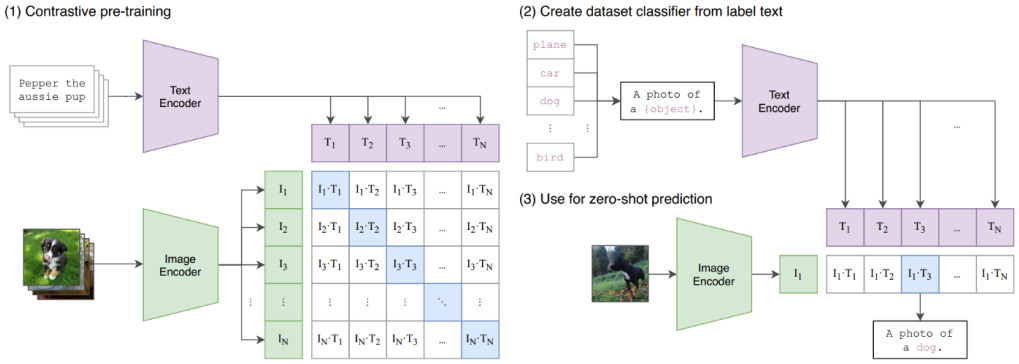


Figure 2.2: The approach of CLIP, as presented by Radford et al. [2]

image encoder. The base model is modified with the ResNet-D improvements [20]. The original global average pooling layer has also been replaced with an attention pooling layer implemented as a single layer of "transformer-style" multi-head QKV attention. For the second architecture, Vision Transformer [21] with some minor modifications is used as the base architecture for the image encoder. It is found that the ViT encoder achieved better performance compared to the ResNet encoder [2].

2.2.2 Text encoder

Transformer [13] is used as the text encoder for CLIP. The architecture is modified according to Radford et al. [22]. The authors used masked self-attention in the Transformer to preserve the ability to initialize the text encoder with a prior pre-trained language model [2].

2.2.3 Training

Contrastive pre-training is performed to train the CLIP model as shown in Figure 2.2. Given a batch of N (image, text) pairs, CLIP is trained to predict which of the $N \times N$ possible (image, text) pairings across a batch are actually correct. To this end, the image encoder first generates N corresponding image embedding vectors, $I_1, I_2, \ldots, I_N$; and the text encoder generates N corresponding text embedding vectors, $T_1, T_2, \ldots, T_N$. Note that the image embedding vectors and the text embedding vectors have the same dimensions. Then, an $N \times N$ matrix A of pairwise cosine similarity between every image embedding and text embedding is calculated. The goal is to maximise the cosine similarity along the diagonal of the matrix, while minimising the off-diagonal entries, i.e. to maximise A_{ii} for $i = 1, \ldots, N$ and minimise A_{ij} for $i \neq j$. The model uses the symmetric cross-entropy loss as its optimisation objective, which minimises in both the image-to-text direction and the text-to-image direction.

2.2.4 Zero-shot prediction

As shown in Figure 2.2, to perform zero-shot prediction, a set of possible text descriptions is provided to the learned text encoder to be encoded into text embeddings. Similarly, the input image is encoded into an image embedding. Finally, CLIP computes the cosine similarities between the image and text embeddings. The text caption of the pair with the highest similarity is then chosen as the final prediction. Note that the learned text encoder acts as a zero-shot linear classifier and the image encoder acts as a "feature extractor" in CNN image classification analogy.

2.3 DALL-E 2

OpenAI's DALL-E 2 [3] is a state-of-the-art text-to-image engine that is able to produce realistic images from text prompts, typically with high accuracy and high fidelity.

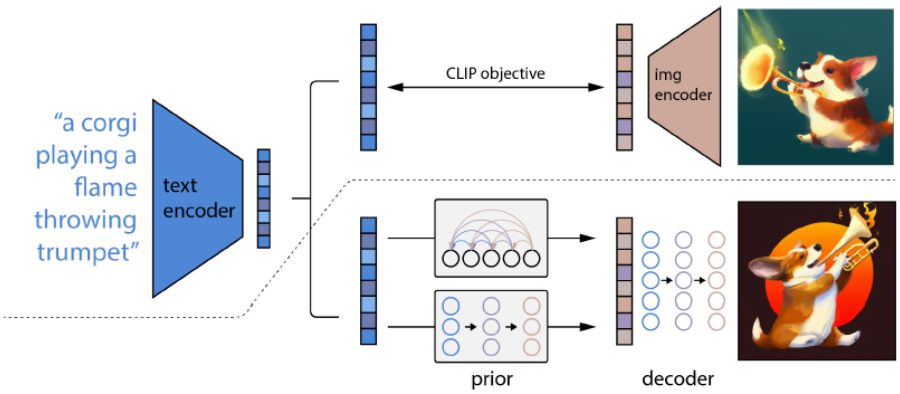


Figure 2.3: The architecture of DALL-E 2 as presented by Ramesh et al. [3].

DALL-E 2 combines CLIP[2] and diffusion models for the problem of text-conditional image generation. DALL-E 2 consists of 3 components: a CLIP model, a prior, and a decoder.

First, a CLIP model is trained to learn a joint representation space for images and texts. The CLIP model is depicted above the dotted line in Figure 2.3. After training, the weights of the CLIP model are frozen. Let g and h be the CLIP image encoder and text encoder respectively. Given an image-caption pair (x, y), let z_i and z_t be its CLIP image embedding and text embedding, i.e. $z_i = g(x)$ and $z_t = h(y)$.

The generative stack to produce images using text captions is (shown below the dotted line in Figure 2.3):

- A prior $P(z_i|y)$ produces CLIP image embeddings z_i conditioned on text captions y.
- A decoder $P(x|z_i, y)$ produces images x conditioned on CLIP image embeddings z_i and text captions y.

The CLIP image decoder reconstructs the image from its CLIP image embedding, while the prior generates the CLIP image embedding based on the text caption. Stacking these two components yields a generative model $P(x|y)$ that produces the image x given caption y:

$$P(x|z_i, y)P(z_i|y) = P(x, z_i|y) = P(x|y) \tag{2.18}$$

Note that the second equation holds because $z_i = g(x)$ is deterministic of x.

Finally the generated image x which is of dimension 64×64 is passed through two diffusion upsamplers, producing a final image of 1024×1024 resolution.

2.3.1 Decoder

A modified version of GLIDE (Guided Language to Image Diffusion for Generation and Editing) [17] is used as the decoder to produce images conditioned on CLIP image embeddings and text captions.

GLIDE is designed to generate high-quality images from textual descriptions, demonstrating a remarkable ability to translate complex textual prompts into corresponding visual representations. Leveraging the power of large-scale transformer models, GLIDE achieved text-to-image generation through training on a diverse range of internet text and images. This enables the model to handle a wide array of textual prompts and generate corresponding images with remarkable fidelity and diversity.

The architecture of GLIDE is based on the ablated diffusion model (ADM) [7]. The authors of the ADM model [7] found a better architecture for diffusion models through a series of ablation studies, which led to improvements in image synthesis over the original DDPM model [1]. However, the ADM model is only conditioned on class labels, the authors of GLIDE [17] made several modifications to condition the model on textual prompts.

The decoder of DALL-E 2 [3] is based on GLIDE but with further modifications to incorporate the CLIP embedding information. As a result, the decoder is conditioned on CLIP embeddings and optionally on text captions. The architecture of the decoder is described in more detail in section 3.3.

2.3.2 Prior

Although a decoder has the ability to invert CLIP image embeddings back into images, there is a need for a prior model that can generate these CLIP image

embeddings from text captions. Without the prior, DALL-E 2 [3] is essentially the same as GLIDE [17], which is conditioned on the textual prompts directly. Ramesh et al. [3] found that with the addition of the prior, the quality of DALL-E 2's generated samples is on par with those produced by GLIDE, with the added advantage of exhibiting a higher degree of diversity in the generations.

Ramesh et al. [3] experimented with an autoregressive prior and a diffusion prior, and concluded that the diffusion prior achieves better accuracy while being computationally more efficient. For the diffusion prior, the authors trained a decoder-only Transformer with a causal attention mask conditioned on the text descriptions. The implementation detail of our prior is described in section 3.4.

Chapter 3

Implementation

In this chapter, we explain the implementation details of our DALL-E 2 model. We first describe the dataset that is used to train the model. Then, we explain the main DALL-E 2 pipeline in terms of the architecture as well as the training/sampling process. We look at CLIP, decoder and prior in order.

3.1 Dataset

We created synthetic datasets of image-caption pairs featuring coloured geometric shapes. We use the built-in functions from the Python Image Library (PIL) to draw and generate the images. All of the images have a grey background. Due to the limitations of our computing resources, the images are 32×32 RGB images, so that they can fit into the GPU memory during training. There are 6 geometric shapes in total: rectangles, squares, triangles, circles, ellipses, and pentagons. Every shape has a colour and is of different sizes. In addition, pentagons and triangles are randomly rotated. The text captions that describe the images are automatically generated based on their shape, colour, size and optionally, their position in the image.

We created two datasets and trained two different models separately. For the first dataset, there are either one or two shapes in each image. We aim to use this dataset to study how DALL-E 2 [3] generates a combination of objects. The position of the shapes in images is not included in the text captions. The training set contains 20,000 images and the test set contains 2,000 images. Some random samples from the first dataset are shown in Figure 3.1. Unless otherwise stated, this dataset is our default dataset and the model trained on this dataset is our default model. The second dataset only contains one shape in every image, but the text caption describes where the shape locates in the image. We aim to use this dataset to study how DALL-E 2 recognises spatial information. The training set contains 25,000 images and the test set contains 2,000 images. Figure 3.2 shows some random samples and captions from the second dataset. Note that for both datasets, we excluded some combinations of colours and shapes in the training set in order to assess the zero-shot capability of our model.

We perform image preprocessing before training. The image is first resized to

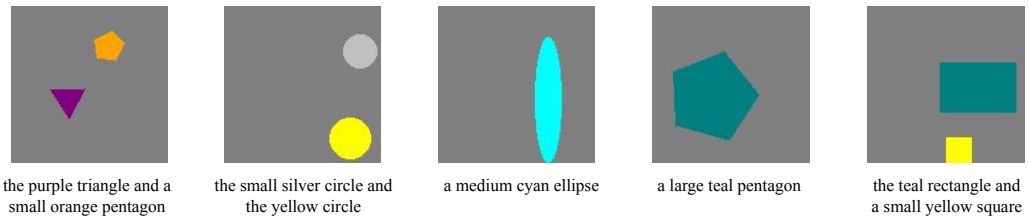


Figure 3.1: Some random samples and their text captions from the first dataset.

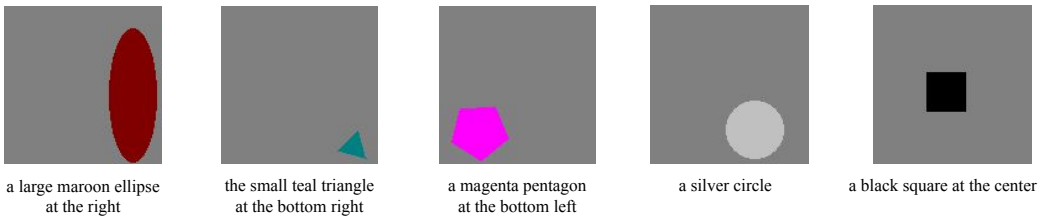


Figure 3.2: Some random samples and their text captions from the second dataset.

32×32. Then, the image is randomly flipped. Finally, following the original DDPM [1], the image is converted to a tensor and scaled between -1 and 1.

3.2 CLIP

In this section, we delve into the technical details and implementation of CLIP [2]. CLIP contains 2 separate models: a text encoder and an image encoder. In the following subsections, we introduce the text encoder and the image encoder in detail, as well as the training details of CLIP [1].

3.2.1 Text encoder

As the first preprocessing step, we pre-tokenise the text caption by white space and punctuation. We then perform lower-cased byte pair encoding (BPE) [23] to tokenise the words into smaller sub-words [2]. When applied to text tokenisation, BPE starts with a large text corpus and treats each character as a token. It then counts the frequency of all possible token pairs and merges the most frequent pair to form a new token. This process is repeated until the number of vocabulary is the same as the pre-defined vocabulary size, resulting in a vocabulary of tokens

[1] Our implementation is adapted from https://github.com/openai/CLIP
[2] We used the BPE implementation provided in the CLIP official repository.

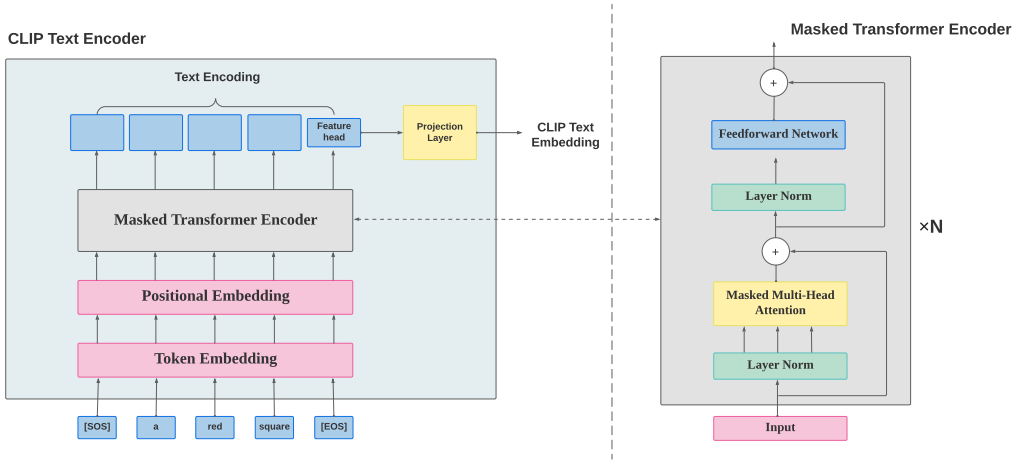


Figure 3.3: The architecture of our CLIP text encoder.

that can range from individual characters to common words or sub-words. One of the key advantages of BPE is that it can handle words that are not in its initial vocabulary by breaking them down into known sub-word units. This makes it particularly useful for dealing with rare and out-of-vocabulary words.

After tokenisation, the tokens are enclosed by the start-of-sentence token `[SOS]` and the end-of-sentence token `[EOS]`. The `[SOS]` token is prepended to the tokens, and the `[EOS]` token is appended to the tokens.

The text encoder of CLIP is an encoder-only transformer [13] with a causal attention mask [3]. The architecture of our CLIP text encoder is illustrated in Figure 3.3. First, the preprocessed tokens are mapped into a vector space that has the same dimension as the transformer's dimension. This is called the token embedding step. Then, a learned positional embedding is added to the token embeddings. Positional embeddings provide essential sequence position information, since the transformer model inherently lacks order awareness of words in the input sentence. These vectors, representing each token's position, are incorporated into the token embeddings prior to transformer input, enabling the model to discern inter-word relationships. Although the original transformer architecture uses a pre-defined sinusoidal positional embedding [13], we opt for a learned positional embedding similar to the ones in BERT [14] and RoBERTa [15] because of its simplicity and flexibility.

Then, the input tokens are passed into the transformer. The transformer has several layers. In each layer, there are two residual blocks: an attention block

[3]We modified and adapted the transformer implementation provided in the CLIP official repository.

followed by a feedforward block.

In the attention block, the first layer is a layer norm and the second layer is a masked multi-headed self-attention. Although there is no masking in the original transformer [13] encoder, the CLIP [2] authors added masking to preserve the possibility of integrating with a language model in the future. We decided to follow the approach adopted by the authors of CLIP [2] and use masked self-attention.

In the feedforward block, the first layer is also a layer norm. The second layer is a multi-layer perceptron with two linear layers and a GELU activation [24] in between. We use GELU as opposed to ReLU that is used in the original transformer [13] because GELU is differentiable everywhere and it is preferred by many recent models such as BERT [14].

In the output layer of the transformer, the token that corresponds to the `[EOS]` input token will be treated as the feature representation of the text caption. It will then be linearly projected into the CLIP embedding space to obtain the CLIP text embedding. The entire output sequence from the transformer is the text encoding and it will be fed into the DALL-E 2 prior network as input (section 3.4).

3.2.2 Image encoder

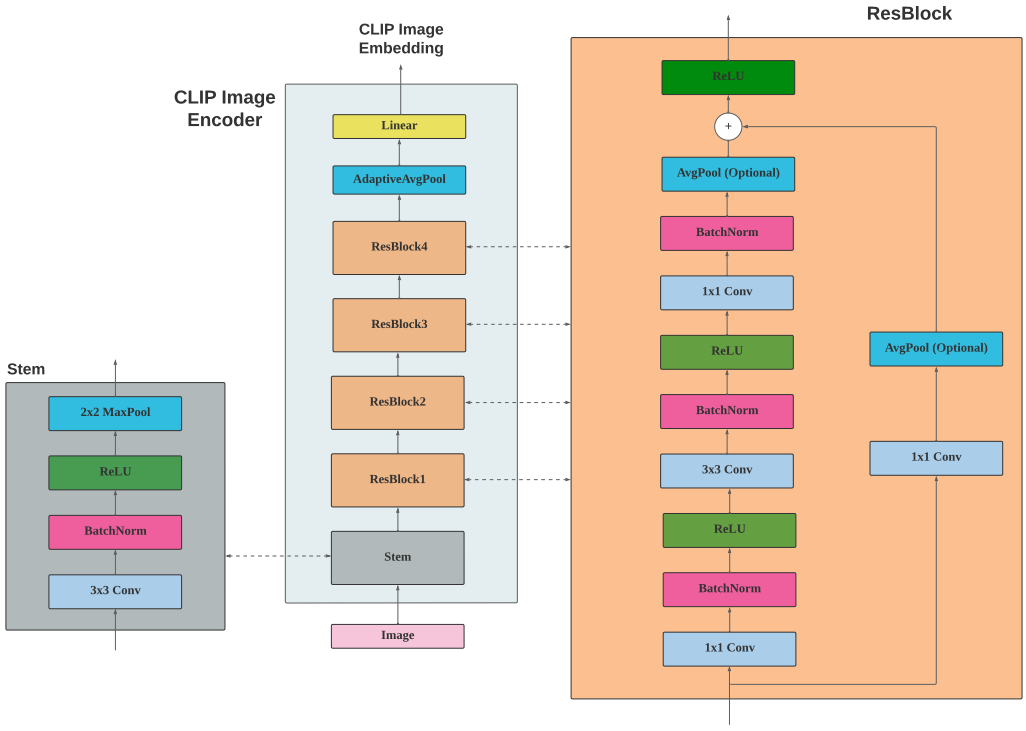


Figure 3.4: The architecture of our CLIP image encoder.

Radford et al. [2] compared ResNets [19] and Vision Transformers (ViTs) [21]

as CLIP image encoders; ViTs were found to achieve slightly better performance. However, the authors of CLIP trained the model on a large dataset of 400 million images [2]. It has been shown that ViTs perform better than ResNets when the dataset is large and perform worse on smaller datasets [21]. Since our dataset is relatively small, we decided to use ResNet as our CLIP image encoder.

In Figure 3.4, we illustrate the architecture of our CLIP image encoder. We built a smaller version of ResNet. The first block is a stem block, which is used to downsample the input image in order to accelerate computation further down the line. The downsampling in the stem block is carried out by the 2×2 max pooling layer.

Next, the feature maps are fed into 4 consecutive residual blocks. In each residual block, the feature maps go through a sequence of convolutional layers, normalisation layers and activation functions. There is also an optional average pooling layer for downsampling. We use a 1×1 convolution as the skip connection for the input as the shortcut. Residual connections take place before the final ReLU activation layer, where the input to the residual block is added to the output.

The output of the last residual block is downsampled to a fixed dimension by an adaptive pooling layer. Finally, the feature maps are flattened and passed into a linear layer. The linear layer projects the extracted feature maps into the space of CLIP image embeddings.

3.2.3 CLIP training

Algorithm 1 CLIP training procedure

Require: Dataset $\mathcal{D}$, temperature parameter τ, CLIP embedding dimension d

Initialise image encoder g_θ, text encoder h_θ
while loss L has not converged **do**
  $x, y \sim \mathcal{D}$ ▷ Sample a batch of n images and text captions
  $z_i \leftarrow g_\theta(x)$ ▷ Encode image, z_i has shape $n \times d$
  $z_t \leftarrow h_\theta(y)$ ▷ Encode text, z_t has shape $n \times d$
  Normalise z_i, z_t
  $S \leftarrow z_i \cdot z_t^T$ ▷ Compute the $n \times n$ pairwise cosine similarities
  $S \leftarrow S/\tau$
  $T \leftarrow (1, 2, \ldots, n)$ ▷ The ground truth labels
  $L_i \leftarrow \text{CrossEntropy}(S, T)$ ▷ Compute cross-entropy loss for images
  $L_t \leftarrow \text{CrossEntropy}(S^T, T)$ ▷ Compute cross-entropy loss for texts
  $L \leftarrow \frac{1}{2}(L_i + L_t)$ ▷ The symmetric loss function
  Take a gradient descent step on $\nabla_\theta L$
end while

The training of CLIP is crucial to the performance of DALL-E 2. Briefly described in section 2.2.3, a more detailed CLIP training procedure is listed in algorithm 1. We first obtain the image embeddings z_i and text embeddings z_t. The aim is to

make the associated pairs of image embeddings and text embeddings as similar as possible while making the non-associated pairs as dissimilar as possible. For every pair of (z_i, z_t) in a batch, we compute the cosine similarity:

$$\text{similarity}(z_i, z_t) = \cos(\theta) = \frac{z_i \cdot z_t}{\|z_i\|\|z_t\|} \tag{3.1}$$

We obtain an $n \times n$ pairwise cosine similarity matrix S, where $S_{i,j}$ represents the cosine similarity between the i-th image embedding and the j-th text embedding. We use two cross-entropy losses: L_i and L_t. L_i represents the loss in the image direction; that is, for the i-th image, we want to increase the likelihood of the corresponding i-th text caption while reducing the likelihood of all other text captions. On the other hand, L_t represents the loss in the text direction. We optimise a contrastive symmetric loss function that is the average of L_i and L_t. Note that in line 8 in algorithm 1, we scale the similarity matrix by a learnable temperature parameter τ to control the range of the cosine similarities. We follow the original CLIP approach [2], by initialising τ to 0.07 and clamping it to prevent training instability.

3.3 Decoder

As discussed in section 2.3.1, the DALL-E 2 decoder $P(x|z_i, y)$ is used to generate images x conditioned on CLIP image embeddings z_i and text captions y. In this section, we will first present our decoder architecture. Then, we will describe the training process and the sampling process.

3.3.1 Architecture

For DALL-E 2 decoder, Ramesh et al. [3] modified the GLIDE [17] architecture by incorporating CLIP embeddings. GLIDE [17] adopted the ADM model [7] architecture but made slight modifications to condition the model on text captions, yielding a model $P(x|y)$.

Ho et al. used a U-Net backbone [25] with group normalisation throughout [26] to represent the reverse process of the original DDPM [1]. Subsequent models such as ADM [7], GLIDE [17] and DALL-E 2 [3] are all based on the U-Net architecture. We therefore also adopt the U-Net architecture for our decoder model. The architecture of U-Net is characterized by its U-shaped design, which consists of a contracting path (encoder) and an expansive path (decoder). The encoder captures the context in the image, while the decoder enables precise localization using transposed convolutions. Furthermore, U-Net employs skip connections between the encoder and decoder to retain high-resolution features lost during downsampling.

Our U-Net architecture is illustrated in Figure 3.5, which is composed of several

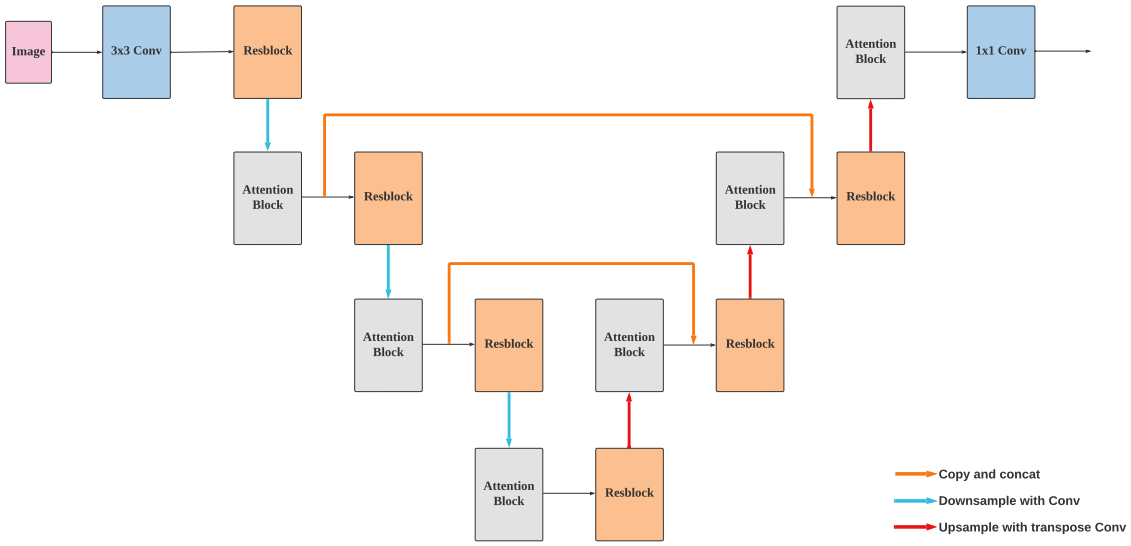


Figure 3.5: The architecture of our decoder U-Net

residual blocks and self-attention [13] blocks [4]. Residual blocks are followed by a downsampling convolution in the U-Net encoder and an upsampling transpose convolution in the U-Net decoder. The output of the residual blocks in the encoder is copied and concatenated to the input of the corresponding residual blocks in the decoder. We decided to incorporate self-attention blocks at every U-Net resolution because they are found to result in a boost in sample quality [7].

Besides the input noisy image x_t, the U-Net also takes in additional information for conditioning, which includes timestep embeddings and text encodings. In Figure 3.6, we demonstrate how the conditioning information is obtained. We explain this process in detail below.

- We need to condition the U-Net model on the diffusion timestep t that comes along with the noisy image x_t, otherwise the model would not have any knowledge of the current diffusion timestep that the image is taken from. The diffusion timestep t is a scalar in $[0, T)$ where T is the total number of diffusion timesteps. We use a sinusoidal position embedding [13] to embed the timestep t:

$$\text{PE}(t, 2i) = \sin(t/10000^{2i/d}) \tag{3.2}$$

$$\text{PE}(t, 2i+1) = \cos(t/10000^{2i/d}) \tag{3.3}$$

[4] Our decoder implementation is inspired by https://github.com/hojonathanho/diffusion

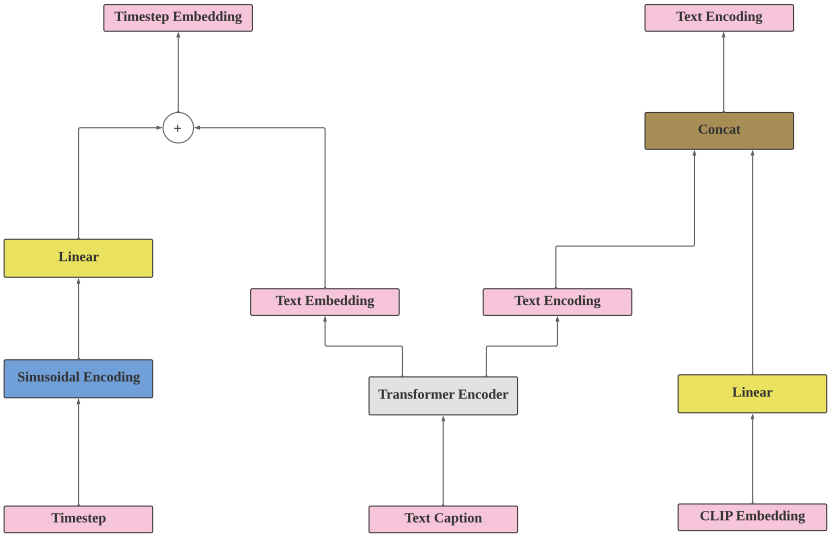


Figure 3.6: The additional conditioning information

where d is the output dimension. The output from the sinusoidal position embedding is then linearly projected to obtain the timestep embedding.

- Following the approach from GLIDE [17], we use a transformer encoder to encode the text caption y. Note that this transformer is different from the one used in the CLIP text encoder. The last token embedding from the output of the transformer is projected and added to the existing timestep embedding. Moreover, the entire sequence of token embeddings from the transformer output layer is taken as the text encoding.

- We follow the approach in [3], in which the CLIP image embedding z_i is linearly projected into four extra token embeddings and concatenated to the output sequence (text encoding) from the transformer encoder. The authors also projected and added the CLIP image embedding to the timestep embedding. We conducted an ablation study and found that sample quality deteriorates as we add the CLIP embedding to the timestep embedding. So, we decided not to follow this approach.

Now, we introduce the residual block and the attention block in detail and explain how the conditioning information is used. In Figure 3.7, we present the architecture of a residual block in our U-Net.

The U-Net residual block takes in the image feature map and the timestep embedding as input. The stem of the residual layer is made up of convolutional layers, group normalisation [26] layers and SiLU [27] activations. The timestep embedding first passes through a linear layer and a SiLU activation function. Then, it is added to the image feature maps after the first SiLU activation in the stem. There is a final upsampling/downsampling layer depending on whether the resid-

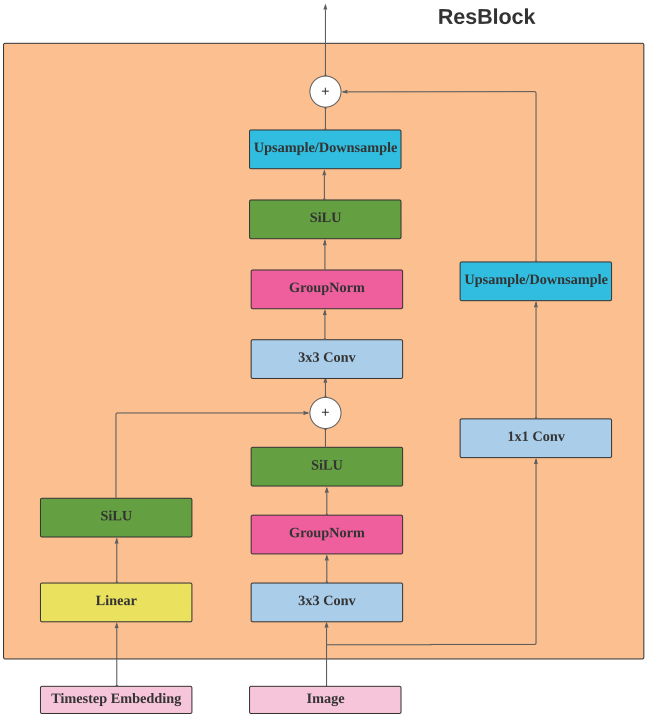


Figure 3.7: U-Net residual block

ual block is in the U-Net decoder or encoder. The upsampling layer is a transpose convolutional layer and the downsampling layer is a normal convolutional layer. Finally, the input of the residual block is added to the output before going through a skip connection shortcut.

Figure 3.8 shows the architecture of an attention block in our U-Net [5]. The attention block takes in the image feature map and the text encoding as input. The text encoding is projected to the dimensionality of the attention layer and then concatenated to the normalised image feature maps. Next, the feature maps are fed into a multi-headed attention layer and a final 1×1 convolutional output projection layer. Finally, the input feature maps are added to the output for residual connections.

Note that the original authors of DALL-E 2 [3] also used two final upsamplers to increase the resolution of the image. We decided not to include the upsampler since that is not the main focus of this project.

[5]We used the implementation of the attention block from https://github.com/openai/improved-diffusion/blob/783b6740edb79fdb7d063250db2c51cc9545dcd1/improved_diffusion/unet.py#L200

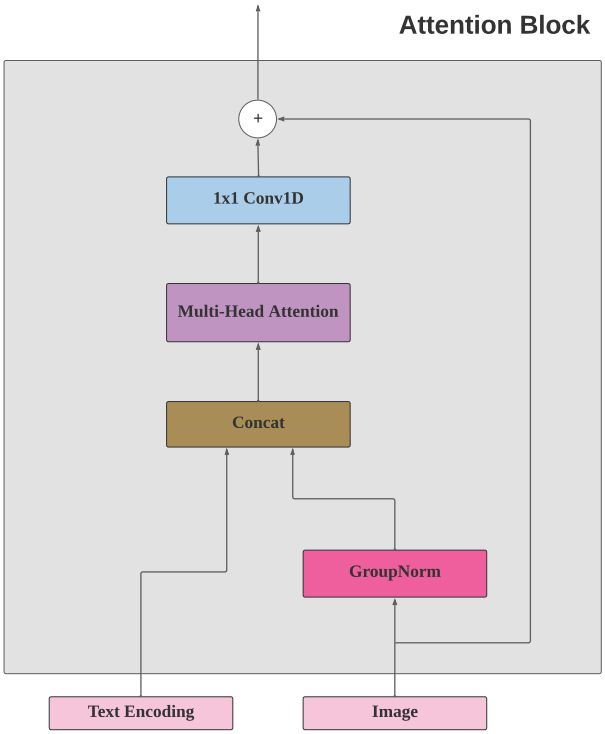


Figure 3.8: U-Net attention block

3.3.2 Training

In algorithm 2, we describe the decoder training procedure. Instead of predicting the original image x_0 directly, our decoder learns to predict the noise ϵ that is added in the forward diffusion process as discussed in section 2.1.2. The decoder is conditioned on both the text caption y and the CLIP image embedding z_i as discussed in the previous subsection, i.e. the decoder is $\epsilon_\theta(x_t, t|y, z_i)$.

During training, a batch of different diffusion timesteps t is randomly sampled and the CLIP image embedding z_i is obtained by encoding the original training image x_0 with the CLIP image encoder. We randomly drop text captions and CLIP embeddings as shown in line 7 to train the unconditional model. This is necessary for classifier-free guidance (section 2.1.3) which Ramesh et al. [3] adopted in the original DALL-E 2 decoder. Ramesh et al. [3] dropped text captions and CLIP embeddings separately by setting $p_t = 0.5$ and $p_c = 0.1$, which essentially trains 3 different unconditional models: $\epsilon_\theta(x_t, t|\emptyset, z_i)$, $\epsilon_\theta(x_t, t|y, \emptyset)$ and $\epsilon_\theta(x_t, t|\emptyset, \emptyset)$. However, we decided to only train a single unconditional model $\epsilon_\theta(x_t, t|\emptyset, \emptyset)$ by dropping text captions and CLIP embeddings simultaneously with $p_t = p_c = 0.2$ as we found that this approach led to better sample quality.

We then sample a random normal noise and perform the forward diffusion process (equation 2.4) as shown in line 9. This yields a noisy image x_t at diffusion

timestep t. We experimented with two different variance schedules β_t. The first one is a linear increasing schedule [1], where $\beta_0 = 10^{-4}$ nad $\beta_T = 0.02$. The second one is a cosine schedule [6]:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2 \tag{3.4}$$

And $\beta_t = 1 - \bar{\alpha}_t/\bar{\alpha}_{t-1}$. We decided to use the cosine variance schedule as we discovered that it resulted in better performance. Next, we use the U-Net to predict the noise $\hat{\epsilon}$ that was added to the original image x_0, conditioning on x_t, t, y and z_i. The loss is the mean-squared error (L_{simple} in equation 2.14) between the ground truth noise and the predicted noise.

Algorithm 2 Decoder training procedure

Require: Dataset $\mathcal{D}$, CLIP image encoder g, CLIP embedding drop probability p_c, text caption drop probability p_t, diffusion schedules $\beta_t, \alpha_t, \bar{\alpha}_t$

1: Initialise ϵ_θ
2: **while** loss L has not converged **do**
3: $x_0, y \sim \mathcal{D}$ ▷ Sample a batch of n images and text captions
4: $t \sim$ discrete uniform$(0, T)$ ▷ Sample a batch of random timesteps
5: $z_i \leftarrow g(x_0)$ ▷ Encode image, z_i has shape $n \times d$
6: Normalise z_i
7: Randomly set $y^{(j)}, z_i^{(j)}$ to 0 with probability p_t, p_c
8: Sample true noise $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$
9: $x_t \leftarrow \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$ ▷ Forward diffusion process
10: $\hat{\epsilon} \leftarrow \epsilon_\theta(x_t, t | y, z_i)$ ▷ Predict the diffusion noise
11: $L \leftarrow \|\epsilon - \hat{\epsilon}\|^2$
12: Take a gradient descent step on $\nabla_\theta L$
13: **end while**

3.3.3 Sampling

In algorithm 3, we present our decoder sampling procedure $P(x|z_i, y)$. Following Ho et al. [1], we fix the posterior variance $\Sigma_\theta(x_t, t) = \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$. We first sample a random noisy image x_T. Then, we perform the reverse diffusion process $p_\theta(x_{t-1}|x_t)$ as described in section 2.1.2. In line 6, we employ classifier-free guidance (section 2.1.3) to obtain the predicted diffusion noise $\hat{\epsilon}$. Line 7 implements equation (2.15), in which the image mean is obtained by removing the added noise from the noisy image. We obtain the noisy image at the previous timestep x_{t-1} by adding the product of a random noise and the posterior variance to the image mean as shown in equation (2.6). Note that we need to clamp the pixel values in x_{t-1}, otherwise the pixel values could exceed the range of the values in the original image distribution.

Algorithm 3 Decoder sampling procedure

Require: model ϵ_θ, text caption y, CLIP image embedding z_i, guidance scale s, diffusion schedules $\beta_t, \alpha_t, \bar{\alpha}_t, \tilde{\beta}_t$

$\Sigma_\theta(x_t, t) \leftarrow \tilde{\beta}_t$ ▷ Fix the posterior variance
$x_T \sim \mathcal{N}(0, \boldsymbol{I})$ ▷ Sample a random latent image
for $t = T, \ldots, 1$ **do**
  $\hat{\epsilon}_{\text{cond}} \leftarrow \epsilon_\theta(x_t, t | y, z_i)$ ▷ Conditional noise prediction
  $\hat{\epsilon}_{\text{uncon}} \leftarrow \epsilon_\theta(x_t, t | \emptyset, \emptyset)$ ▷ Unconditional noise prediction
  $\hat{\epsilon} \leftarrow \hat{\epsilon}_{\text{uncon}} + s \cdot (\hat{\epsilon}_{\text{cond}} - \hat{\epsilon}_{\text{uncon}})$ ▷ Classifier-free guidance
  $\mu(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\hat{\epsilon})$
  if $t > 1$ **then** ▷ Add posterior noise
    $\tilde{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
    $x_{t-1} \leftarrow \mu(x_t, t) + \Sigma_\theta(x_t, t) \cdot \tilde{\epsilon}$
  else
    $x_0 \leftarrow \mu(x_t, t)$
  end if
  Clamp x_{t-1} between -1 and 1
end for
return x_0

3.4 Prior

As discussed in section 2.3.2, the DALL-E 2 prior produces CLIP image embeddings z_i from text captions y. We will present the architecture, training procedure and sampling procedure of our prior in this section.

3.4.1 Architecture

Ramesh et al. [3] explored an autoregressive prior and a diffusion prior, and they found that the diffusion prior produces higher-quality samples while remaining more computationally efficient. Therefore, we decided to also use a diffusion model for our prior.

In Figure 3.9, we present the architecture of our prior model. We use a decoder-only transformer [13] with a causal attention mask to represent the reverse diffusion process. The inputs to the transformer, in order, are:

- The CLIP text embedding z_t, which is obtained by encoding the text caption y with the trained CLIP text encoder. The text embedding is then linearly projected to a number of token embeddings as input of the transformer.
- The CLIP text encoding c (see section 3.2.1), which is the entire output sequence from the CLIP text encoder. It has the shape $b \times n \times d$, where b is the batch size, n is the maximum number of output tokens from the CLIP encoder, and d is the dimension of the transformer in the CLIP text encoder. We first reshape the text encoding to $b \times d \times n$, and then use a 1D convolution with a kernel size of 1 and an output channel of 1 to embedding-wise

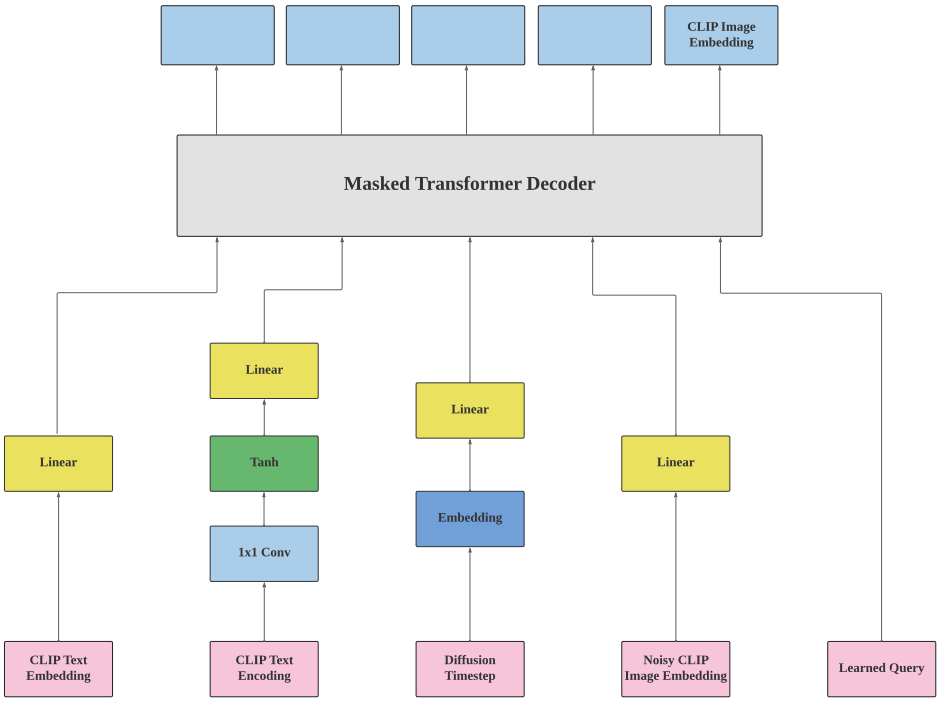


Figure 3.9: The architecture of our prior model

pooling. This results in a tensor of shape $b \times 1 \times n$. Finally, this tensor passes through an activation function and is linearly projected to a number of token embeddings for the prior transformer.

- The diffusion timestep t. The timestep first goes through an embedding layer to be mapped to the dimensionality of the transformer inputs. It is then linearly projected to a number of input token embeddings.
- The noisy CLIP image embedding $z_i^{(t)}$ at timestep t. It is also linearly projected to several input token embeddings.
- A final learned query, which is a learnable parameter and has dimension d_{model}, the dimension of the prior transformer. The output of the token embedding that corresponds to the learned query is the predicted CLIP image embedding.

The sequence of the inputs described above is then concatenated and passed into the transformer decoder. We use a causal attention mask in the self-attention mechanism of the transformer model [13], effectively masking the influence of future inputs by setting their attention scores to negative infinity, which becomes zero after applying a softmax function. This ensures that the model cannot attend to future positions in the input sequence, enforcing a form of causality in the model's predictions. The output token embedding that corresponds to the learned query (the last token embedding in the output sequence) is taken as the predicted CLIP unnoised image embedding $z_i^{(0)}$.

3.4.2 Training

Algorithm 4 Prior training procedure

Require: Dataset $\mathcal{D}$, CLIP image encoder g, CLIP text encoder h, diffusion schedules $\beta_t, \alpha_t, \bar{\alpha}_t$

1: Initialise the prior model f_θ
2: **while** loss L has not converged **do**
3: $x, y \sim \mathcal{D}$ ▷ Sample a batch of n images and text captions
4: $t \sim$ discrete uniform$(0, T)$ ▷ Sample a batch of random timesteps
5: $z_i^{(0)} \leftarrow g(x)$ ▷ Encode image, $z_i^{(0)}$ has shape $n \times d$
6: $z_t, c \leftarrow h(y)$ ▷ Obtain text embeddings and text encodings
7: Normalise $z_i^{(0)}, z_t$
8: Sample forward noise $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$
9: $z_i^{(t)} \leftarrow \sqrt{\bar{\alpha}_t} z_i^{(0)} + \sqrt{1 - \bar{\alpha}_t}\epsilon$ ▷ Forward diffusion process
10: $\hat{z}_i \leftarrow f_\theta(z_i^{(t)}, t | z_t, c)$ ▷ Predict the image embedding
11: $L \leftarrow \|\hat{z}_i - z_i^{(0)}\|^2$
12: Take a gradient descent step on $\nabla_\theta L$
13: **end while**

In algorithm 4, we present the training procedure of the prior model $f_\theta(z_i^{(t)}, t|z_t, c)$. Unlike our decoder which predicts the diffusion noise ϵ, we follow [3] and make our prior model predict the CLIP image embedding z_i directly. In the training loop, we obtain the text embedding and the corresponding ground truth image embedding using our CLIP encoders. We also obtain the text encodings. To improve efficiency, we cache the CLIP embeddings as well as the text encodings during training. Then, in line 9, we perform the forward diffusion process on the ground truth image embedding $z_i^{(0)}$ to obtain the noisy image embedding $z_i^{(t)}$ at timestep t. Next, we predict the image embedding $\hat{z}_i$ using the prior network, conditioned on the noisy image embedding, diffusion timestep, text embedding and text encoding. The loss is simply the mean-squared error between the predicted image embedding and the ground truth image embedding. Note that for our prior, we did not employ classifier-free guidance [16] as we did in our decoder model (section 3.3). As there is little practical use for our prior model to generate CLIP image embeddings unconditionally.

3.4.3 Sampling

Algorithm 5 demonstrates our prior sampling procedure which predicts CLIP image embeddings from text captions. Similar to our decoder model, the posterior variance Σ_θ is set to $\tilde{\beta}_t$. Following Ramesh et al. [3], we generate two samples of image embeddings from two different noisy image embeddings, and select the one with a higher cosine similarity (dot product) with the CLIP text embedding. This approach improves sample quality by reducing the variance of the generated samples. As discussed in the previous subsection, our prior model directly predicts the mean of the image embedding instead of the diffusion noise. If we are not in the final step of the reverse process, i.e. $t > 1$, we add a posterior noise to the

Algorithm 5 Prior sampling procedure

Require: model f_θ, text caption y, CLIP text encoder h, diffusion schedules $\beta_t, \alpha_t, \bar{\alpha}_t, \tilde{\beta}_t$

1: $\Sigma_\theta(z_i^{(t)}, t) \leftarrow \tilde{\beta}_t$ ▷ Fix the posterior variance
2: $z_t, c \leftarrow h(y)$ ▷ Obtain text embeddings and text encodings
3: $z_1^{(T)}, z_2^{(T)} \sim \mathcal{N}(0, \boldsymbol{I})$ ▷ Sample two random latent image embeddings
4: **for** $t = T, \ldots, 1$ **do**
5: $\quad \hat{z}_1 \leftarrow f_\theta(z_1^{(t)}, t | z_t, c)$ ▷ Predict the first image embedding
6: $\quad \hat{z}_2 \leftarrow f_\theta(z_2^{(t)}, t | z_t, c)$ ▷ Predict the second image embedding
7: $\quad$ **if** $t > 1$ **then** ▷ Add posterior noise
8: $\quad\quad \tilde{\epsilon}_1, \tilde{\epsilon}_2 \sim \mathcal{N}(0, \boldsymbol{I})$
9: $\quad\quad z_1^{(t-1)} \leftarrow \hat{z}_1 + \Sigma_\theta(x_t, t) \cdot \tilde{\epsilon}_1$
10: $\quad\quad z_2^{(t-1)} \leftarrow \hat{z}_2 + \Sigma_\theta(x_t, t) \cdot \tilde{\epsilon}_2$
11: $\quad$ **else**
12: $\quad\quad z_1^{(0)} \leftarrow \hat{z}_1$
13: $\quad\quad z_2^{(0)} \leftarrow \hat{z}_2$
14: $\quad$ **end if**
15: **end for**
16: **if** $z_1^{(0)} \cdot z_t > z_2^{(0)} \cdot z_t$ **then** ▷ Return the z_i that yields a higher dot product
17: $\quad$ **return** $z_1^{(0)}$
18: **else**
19: $\quad$ **return** $z_2^{(0)}$
20: **end if**

mean to obtain the noisy image embedding at the previous timestep (line 9 and 10). Otherwise, we return the mean as the final unnoised image embedding.

Chapter 4

Experiments & Evaluations

In this chapter, we conduct experiments and evaluations on the model. We begin by introducing some metrics that are used to assess deep generative models. Then, we investigate the effects of the classifier-free guidance scale on the generated samples. Next, we study the properties of CLIP embeddings through a series of experiments. We also experiment with the text-guided image manipulation technique, and then we assess the attribute-binding capabilities of our model. Finally, we perform evaluations to assess the performance of our model.

4.1 Preliminaries

When evaluating deep generative models, the two most important aspects are fidelity and diversity. Fidelity refers to the quality of the individual samples generated by the model, with higher fidelity meaning the samples are more realistic or closer to the true data distribution. Diversity, on the other hand, refers to the variety of different samples that the model can generate. Ideally, a generative model should be able to produce both high-fidelity and diverse samples. However, in practice, there is often a trade-off between these two objectives. Improving the fidelity of the samples may lead to a model that generates less diverse samples, as the model might focus on generating only the most common or typical samples from the data distribution. Conversely, encouraging the model to generate more diverse samples might result in lower fidelity, as the model might generate samples that are less typical or further from the true data distribution. Finding the right balance between fidelity and diversity is a crucial aspect in training and using generative models. In the following subsections, we will introduce some of the evaluation metrics for deep generative models that we will be using to assess the performance (fidelity and diversity) of our DALL-E 2 model.

4.1.1 FID

The Fréchet Inception Distance (FID) [28] is a popular metric used to evaluate the quality of images generated by diffusion models and other types of generative models. It was introduced as an improvement over the Inception Score (IS) [29],

another commonly used metric for evaluating deep generative models. FID has been shown to correlate well with human judgment of image quality, making it a reliable metric for evaluating the performance of generative models. FID measures the distance between the distribution of generated images and the distribution of ground truth images, providing a quantitative measure of the similarity between the two distributions.

FID is calculated using the feature extraction layers of the Inception-v3 model [30], a convolutional neural network that has been pre-trained on ImageNet for image classification. The Inception model is used to extract features from the intermediate layer of the network for both the generated and real images. These features are assumed to follow a multivariate Gaussian distribution, and the parameters (mean and covariance) of these distributions are computed. FID is then calculated as the Fréchet distance between these two Gaussian distributions:

$$\text{FID} = \|\mu_1 - \mu_2\|_2^2 + \text{tr}\left(\Sigma_1 + \Sigma_2 - 2\left(\Sigma_1^{\frac{1}{2}} \cdot \Sigma_2 \cdot \Sigma_1^{\frac{1}{2}}\right)^{\frac{1}{2}}\right) \tag{4.1}$$

where μ_1, Σ_1 are the mean and variance of the generated image features, μ_2, Σ_2 are the mean and variance of the ground truth image features, respectively. The lower the FID, the better the generation quality. Since FID takes into account the mean as well as the variance, it is a metric that measures the fidelity and diversity of the generated samples at the same time.

However, the Gaussian assumption of FID may not always hold in practice, and a significant limitation of FID is its inherent high bias in the sample size [31]. To accurately compute FID, a sufficiently large sample size is required. Smaller sample sizes can result in an inflated estimation of the actual FID.

4.1.2 KID

Kernel Inception Distance (KID) [32] is another metric used to evaluate the performance of generative models. Similar to FID [28], KID also uses a pre-trained Inception model [30] to extract features from images. However, unlike FID, KID does not make assumptions about the Gaussian distribution of these features, making it a more flexible and robust metric.

KID is based on the Maximum Mean Discrepancy (MMD) [33], a measure of distance between two distributions. To compute KID, features are extracted from the real and generated samples using the Inception model. Then, the MMD is computed between the distributions of these features, using a polynomial kernel. The square of this MMD value is the KID score. A lower KID score indicates that the generated images are more similar to the real images.

One of the main advantages of KID over FID is that it is unbiased and does not require a large sample size to provide reliable results. This makes KID a more practical metric for scenarios where only a limited number of samples are available. Additionally, KID has been shown to correlate well with human judgment of image quality, making it a useful tool for evaluating and comparing the performance of

generative models.

4.1.3 Precision and recall

Commonly used evaluation methods, such as the FID [28] and the KID [32], while correlating well with the fidelity and diversity of the generated samples, only yield one-dimensional scores. A low FID or KID score may indicate high fidelity, high diversity, or anything in between. Precision and recall [34] was proposed to disentangle the divergence into two separate dimensions, providing a more nuanced evaluation of generative models. Precision represents the proportion of high-quality generated samples, capturing fidelity, while recall quantifies the coverage of the ground truth data manifold by the generated samples, reflecting diversity. This method attempts to compare the learned distribution Q with a reference distribution P: precision intuitively measures the proportion of Q that is covered by P, while recall measures the proportion of P that is covered by Q. Similar to FID and KID, this method operates on the Inception-v3 [30] feature space. Moreover, similar to the conventional interpretation, an F_1 score can also be computed:

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.2}$$

4.2 Effects of guidance scale

As explained in section 2.1.3, the classifier-free guidance scale s is a hyperparameter that controls the strength of the guidance signal during the diffusion process. It determines how far the model's predictions move towards the conditional generation and away from the unconditional generation. It has been shown that the guidance scale balances the trade-off between fidelity and diversity [7]. Choosing an appropriate value for the guidance scale is crucial for achieving good performance with classifier-free guidance.

In this section, we study the effect of the guidance scale on the generated samples. The guidance scale only affects our decoder (section 3.3) which employs classifier-free guidance. Figure 4.1 and 4.2 show some examples of generated samples using different guidance scales. When the guidance scale is small, the generated samples are diverse but the sample quality is poor. As the guidance scale increases, the sample quality improves together with better image-text alignment, while the generated samples also become more homogeneous.

We quantitatively investigate the effect of the guidance scale using the evaluation metrics described in section 4.1. To this end, we use our DALL-E 2 model to generate random samples using different guidance scales. The textual prompts used in the generation are randomly chosen from the ones in the training dataset, so that the generated samples all follow the same distribution. One caveat is that FID [28] is usually computed with a minimum of 10,000 samples, as smaller sample sizes can introduce bias and lead to overestimations of the actual FID score [31]. However, constrained by the computation resources available to us, generating such a

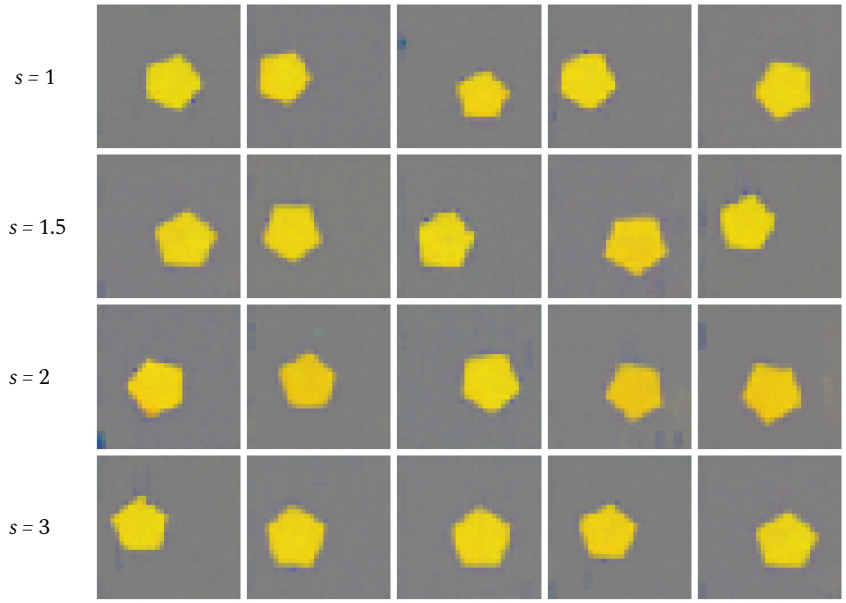


Figure 4.1: Various randomly generated samples corresponding to the text caption "a large gold pentagon", using different classifier-free guidance scales s.

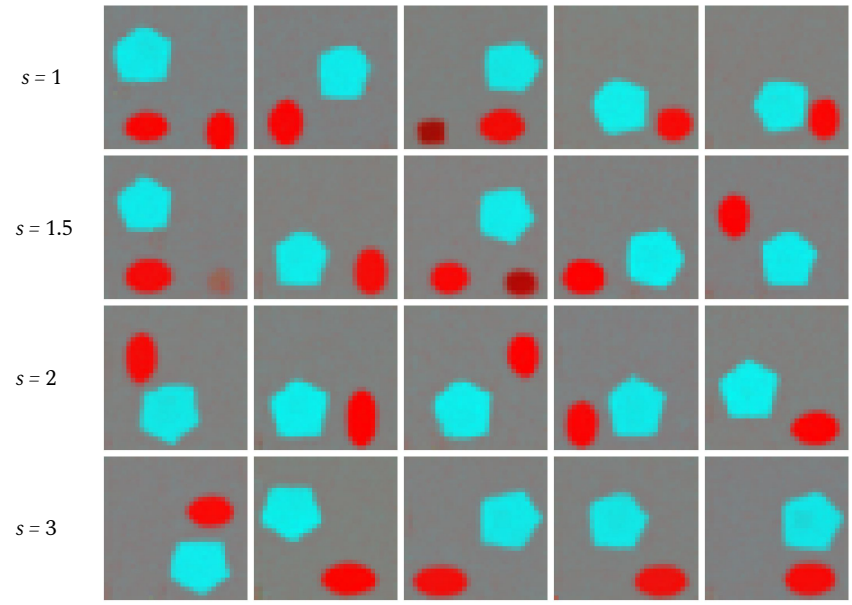
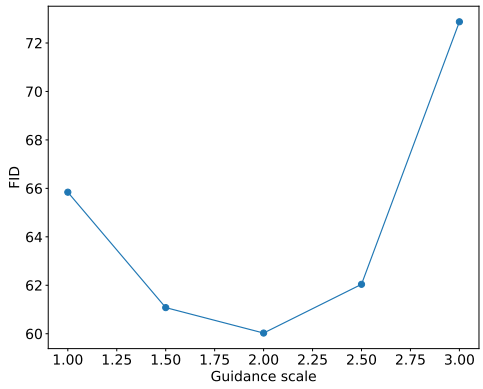


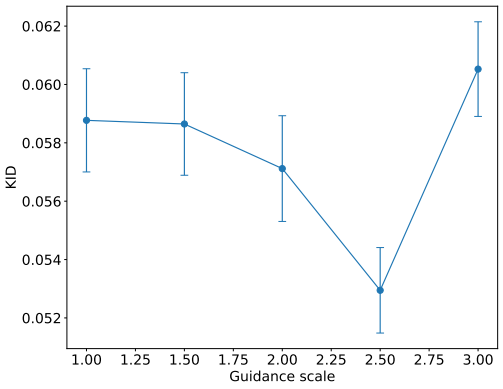
Figure 4.2: Various randomly generated samples corresponding to the text caption "a large cyan pentagon and the red ellipse", using different classifier-free guidance scales s.

large number of samples for each guidance scale is time-consuming. So we opt for a smaller sample size; we generate 3400 random samples for every guidance scale to ensure fairness.

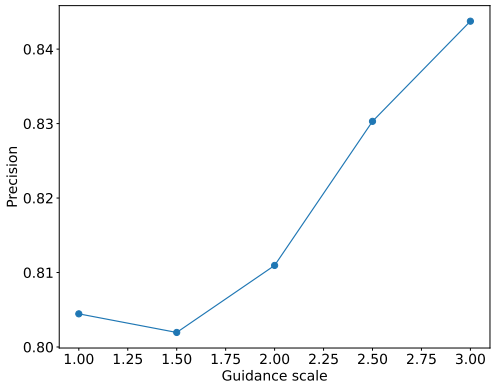
In Figure 4.3, we present various evaluation scores as the guidance scale is varied in $\{1, 1.5, 2, 2.5, 3\}$. Note that lower FID and KID scores indicate better sample quality. Setting $s = 0$ yields the unconditional decoder and setting $s = 1$ yields the



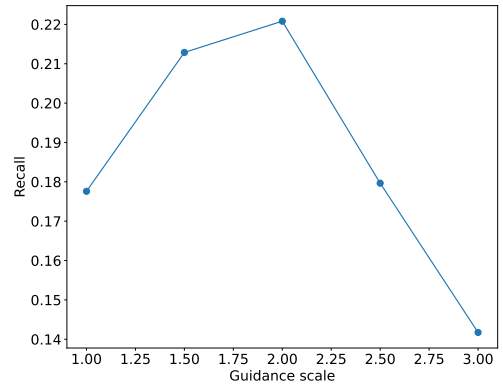
(a) FID



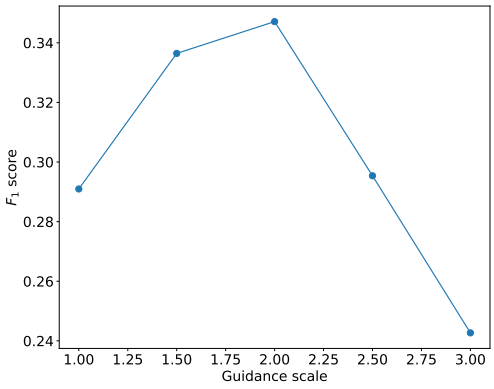
(b) KID



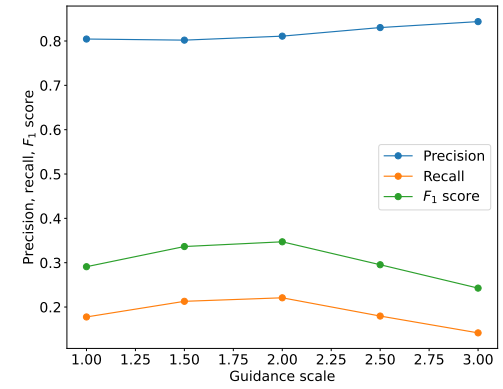
(c) Precision



(d) Recall



(e) F_1 score



(f) Precision, recall and F_1 score

Figure 4.3: Change in FID, KID, precision, recall and F_1 score as we change the classifier-free guidance scale in $\{1, 1.5, 2, 2.5, 3\}$.

conditional decoder. Setting $s > 1$ enables classifier-free guidance as the generation is guided further towards the conditional generation.

We see that as we increase the guidance scale, we have higher fidelity as the precision increases. As opposed to the hypothesis that the recall (diversity) should

decrease as the guidance scale increases, we find that our recall increases at first. A reason for this could be that despite the high diversity in the samples, the quality is so suboptimal when the guidance scale is small that many samples are not recognised as realistic images from the training dataset. As we further increase the guidance scale, recall begins to drop, conforming to the hypothesis that a higher guidance scale results in lower diversity. The recall values are smaller than the precision values because the size of the generated dataset (3400 images) is much smaller than that of the training dataset (20,000 images). FID, KID and F_1 scores are all holistic measures that depend on both fidelity and diversity. And the shapes of their graphs in Figure 4.3 demonstrate the fidelity/diversity trade-off. We find that the best values for the guidance scale are achieved at intermediate points, which could be either $s = 2$ or $s = 2.5$.

To summarise, the guidance scale is important because it balances the trade-off between following the guidance signal and preserving the stochastic nature of the diffusion process. If the guidance scale is too large, the model might overfit to the guidance signal and fail to capture the diversity of the target distribution. Conversely, if the guidance scale is too small, the model might not follow the guidance signal closely enough, resulting in poor-quality samples.

4.3 Investigate CLIP embeddings

In DALL-E 2, the CLIP embeddings are used to bridge the gap between the textual prompts and the images. In this section, we explore the semantics and relationships between the CLIP embeddings in the joint latent space.

4.3.1 PCA on CLIP embeddings

Since the CLIP latent space is a high-dimensional vector space, we use principal component analysis (PCA) to perform dimensionality reduction on the set of CLIP embeddings of all images in the training set. We project a selected subset of the image embeddings and text embeddings onto the first two principal components, which explains 12.27% and 10.08% of the variance respectively. The PCA projection is shown in Figure 4.4. We project the CLIP image embeddings of all six shapes in our dataset. We find that the shapes form two separate clusters along the first principal component, with ellipses, circles, pentagons and triangles being in the first cluster and rectangles and squares being in the second cluster. We also note that pentagons, circles and ellipses form a continuum along the second principal component, meaning that in the CLIP latent space, pentagons are similar to circles and circles are similar to ellipses. These observations conform with our intuition. We further note that compared to shapes, colours do not seem to play a big part in the first two principal components, as the image embeddings of red triangles and red squares are separated into two different clusters.

We also project the CLIP text embeddings of magenta circles and red squares in the PCA to investigate the similarity between the associated text embeddings and

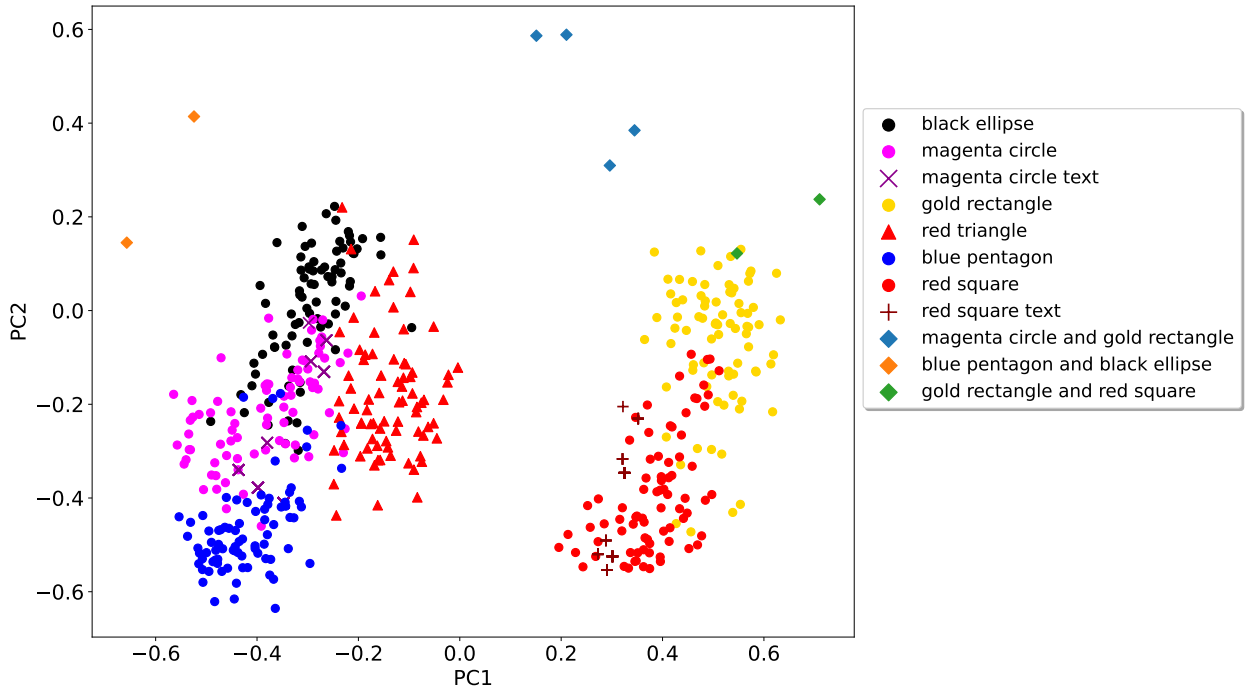


Figure 4.4: The CLIP embeddings projected onto the first two principal components

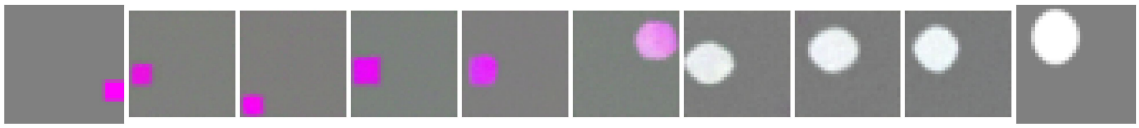
image embeddings. We find that these text embeddings fall into the cluster of the corresponding image embeddings, denoting a high alignment of text embeddings and image embeddings in the first two principal components. Furthermore, we study the image embeddings of shape combinations. Interestingly, the combinations all appear at the top in Figure 4.4, and the first principal component of the combination indicates what shapes are present in that combination. For example, an image that contains a circle and a rectangle will have an image embedding in the middle between a circle's embedding and a rectangle's embedding.

The clustering of shapes suggests that the model has learned meaningful representations of shape categories. By understanding the distribution of shapes in the latent space, we can better guide the generation process to ensure that the produced samples align with the desired shapes. This information can be utilized to design more effective sampling strategies that capture the full range of shape variations. For example, by leveraging the clustering and continuum observed, we can generate diverse samples within specific shape categories or smoothly transition between different shapes.

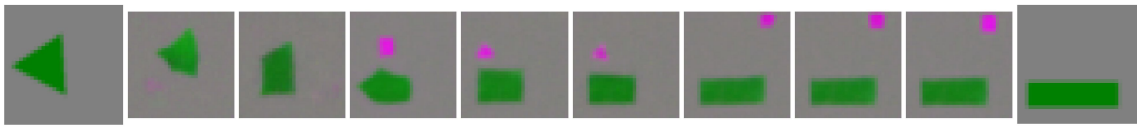
4.3.2 Image embedding interpolations

The PCA visualisations in the previous section gave us some insights into the structure of the high-dimensional CLIP latent space. To further explore the latent space and to test if our model has learned a smooth and generalised latent representation, we perform interpolations of image embeddings. We extract the image embeddings $z_i^{(1)}$ and $z_i^{(2)}$ from two selected images using the CLIP image encoder.

We linearly interpolate between $z_i^{(1)}$ and $z_i^{(2)}$ to obtain a sequence of intermediate embeddings $z_i' = r \cdot z_i^{(2)} + (1 - r) \cdot z_i^{(1)}$, by varying $r \in [0, 1]$. Then, we only use our decoder (section 3.3) generate images using the intermediate image embeddings z_i'. For all of the decoder generations, we use a guidance scale $s = 2$.



(a) A small magenta square → A white ellipse



(b) A large green triangle → A large green rectangle

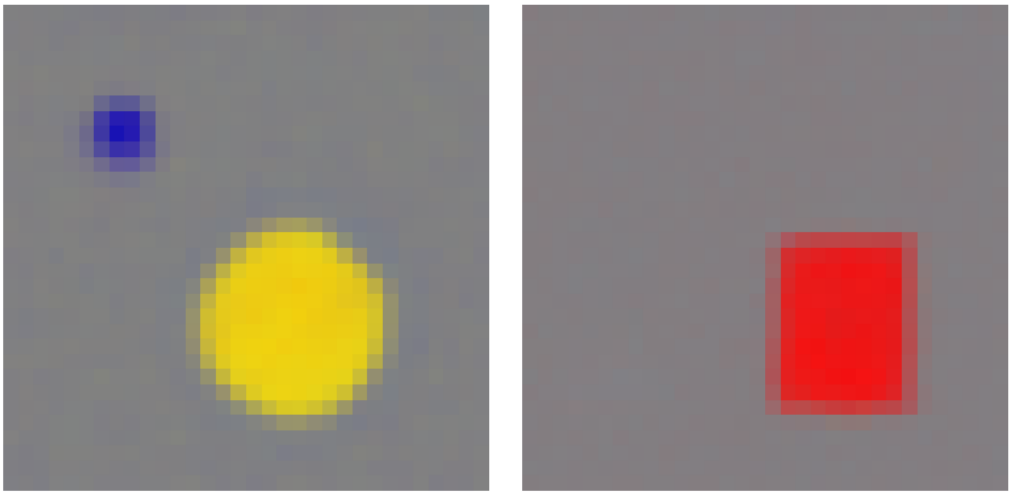
Figure 4.5: Generated images along the trajectory of the image embedding interpolations. The leftmost and rightmost images are the original images.

In Figure 4.5, we show two example interpolations. In the first interpolation, we can see that the transition is quite smooth and the intermediate points along the trajectory yield meaningful representations. In the second interpolation from a large green triangle to a large green rectangle, we note that in this trajectory, the shape of the object is first transformed and then the orientation is adjusted. We can also see that the interpolation quality is not as good as the first one. There are some odd shapes in the early trajectory and an additional artefact is present in many intermediate images. These observations indicate that the model struggled to maintain smooth and coherent transitions between the shapes in this particular interpolation. One possible reason for this is that the specific combination of shape and colour in the interpolation posed a more complex task for the model to handle. It is possible that the model did not have sufficient exposure to similar shape and colour combinations during training, leading to difficulties in generating coherent and visually appealing interpolations.

From the image embedding interpolations, we find that our model has learned a generalisation in the latent space as it is able to generate samples from arbitrary points in the latent, not just from the encodings of the training data. However, sometimes the sample quality from unseen points in the latent space is compromised.

4.3.3 Exploring text embeddings

The CLIP text encoder is a continuous space encoder that maps textual prompts into a continuous latent space. It has been demonstrated that these representations capture the syntactic and semantic language information, and relationships between different texts are characterised by their latent vector offset [35]. An example is that "King - Man + Woman" yields a vector close to "Queen", assuming all the words are vector embeddings and additions and subtractions are vector operations. To test if the CLIP text embeddings satisfy these properties, we first perform vector arithmetic using the CLIP text embeddings. Then we use the prior to generate an image embedding from the final text embedding. Finally, we use the decoder to visualise the image that corresponds to the text embedding after the vector offset operations.



(a) "A large gold rectangle" − "A rectangle" + "A circle"

(b) "A black rectangle" − "Black" + "Red"

Figure 4.6: Two visualisations of the vector offset operation.

In Figure 4.6, we present two example visualisations of the vector offset operation. For the first image, we subtract the text embedding of "A rectangle" from that of "A large gold rectangle", and we subsequently add the embedding of "A circle". We can see that the image generated from the resulting text embedding is indeed a large gold circle. The second image is based on the text embedding of "A black rectangle", but we take away the black colour and add a red colour. The resulting text embedding managed to produce a red rectangle. We can see that our CLIP text encoder managed to capture syntactic and semantic information manifested through vector offset, consistent with the findings from [35].

4.4 Text-guided image manipulation

A major advantage of using CLIP in DALL-E 2 is that text embeddings and language embeddings are mapped to the same latent space. Ramesh et al. proposed text diffs [3], which allows us to modify an existing image to reflect a new text

description. If we have an image x and its corresponding text caption y, we extract their respective image embedding z_i and text embedding z_t using the CLIP model. Assume that we would like to modify the image with a new text description, we would obtain the new text embedding z_t'. We then compute the text diff vector $z_d = \text{norm}(z_t' - z_t)$ by first taking the difference between the new text embedding and the original text embedding and then normalising. Now we linearly interpolate from the original image embedding to the text diff vector, yielding intermediate CLIP representations $z_r = r \cdot z_d + (1 - r) \cdot z_i$. We then use the decoder to generate the modified image using z_r. In Figure 4.7, we show an example of text-diff generation as we linearly increase r from 0 to 0.5.

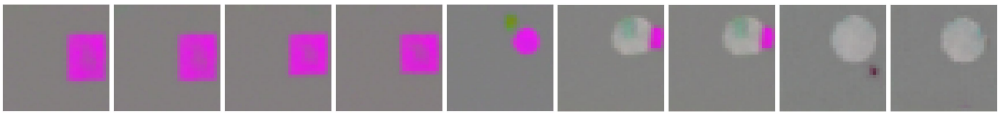


Figure 4.7: Text diff applied to the image corresponding to the text caption "a magenta rectangle". The new text description is "a large silver circle". We show a sequence of text diff modifications as we linearly increase r from 0 to 0.5.

The success of the modifications is particularly notable when using $r \approx 0.5$. At this point, the modified image closely aligns with the new text description, showcasing the ability of the model to incorporate the textual prompts and generate images that reflect the intended changes. The modified image effectively captures the essence of a large silver circle, demonstrating the capability of the text-guided image manipulation technique to produce meaningful transformations.

However, it is important to consider the potential degradation of samples when using r values greater than 0.5. As r increases beyond this threshold, there is a possibility that the quality of the generated images may decline. This degradation can be attributed to several factors, including the interpolation process and the sensitivity of the model to larger perturbations in the latent space. It is crucial to acknowledge this limitation and be mindful of selecting appropriate values of r to maintain the desired image quality in the modified results.

We can see that the final modified image reflects the new text description well. It is important to note that text-guided image manipulation is not the same as generating from the new text description directly. Text diff enables us to perform modifications based on the original image while still preserving some properties of the original image such as position in this case. Preserving certain properties of the original image, such as position, is a key aspect of text-guided image manipulation. By utilizing the text diff approach, modifications can be performed based on the original image while retaining important visual attributes. This ensures coherence and consistency in the transformation process. The ability to maintain some original properties in the image is essential to ensure that the modifications are visually meaningful and retain the intended context.

4.5 Attributes binding

The original authors of DALL-E 2 [3] found that the model struggles to bind different attributes to different objects as the model would often mix up attributes and objects. They hypothesized the reason is that CLIP embeddings do not inherently encode the attributes-binding information. To test this, we trained two separate decoders, one is only conditioned on CLIP embeddings and the other is only conditioned on text captions. Note that the second decoder model is essentially the same as GLIDE [17]. In Figure 4.8, we present comparisons of different generated samples from these two models. We can see that when there are two objects, the model that is only conditioned on CLIP embeddings would often mix up colours and objects and assign colours to the wrong objects. Whereas the model that is only conditioned on text captions manages to bind each colour to each shape correctly. We can see that CLIP indeed fails to capture the attributes binding information while natural language text captions allow the model to fill this gap. A potential limitation to consider is the availability and quality of the training data. The performance of attribute binding may be limited by the diversity and coverage of attribute-object pairs in the training dataset. Insufficient representation of attribute-object combinations during training can impact the model's ability to accurately bind attributes to objects.

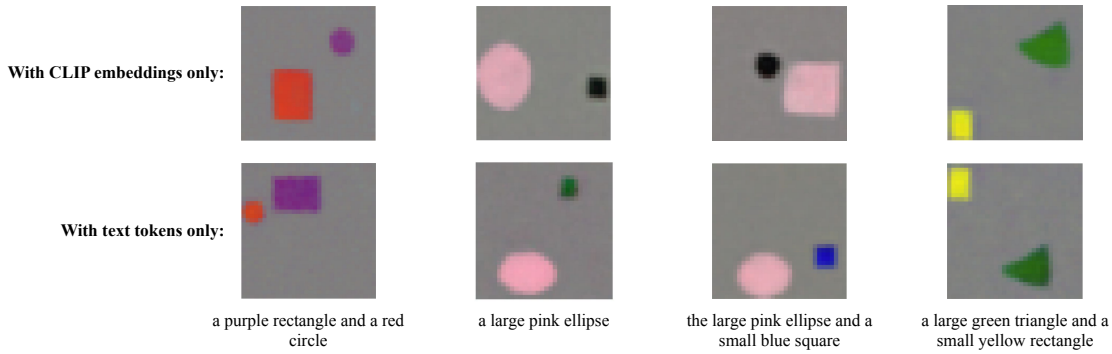


Figure 4.8: Comparison of the CLIP-embeddings-only decoder and the text-only decoder on their attributes binding capabilities. When there is only one object (the second sample), the two models produce similar results. But if there are two objects (the first and the third sample), the CLIP-embedding-only model would often mix up colours and shapes.

4.6 Final evaluation

In this section, we conduct final evaluations of our models. We first evaluate the individual constituent models, i.e. CLIP and prior. Then, we evaluate the entire DALL-E 2 pipeline. Next, we perform evaluations of the zero-shot capability to see to what extent can our model generalise. Finally, we discuss the computational efficiency of our model.

4.6.1 CLIP

CLIP encodes images and texts into a joint latent space and tries to align the corresponding embeddings of image-caption pairs. To assess how well our CLIP model manages to align them, we use the cosine similarity as introduced in section 3.2.3. Cosine similarity ranges from -1 to 1, and the higher the cosine similarity, the more similar the two vectors are. Table 4.1 shows the evaluation results of our CLIP model on the held-out test set. The test set contains some images and captions that are not present in the training set, so the results reflect the zero-shot capability of our CLIP model. The image-text similarity is the cosine similarity between text embeddings and the corresponding image embeddings. We achieved a value of 0.852 for this metric, denoting a good alignment between text embeddings and image embeddings. The random similarity is the cosine similarity between text embeddings and random image embeddings. We achieved a low value for random similarity, which means that our CLIP model has learned to make non-associated image-caption pairs as dissimilar as possible.

Image-text similarity	Random similarity	Top-1 accuracy	Top-5 accuracy
0.852	-0.016	83.71%	84.32%

Table 4.1: Zero-shot evaluation results of CLIP on the held-out test set.

We also use classification accuracy as an auxiliary measure of our CLIP's performance. As discussed in section 2.2.4, we can use CLIP to perform classification, where the text caption that yields the highest cosine similarity with an image out of a set of possible text captions is assigned to the image. We define the set of captions to be all of the captions from the test set. We can see that our CLIP model achieves 80%+ for both the top-1 accuracy and the top-5 accuracy.

4.6.2 Prior

Now, we perform evaluations of our prior model to assess the quality of the generated image embeddings. Similar to CLIP evaluations, we also use cosine similarity as our similarity measure. We conduct the evaluations on the same held-out test set and we present the results in table 4.2. Text similarity denotes the similarity between the generated image embeddings and the original text embeddings while image similarity denotes the similarity between the generated image embeddings

and the ground truth image embeddings. These two metrics are the two most important performance measures for the prior, with both achieving 0.9+. Baseline similarity is the same as the image-text similarity in table 4.1, which represents the similarity between the text embeddings and the ground truth image embeddings. This metric serves as a baseline for the performance of the prior model. Random similarity is computed by computing the cosine similarity between the generated image embeddings and a random text embedding. This metric is used to expose an overfit prior with the lower its value the better. Our value of -0.001 indicates a good fit of our prior.

Text similarity	Image Similarity	Baseline similarity	Random similarity
0.934	0.906	0.849	-0.001

Table 4.2: Zero-shot evaluation results of the prior on the held-out test set.

4.6.3 Full pipeline

In this subsection, we evaluate the end-to-end text-to-image generation capability of the DALL-E 2 model. We analyze the quality of the generated images and their alignment with the input text captions. Additionally, we investigate the novelty of the generated samples to determine if the model is generating genuinely new images or merely reproducing training examples. Through these evaluations, we gain insights into the performance and limitations of the full pipeline.

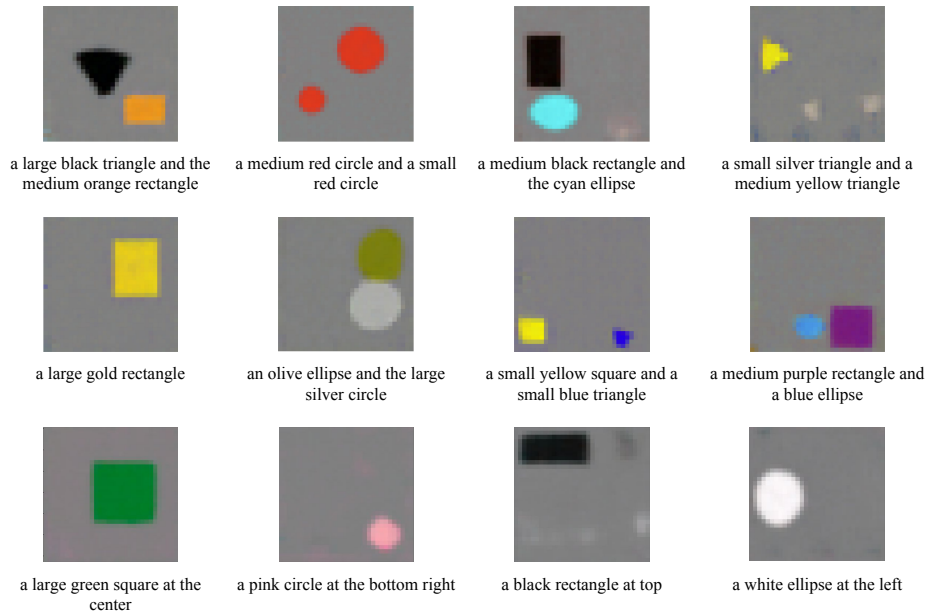


Figure 4.9: Generated samples from our DALL-E 2 model using random text captions.

The generated samples are presented in Figure 4.9. We can see that most of the samples generated by our model correspond well to the input text captions. In

terms of sample quality, there exists some degree of shape distortion, e.g. the black triangle in the top left image. Small objects particularly suffer from distortion due to the limited image resolution. We also note that occasionally there are small artefacts present in the generated images. In addition, we specify the position of the objects in the image for some generated samples as shown in the last row in Figure 4.9. We find that the model is able to distinguish the spatial information in text captions and place the object in the correct position, although there exist some occasional artefacts.

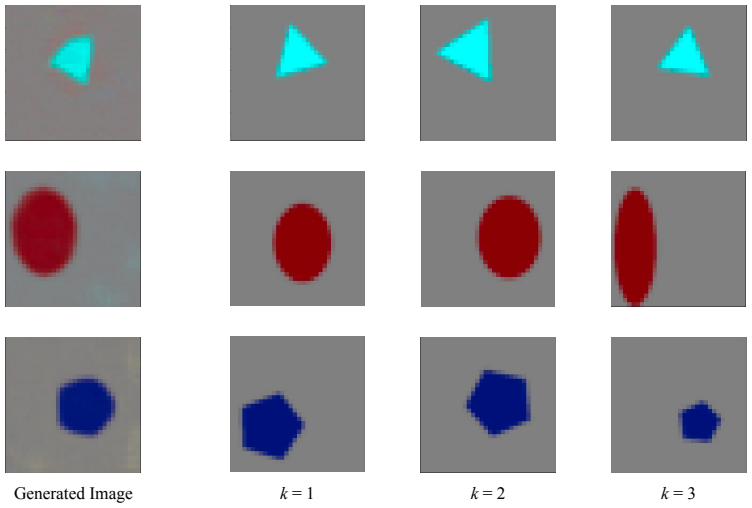


Figure 4.10: Generated samples and their top 3 nearest training images in terms of CLIP image embeddings.

We want to know whether our model is actually generating novel samples that it has never seen before or is merely remembering a mapping from text captions to training images. To investigate this, we first use the model to generate an image and use CLIP to obtain its image embedding. Then, we execute the k-nearest neighbour algorithm on the CLIP image embedding space. We retrieve the k-nearest neighbours to the generated image from the set of image embeddings of all training images. In Figure 4.10, we generate some images using the model and present their top 3 nearest neighbours from the training images. We can see that as opposed to remembering and replicating the training images during generation, the model is indeed generating new samples different from the training images that it has seen before.

4.6.4 Zero-shot evaluation

The generalisability of our model and its ability to generate samples for unseen images and text descriptions are important aspects to assess. We would like to assess the generalisability and investigate to what extent can the model generalise to unseen images and text descriptions. In this subsection, we present a number

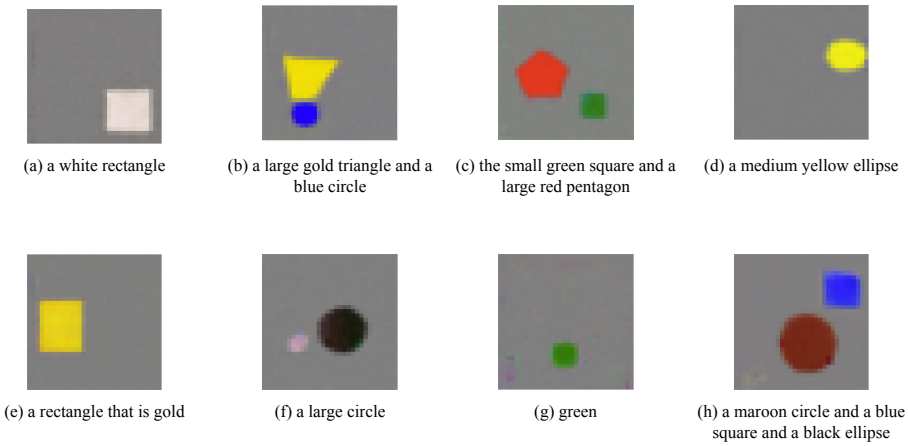


Figure 4.11: Zero-shot generated samples corresponding to unseen text captions and objects.

of generated samples corresponding to unseen text descriptions during training to evaluate various zero-shot capabilities. These capabilities are as follows:

1. **Recognition of Unseen Objects:** If the model has not seen a white rectangle during training, but it has seen rectangles of other colours as well as other shapes that are white, can it synthesise the available information and generate a white rectangle? To assess this capability, we deliberately excluded some combinations of colours and shapes in the training set, as mentioned in section 3.1. During test time, we provide captions that contain the excluded combinations to the model. The generated samples are shown in the first row in Figure 4.11, where all of the generated objects are unseen during training. We can see that the model managed to recognise and generate the unseen objects correctly, even when there are two unseen objects present.

2. **Rephrasing of Captions:** To evaluate the model's ability to understand the same meaning expressed in different ways, we rephrase seen captions into novel captions. For example, we rephrase "a gold rectangle" to "a rectangle that is gold." Figure 4.11 (e) demonstrates this scenario, where the provided caption is not present in the training set. The model successfully generates a gold rectangle, showcasing its capability to understand and generate samples for equivalent descriptions.

3. **Understanding of Partial Captions:** In the training set, object captions include both colour and shape information. However, we examine whether the model can understand captions that only provide either the shape or the colour. Figure 4.11 (f) presents the text caption "a large circle," and the model generates a large black circle accurately. Similarly, in Figure 4.11 (g), the text caption "green" prompts the model to generate a small square that

is green. These examples demonstrate the model's ability to infer missing information from partial captions.

4. **Extension of Object Count:** While the training set limits the presence of up to two objects per image, we investigate whether the model can generate more than two objects. However, in Figure 4.11 (h), we find that the model fails to generate more objects than it has seen before. Additionally, the generated objects only correspond to the first two objects mentioned in the caption.

In conclusion, our model demonstrates the ability to generalize to many different types of unseen scenarios. It successfully generates samples for unseen objects, understands rephrased captions, and infers missing information from partial captions. However, there are limitations in its ability to generate more objects than those encountered during training.

4.6.5 Computational efficiency

The architectural modifications discussed in chapter 3 have drastically reduced the number of parameters of our DALL-E 2 model. In table 4.3, we compare our model with the original DALL-E 2 in terms of the number of model parameters. We can see that our DALL-E 2 is a much smaller model with the number of parameters being only 3% of that in the original DALL-E 2.

	CLIP	Prior	Decoder	Total
Original DALL-E 2	972M	1B	4.5B	6.5B
Our DALL-E 2	4.3M	4.3M	10M	18.6M

Table 4.3: Comparison of the number of model parameters between the original DALL-E 2 [3] and our DALL-E 2.

We also report the training time. All of the models are trained using 32×32 images. Our final decoder model is trained for 470,000 iterations (gradient descent steps) on a 24GB NVIDIA A30 GPU, and the training lasted for 26 hours. Our final prior is trained for 10,000 iterations on the same GPU and the training process took 1 hour. We train our CLIP model on a 16GB NVIDIA Tesla T4 for 2,200 iterations and it took half an hour to train. Finally, we report the sampling time. We averaged over 1000 sampling processes and the average end-to-end sampling time is 7.72 ± 0.13s.

Chapter 5

Conclusion

In this project, we have conducted an exploration of OpenAI's DALL-E 2, from its theoretical underpinnings to its practical implementation. Through a detailed examination of its architecture and its training/sampling process, we have demystified the process of implementing and experimenting with this complex model. The architectural modifications presented in this work, including a reduction to 3% of the original DALL-E 2's parameters, have made it feasible to train DALL-E 2 on commercial GPUs with less training data. This significant reduction in parameter size was enabled by the use of a customised dataset of simple geometric shapes. This simple dataset also yielded more interpretable experimental results, enabling us to better understand the DALL-E 2 model.

We have conducted a series of experiments to demonstrate the capabilities and limitations of our DALL-E 2 model. We quantitatively and qualitatively demonstrated the effect of the classifier-free guidance scale on the sample fidelity and diversity. We then investigated the properties of the CLIP embeddings that are used extensively in our DALL-E 2 model. Through principal component analysis on the CLIP latent space, we found that CLIP embeddings encode intuitive and interpretable information of both images and texts. We performed linear interpolation in the CLIP latent space and discovered that CLIP learns a smooth latent representation, in which every point encodes meaningful information. We computed vector offsets of the CLIP text latents and found that text latents capture syntactic and semantic language information. Next, we experimented with DALL-E 2's text-guided image manipulation feature using text diffs and found that the resulting image conforms with the guiding text while preserving some properties of the base image. By experimenting with two decoders conditioned on CLIP embeddings and text captions respectively, we found that the results supported the hypothesis that natural languages retained more attribute-binding information than CLIP embeddings.

Finally, we evaluated both the individual constituent models and the entire DALL-E 2 pipeline. Besides quantitative evaluation metrics, we subjectively evaluated the sample quality and caption alignment of the generated samples. Although there are occasional distortions and artefacts, the generated samples achieved plausible results in both aspects most of the time. Next, we conducted k-nearest neighbours

of the generated samples to verify that our model is generating novel samples. We also conducted experiments to assess our model on different kinds of zero-shot tasks. Out of 4 different zero-shot tasks, our model succeeded in 3 of them, demonstrating good generalisability.

5.1 Future work

DALL-E 2 is a highly complex model with a lot of moving parts. Although we performed some degree of hyperparameter tuning, we are certain the current set of hyperparameters is far from optimal. To truly reflect the full potential of DALL-E 2, we would need to conduct more extensive hyperparameter tuning.

Our current dataset is very simple, containing only geometric shapes at 32×32 resolution. If we were granted sufficient computing resources, we would like to upscale our dataset to a more sophisticated one or even use real-world internet datasets similar to the one used in the original DALL-E 2. If we were to use a more sophisticated dataset, we would also need to upscale our model. Sampling speed then becomes crucial with such large models. We would implement the denoising diffusion implicit models (DDIMs) [36], which is shown to greatly improve sampling speed. We would also adopt a learnable posterior variance schedule [6] as opposed to using the fixed variance schedule that we currently use. The learnable variance schedule is found to greatly reduce the number of diffusion timesteps, thus improving sampling speed.

We would also like to implement the original DALL-E 2's inpainting and outpainting features. Inpainting allows users to fill in missing parts of an image while outpainting extends beyond the original image boundaries. We would also conduct experiments to better understand how these techniques work.

5.2 Ethical Issues

This project does not involve any human participants or any form of personal data because the training images only consist of simple objects drawn by a script. We also do not foresee any scenario of dual use or misuse as the model will only be capable of generating images of simple objects.

One issue of the project is that training a deep learning model is not energy efficient. Training a large Transformer model can emit nearly 5 times as much as an average American car during its lifetime [37]. To mitigate this issue, we would reduce the number of model parameters to make the training faster and more energy efficient. We could also implement the mechanisms described in the previous section to accelerate training and sampling.

Appendix A

Hyperparameters

	CLIP	Decoder	Prior
Diffusion steps	–	300	200
Noise schedule	–	cosine [6]	cosine [6]
Posterior variance schedule	–	fixed [1]	fixed [1]
U-Net channels	–	$64, 128, 256$	–
Time embedding dimension	–	32	–
Null embedding rate	–	0.2	–
CLIP embedding dimension	256	–	–
Transformer dimension	64	64	256
Transformer heads	16	8	16
Transformer layers	8	6	8
Max sequence length	33	33	33
Batch size	256	64	256
Iterations	2.2 K	470 K	10 K
Learning rate	$3\text{e}-4$	$1.0\text{e}-3$	$1.0\text{e}-2$
Adam β_2	0.999	0.999	0.999
Adam ϵ	$1.0\text{e}-8$	$1.0\text{e}-8$	$1.0\text{e}-8$

Appendix B

More Generated Samples

Figure B.1: We also train the model on the CIFAR-10 dataset and we visualise some of the generated samples.

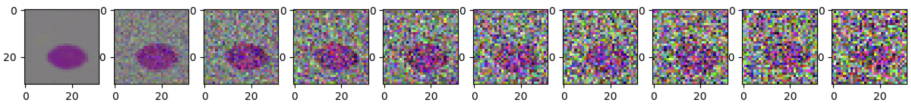


Figure B.2: The DALL-E 2 sampling process that corresponds to the caption "a large purple ellipse".

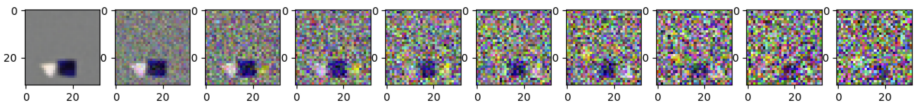


Figure B.3: The DALL-E 2 sampling process that corresponds to the caption "a white triangle and a small navy square".

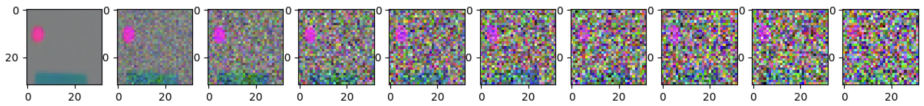


Figure B.4: The DALL-E 2 sampling process that corresponds to the caption "a teal rectangle and a magenta ellipse".

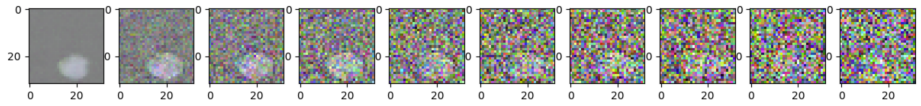


Figure B.5: The DALL-E 2 sampling process that corresponds to the caption "a large silver ellipse".

Bibliography

[1] Ho J, Jain A, Abbeel P. Denoising Diffusion Probabilistic Models. arXiv; 2020. Available from: https://arxiv.org/abs/2006.11239. pages 4, 8, 11, 12, 13, 14, 19, 22, 26, 31, 54

[2] Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, et al.. Learning Transferable Visual Models From Natural Language Supervision. arXiv; 2021. Available from: https://arxiv.org/abs/2103.00020. pages 4, 8, 12, 16, 17, 18, 22, 24, 25, 26

[3] Ramesh A, Dhariwal P, Nichol A, Chu C, Chen M. Hierarchical Text-Conditional Image Generation with CLIP Latents. arXiv; 2022. Available from: https://arxiv.org/abs/2204.06125. pages 4, 6, 8, 9, 11, 18, 19, 20, 21, 26, 28, 29, 30, 32, 34, 44, 46, 51

[4] Kingma DP, Welling M. Auto-Encoding Variational Bayes; 2022. pages 8, 11

[5] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative Adversarial Nets. In: Ghahramani Z, Welling M, Cortes C, Lawrence N, Weinberger KQ, editors. Advances in Neural Information Processing Systems. vol. 27. Curran Associates, Inc.; 2014. Available from: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf. pages 8, 11, 12

[6] Nichol A, Dhariwal P. Improved Denoising Diffusion Probabilistic Models. arXiv; 2021. Available from: https://arxiv.org/abs/2102.09672. pages 8, 13, 14, 31, 53, 54

[7] Dhariwal P, Nichol A. Diffusion Models Beat GANs on Image Synthesis; 2021. pages 8, 15, 19, 26, 27, 38

[8] Saharia C, Chan W, Saxena S, Li L, Whang J, Denton E, et al.. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. arXiv; 2022. Available from: https://arxiv.org/abs/2205.11487. pages 8, 11

[9] Rombach R, Blattmann A, Lorenz D, Esser P, Ommer B. High-Resolution Image Synthesis With Latent Diffusion Models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2022. p. 10684-95. pages 8, 11

[10] Sohn K, Lee H, Yan X. Learning Structured Output Representation using Deep Conditional Generative Models. In: Cortes C, Lawrence N,

Lee D, Sugiyama M, Garnett R, editors. Advances in Neural Information Processing Systems. vol. 28. Curran Associates, Inc.; 2015. Available from: https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf. pages 11

[11] Zhang H, Xu T, Li H, Zhang S, Wang X, Huang X, et al.. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. arXiv; 2016. Available from: https://arxiv.org/abs/1612.03242. pages 11

[12] Xu T, Zhang P, Huang Q, Zhang H, Gan Z, Huang X, et al.. AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks. arXiv; 2017. Available from: https://arxiv.org/abs/1711.10485. pages 11

[13] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al.. Attention Is All You Need. arXiv; 2017. Available from: https://arxiv.org/abs/1706.03762. pages 12, 17, 23, 24, 27, 32, 33

[14] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv; 2018. Available from: https://arxiv.org/abs/1810.04805. pages 12, 23, 24

[15] Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, et al.. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv; 2019. Available from: https://arxiv.org/abs/1907.11692. pages 12, 23

[16] Ho J, Salimans T. Classifier-Free Diffusion Guidance; 2022. pages 15, 34

[17] Nichol A, Dhariwal P, Ramesh A, Shyam P, Mishkin P, McGrew B, et al.. GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models. arXiv; 2021. Available from: https://arxiv.org/abs/2112.10741. pages 16, 19, 20, 26, 28, 46

[18] Alcorn MA, Li Q, Gong Z, Wang C, Mai L, Ku WS, et al.. Strike (with) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects. arXiv; 2018. Available from: https://arxiv.org/abs/1811.11553. pages 16

[19] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. p. 770-8. pages 16, 24

[20] He T, Zhang Z, Zhang H, Zhang Z, Xie J, Li M. Bag of Tricks for Image Classification with Convolutional Neural Networks. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society; 2019. p. 558-67. Available from: https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00065. pages 17

[21] Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, et al.. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv; 2020. Available from: https://arxiv.org/abs/2010.11929. pages 17, 24, 25

[22] Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I. Language Models are Unsupervised Multitask Learners; 2019. . pages 17

[23] Sennrich R, Haddow B, Birch A. 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). Berlin, Germany: Association for Computational Linguistics; 2016. p. 1715-25. Available from: https://aclanthology.org/P16-1162. pages 22

[24] Hendrycks D, Gimpel K. Gaussian Error Linear Units (GELUs); 2023. pages 24

[25] Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab N, Hornegger J, Wells WM, Frangi AF, editors. Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. Cham: Springer International Publishing; 2015. p. 234-41. pages 26

[26] Wu Y, He K. Group Normalization. In: Proceedings of the European Conference on Computer Vision (ECCV); 2018. . pages 26, 28

[27] Elfwing S, Uchibe E, Doya K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. Neural Networks. 2018;107:3-11. Special issue on deep reinforcement learning. Available from: https://www.sciencedirect.com/science/article/pii/S0893608017302976. pages 28

[28] Heusel M, Ramsauer H, Unterthiner T, Nessler B, Hochreiter S. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, et al., editors. Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc.; 2017. Available from: https://proceedings.neurips.cc/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf. pages 36, 37, 38

[29] Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X, et al. Improved Techniques for Training GANs. In: Lee D, Sugiyama M, Luxburg U, Guyon I, Garnett R, editors. Advances in Neural Information Processing Systems. vol. 29. Curran Associates, Inc.; 2016. Available from: https://proceedings.neurips.cc/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf. pages 36

[30] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the Inception Architecture for Computer Vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. p. 2818-26. pages 37, 38

[31] Chong MJ, Forsyth D. Effectively Unbiased FID and Inception Score and Where to Find Them. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2020. . pages 37, 38

[32] Bińkowski M, Sutherland DJ, Arbel M, Gretton A. Demystifying MMD GANs; 2021. pages 37, 38

[33] Gretton A, Borgwardt KM, Rasch MJ, Schölkopf B, Smola A. A Kernel Two-Sample Test. Journal of Machine Learning Research. 2012;13(25):723-73. Available from: http://jmlr.org/papers/v13/gretton12a.html. pages 37

[34] Kynkäänniemi T, Karras T, Laine S, Lehtinen J, Aila T. Improved precision and recall metric for assessing generative models. Advances in Neural Information Processing Systems. 2019;32. pages 38

[35] Mikolov T, Yih Wt, Zweig G. Linguistic Regularities in Continuous Space Word Representations. In: Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Atlanta, Georgia: Association for Computational Linguistics; 2013. p. 746-51. Available from: https://aclanthology.org/N13-1090. pages 44

[36] Song J, Meng C, Ermon S. Denoising Diffusion Implicit Models; 2022. pages 53

[37] Strubell E, Ganesh A, McCallum A. Energy and Policy Considerations for Deep Learning in NLP. arXiv; 2019. Available from: https://arxiv.org/abs/1906.02243. pages 53