Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Spatial Knowledge Injection into Diffusion-based Image Generation

Author:
Krishan Sritharar

Supervisor:
Prof. Alessandra Russo
Mr Yaniv Aspis

Second Marker:
Prof. Francesca Toni

November 2022 - June 2023

Abstract

In recent years, the growing field of artificial intelligence has opened up exciting opportunities in the realm of image synthesis, with a particular interest in the potential of text-to-image generation. Among the multitude of techniques available, diffusion models have emerged as a particularly promising approach, offering the capability to generate highly realistic images by sampling from complex probability distributions over time. However, the complexity of spatially and semantically accurate image generation from textual prompts remains a significant challenge within the field.

This project addresses these complexities by introducing a novel approach that integrates spatial knowledge into image generation using diffusion models. We developed a new methodology that extends traditional diffusion models by integrating with scene graph, which are a non-ambiguous, easily-specified tool for representing spatial relationships and semantics within a scene. Our unique two-stage generation process is composed of an annotator model and a conditioned diffusion model, which leverages the powers of Stable Diffusion and ControlNet to dictate the spatial orientation of objects to the diffusion process.

We use a custom dataset comprising images of MNIST digits arranged in grid formations, serving as an ideal sandbox to explore our proposed methodology's effectiveness. This choice of dataset focuses attention on the spatial layouts of objects, eliminating any other variables which may interfere with the results. We train four main methods of models: a baseline, a hand-crafted scene graph representation, and two control models designed using ControlNet, represented as typed digits and dots.

Evaluation of each method was performed with our custom design accuracy metrics and digit classifier. We observed object and relationship count accuracy, as well as relationship integrity and thoroughly evaluated all the models we trained, providing insights into surprising areas of performance. We deduced the dots approach as the superior concept since it outperformed the baseline results by nearly 20%.

By highlighting the importance of spatial understanding in image generation and by providing a methodology for fulfilling this request, this work makes a significant contribution to the field and paves the way for more sophisticated applications that will reshape the landscape of AI-driven image generation.

Acknowledgements

I would first and foremost like to express my deep gratitude to my supervisors **Prof. Alessandra Russo** and **Mr Yaniv Aspis**, for their unwavering support and guidance throughout this project.

I am deeply appreciative of the regular and dedicated time that they allocated for me over the past months. These discussions significantly contributed to the progress of the project, and their diligence in ensuring we stayed on track was both admirable and highly beneficial. Their readiness to provide insightful guidance and constructive feedback, amidst their numerous responsibilities, is something I sincerely value. Their proactive approach truly played a pivotal role in the successful completion of this project.

Special thanks to all my friends for their insightful discussions, constant encouragement, and their collaborative spirit throughout this amazing final year.

Finally, but most importantly, to my parents and my little sister, thank you for your unyielding support and love throughout my entire academic journey and life. Your belief in my capabilities has been a source of motivation and resilience, and I would not be anywhere without you all.

Contents

Chapter 1

Introduction

Since the emergence of OpenAI's DALL-E 2, Stable Diffusion, Midjourney and other extremely impressive image generation AI, we have experienced a large shift in the amount of AI power that is now available to the general public. With a simple interface, we can now create very complex images in seconds, with only a brief description of the image's contents. Much of the creativity and realism of these text-to-image models have stemmed from the usage of diffusion models at the core. These are generative models which create images by learning to denoise randomly sampled noise. They have shown great success in generating high-quality samples whilst also being more flexible and scalable in comparison to similar purpose methods.

Introduced only in mid-to-late 2022, these image-generation AI models have taken the world by storm, generating a lot of attention and inviting a lot of further research in this space. This has identified many successes but also areas of improvement which aren't completely fulfilled by the current solutions. One such aspect is the spatial relationships between objects produced in the final image. It is difficult to extract the keywords describing this from the text prompt, and for this to be reflected in the final image. This makes seemingly simple prompts like "a red ball in front of a blue one" or "a red basketball with flowers on it, in front of a blue one with a similar pattern" repeatedly produce inconsistently incorrect results. This project is about exploring this problem in more depth and proposing a solution for solving it.

1.1 Motivation

The aim of this project is to enhance the authenticity of AI-generated images and align them more closely with user intentions. Despite progress in creating realistic images, many generated results lack the intricate spatial relationships found in real-world scenes, compromising their credibility and appearing unconvincing. Our goal is to devise a method for infusing spatial understanding into models, and improving their image generation quality in terms of obedience to the prompt. We have chosen to conduct this research using diffusion models as they are the current state-of-the-art in image generation technology.

Generating images that are more faithful to the user's prompt is important as they are in high demand for a variety of applications, including computer graphics, virtual

reality, and machine learning. A highly beneficial use case here will be improving the performance of CNNs. Collecting high-quality, diverse data can be thought of as one of the most difficult tasks of machine learning, but a good image generation model will be able to bulk produce realistic data, with the added possibility for data augmentation to reduce bias-driven classification.

For instance, most images of camels that we see are in the desert, so it is reasonable to assume a classifier has associated the background with the animal and it may have worse performance when the background is different. It is not easy however to get real photographs of camels in grassy fields, rain forests or snow but this shouldn't be a problem for an image generation model, as long as we can reliably produce images with objects in the correct positions relative to one another.

1.2 Objectives

This project aims to explore the possibilities of injecting spatial knowledge into the diffusion model process to create images better representing the given text prompt. At a high level, the objectives of this project are to:

1. Convert the spatial relations represented in a scene graph corresponding to a text prompt into a representation of knowledge.
2. Explore different diffusion processes and methods for injecting this spatial knowledge into the image generation process.
3. Evaluate the effectiveness of the proposed methods on a variety of tasks.

1.3 Technical Challenges

Several obstacles lie in this project's path in terms of technical challenges. Let's discuss a few, including what potential approaches they unlock.

Representing spatial knowledge

Real-world spatial relations are complex and require high levels of abstraction and scaling to be represented meaningfully. The difficulty also lies in handling abstract concepts, scales, ambiguous situations, and integrating spatial knowledge with other types of knowledge like semantics and time. Successful representation allows for more effective understanding and transfer of knowledge of scenes.

Training diffusion models

Training requires capturing complex, high-dimensional distributions, which often involve significant computational resources and sophisticated optimisation techniques. Additionally, they require careful handling of reverse dynamics, where small errors can quickly amplify, making the training process sensitive and potentially unstable. We avoid these pitfalls, by using a robust existing codebase for training our models.

Injecting spatial knowledge

The challenge lies in bridging the gap between the abstract, high-dimensional spaces of these models and the intuitive, structured understanding of space humans have. Deep acquaintance and modification of models are required to infuse the spatial knowledge in a way that doesn't disrupt the model's natural generation process.

Evaluating effectiveness of new models

Defining suitable success measures to focus on spatial relationship following requires careful consideration due to the subjective nature of image perception, and dealing with the absence of a clear 'correct' image for comparison. Additionally, the variability caused by configuration and underlying architecture reflected in model outputs, alongside the significant computational resources required for testing can complicate the evaluation process.

Generalising the approach

The range of possible prompts, contexts, spatial relationships, and desired outputs is vast and diverse. Learning how to correctly apply and adapt specific spatial knowledge to different contexts while maintaining a realistic and believable generation is a complex problem. Generalising however will allow for widespread usage and more faithful image generation across models.

1.4 Contributions

The contributions of this project can be summarised as follows:

1. Introducing a novel method of integrating spatial knowledge into diffusion models for image generation. This method is built on ControlNet's backbone and we created control models to generate more spatially obedient images.
2. Evaluating and analysing this novel method thoroughly through several models and 1000s of images. We designed two new accuracy metrics specifically for reviewing the spatial correctness in generated images.
3. The project and supporting codebase is fully open-source and available at: https://github.com/KrishanSritharar2000/Digit-Spatial-Diffusion and all the trained models are available at: https://huggingface.co/Krishan3168/Digit-Spatial-Diffusion.

This report will begin by discussing image generation and knowledge representation methods in the Background in Chapter 2, followed by an overview of project's approach in Chapter 3, before diving into the technical details. We begin that by a discussion on technical setup in Chapter 4, and then explain all steps of implementation in Chapter 5. We discuss our evaluation methodology and results in Chapter 6, before talking about related work in Chapter 7 and finally wrapping everything up with a conclusion in Chapter 8.

Chapter 2

Background

In this chapter, we introduce the theory behind image generation and knowledge representation. We discuss in detail the current methods of achieving them both, compare their advantages and drawbacks, and cover what currently are the most appropriate use cases for them.

2.1 Image Generation

Image generation is the process of creating new, previously unseen images using a trained machine learning model. They are generated based on some initial prompt, usually text, and can lead to very realistic outputs, which are generally hard to distinguish from artwork produced by humans.

The idea of using machine learning to generate images was first introduced back in the 1970s and 1980s. Harold Cohen, who is now praised as a pioneer in computer, algorithmic and generative art, created AARON [1, 2], a computer program designed to create art that is comparable to human artists. It uses a set of rules and a knowledge base of art-making techniques to generate visuals and was able to render them onto canvases using a specialised drawing arm. AARON couldn't learn new styles of imagery on its own instead, new capabilities needed to be hand-coded by Cohen. Despite this, it was capable of creating truly original and creative works based on what it knows, that followed a similar style of artwork at the time. This achievement makes it quite comparable to modern-day alternatives which accomplish similar feats compared to modern images.

Image generation as we know it however, only started in 2014 with the introduction of GANs 2.1.1.1 [3]. Neural networks, increased computational power and extremely large datasets are what have helped advance this field to how it is today. Several methodologies and architectures have been invented for this purpose and we will explore them in the following sections.

2.1.1 Possible Approaches

2.1.1.1 GANs

Generative Adversarial Networks (GANs) [3, 4], introduced by Ian Goodfellow and colleagues in 2014, are a powerful tool for image generation, among other applications. GANs consist of two main components, a generator network G and a discriminator network D. The generator network maps a random noise vector to synthetic samples such as images, aiming to capture the real data distribution. In contrast, the discriminator network takes an image as input and classifies whether it is real or generated by G. This adversarial process (shown in Figure 2.1) continues until G can create images indistinguishable from real ones [3].

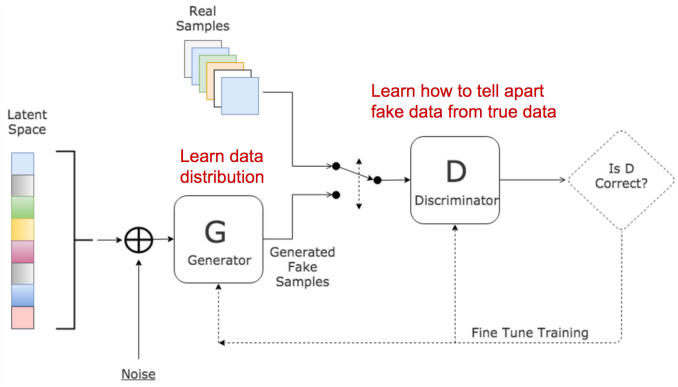


Figure 2.1: Architecture of Generative Adversarial Network [4]

While GANs have shown great potential, their use is not without difficulties [5]. Training challenges include mode collapse, which limits the diversity of generated images, and training instability due to the adversarial nature of the networks. Beyond these issues, GANs often lack control over specific features of the generated images, making them less suitable for tasks such as image manipulation. Evaluation of the generated images is also difficult due to the absence of clear objective functions. Moreover, GANs often struggle to generalise to large, high-dimensional data and perform poorly on rare or unseen data during training.

New methods for improving the performance and stability of GANs include the use of different architectures, loss functions, regularisation methods, and techniques such as Wasserstein GANs, Progressive GANs, and BigGANs [3, 6, 7]. Despite their challenges, GANs continue to be a promising area of research in image generation tasks, finding use in image synthesis, NLP [8] and style transfer among others.

2.1.1.2 VAEs

Variational Autoencoders (VAEs) [9, 10], are a type of generative model which can learn a compact, probabilistic representation of complex data like images or text. They operate by compressing input data to a lower-dimensional space and then reconstructing the original data from this compressed representation. VAEs consist of two main parts: an encoder and a decoder (shown in Figure 2.2). The encoder, a neural network, maps input samples to a lower-dimensional latent space via a stochastic process, producing a probability distribution. The decoder then uses this distribution in the latent space to reconstruct the input sample.

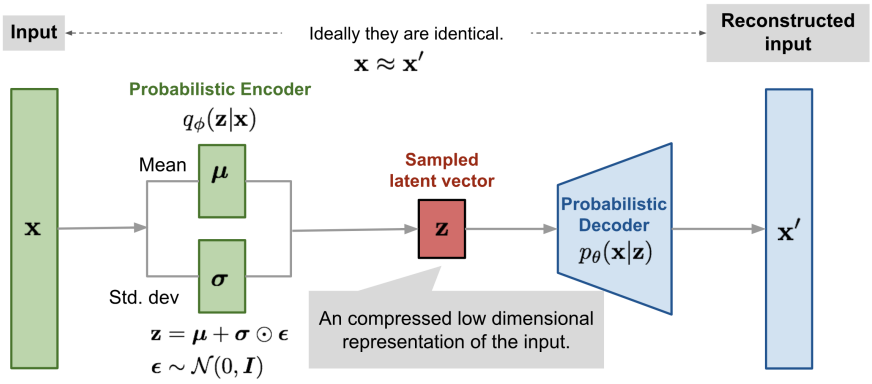


Figure 2.2: VAE architecture using a multivariate Gaussian assumption [9]

Unlike traditional autoencoders, VAEs introduce a probabilistic layer, enabling them to learn continuous data. They aim to maximise the likelihood of the training data and encourage the latent representations to adhere to a Gaussian prior. VAEs' loss function comprises a reconstruction loss (difference between input and reconstruction) and a regularisation loss [10] (distance of the latent space from a Gaussian).

VAEs share some challenges with GANs like mode collapse, training instability, and limited interpretability. One notable drawback is the lower quality of their generated samples [11], which can often be blurry or unrealistic. Modifications have been suggested to address this issue by not using the variational Bayes method [12].

While many applications overlap with GANs, VAEs excel at tasks such as anomaly detection, data compression, latent space manipulation, and image manipulation. They can identify anomalies by recognising high reconstruction errors, compress data efficiently, and generate new images or text by manipulating the latent space. The interpretability of this space however remains a challenge.

2.1.1.3 Flow-based Models

Flow-based models [13] are a class of generative models which use normalising flows to model complex data distributions. By transforming a simple prior distribution into the target data distribution via a series of invertible transformations (flows), they can learn the actual probability density of the real data, unlike GANs or VAEs.

A normalizing flow, the basic unit of flow-based models, maps a prior distribution to a target distribution through a sequence of invertible functions. These models are trained by maximizing the likelihood of the training data. The well-known Real NVP model [14], leverages affine transformations to learn the data distribution. This leads to an unsupervised learning algorithm with exact log-likelihood computation and efficient inference of latent variables, a challenging feat for other models.

While flow-based models have strengths like efficient computation and memory usage, model interpretability, and high-quality sample generation, they also have drawbacks. They require many parameters for complex distributions, are sensitive to initial distribution choice, and have limited control over generated samples. Furthermore, they struggle with modeling discrete data, such as categorical variables and integers, as they were designed for continuous data.

2.1.2 Latent Diffusion Models

Diffusion models [15, 16, 17, 18] are the cutting edge in image generation, pushing the boundaries of what is possible in terms of realism, creativity and control. They are a type of generative model that uses iterative refinement to generate very high-resolution and lifelike images. The fundamental concept involves using a sequence of invertible transformations to gradually evolve pure noise into a complex data distribution representing the desired image.

2.1.2.1 Background and Overview

Diffusion models are inspired by non-equilibrium thermodynamics from physics. This is the idea of molecules diffusing from areas of high density to low. In information theory, this equates to the loss of information due to the gradual intervention of noise. At their heart, diffusion models (specifically denoising diffusion models (DDPMs) [19] which are what we are referring to) define a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse the diffusion process to construct desired data samples from the noise. This makes them quite similar to flow-based methods 2.1.1.3 since they also have a concept of normalising flows and a sequence of invertible transformations.

Latent diffusion models (LDMs) introduce one extra component on top of traditional diffusion models, dramatically reducing the overall computational load. DDPMs operate on the image space meaning their inputs are usually of the size $(256, 256, 4)$ or even larger. This consequently provides lots of data to the diffusion model, making it more complex and taking much longer to train. LDMs however, introduce an encoding and decoding layer (essentially a VAE 2.1.1.2) on top of the diffusion model to reduce the input image dimensionality. Now with the same $(256, 256, 4)$ input image, our encoder can produce a latent space encoding of dimensionality $(32, 32, 4)$, thus reducing the amount of data whilst retaining most, if not all, the underlying information.

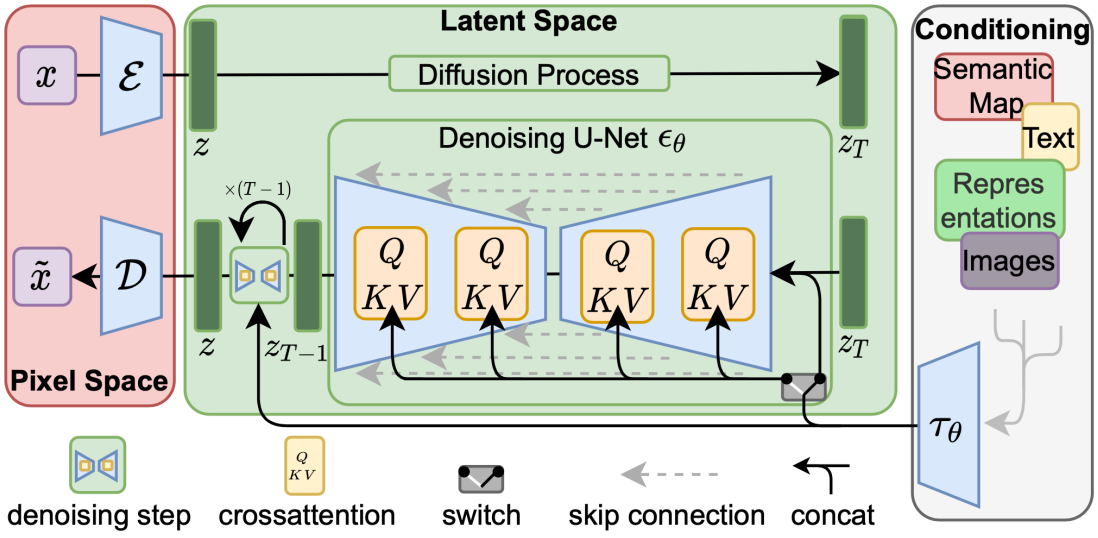


Figure 2.3: Architecture of latent diffusion model [20]

Figure 2.3 above provides a good overview of the entire process. In red we have the VAE process, with the Encoder and Decoder converting from the pixel space to the latent space and vice-versa. In green, we have the forward and reverse diffusion processes, with their respective architectures to create a noised vector and learn how to recover this. In the reverse process, we include conditioning mechanisms, which allow extra information, such as text and images, to be passed in and mapped to the generated images to understand a relationship.

2.1.2.2 Comparison with Other Image Generators

Let's compare the other generative models discussed in Chapter 2.1.1 with LDMs.

Generative Adversarial Networks (GANs)

GANs are capable of generating high-quality and sharp images, often outperforming other generative models in terms of visual quality. However, GANs are known for their instability during training and their susceptibility to mode collapse, where the generator only produces a limited variety of samples. On the other hand, LDMs offer a more stable training process since they directly optimise the data likelihood, and they have built-in uncertainty quantification due to their probabilistic nature, which is not readily available in GANs. Generating new samples can be computationally expensive due to the iterative nature of the process. It requires multiple steps (or 'diffusion' stages) to produce a sample, which could be slower compared to GANs.

Variational Autoencoders (VAEs)

VAEs, similar to LDMs, are probabilistic models and offer a well-defined latent space that allows for smooth interpolations and manipulations. However, the images generated by VAEs often suffer from being blurry. In contrast, LDMs have shown their ability to generate high-quality images. Moreover, LDMs employ an explicit noise process that allows for controlled generation and denoising, offering a different perspective compared to VAEs. It is important to note that VAEs are used in the architecture of LDMs so naturally LDMs will perform at the same level or better.

Flow-based Models

Like flow-based models, LDMs employ a similar idea of transforming a simple noise distribution into the data distribution, but through a discrete-time stochastic process instead. Flow-based models provide exact likelihoods and are naturally invertible, allowing for both efficient sampling and density estimation. However, they often struggle with generating high-quality images due to the architectural constraints needed to ensure invertibility. LDMs, meanwhile, can utilize flexible architectures like Transformers or ConvNets, which may result in better generation quality.

It is important to mention that out of all of these modes, LDMs have recently emerged as state-of-the-art generative models, excelling in a wide variety of tasks and outperforming other models in several benchmarks. Many breakthroughs and revolutionary models are being design used them, and their current infancy in research is leaving a lot on the table for innovative advancements and improvements. It is therefore why they have been chosen as the focus for this project.

2.1.2.3 Mathematical Derivations

LDMs can be broken into 4 main processes [19, 21]:

Encoding

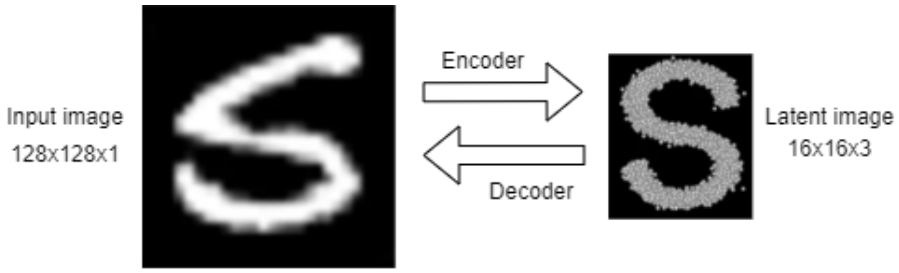


Figure 2.4: Encoding and decoding process

Given an input image x sampled from a real data distribution, we want to transform it into a latent encoding z, which represents the image in a different distribution. Figure 2.4 shows this process whereby we create a compressed representation of this data, with a smaller dimensionality of $16 \times 16 \times 3 = 768$ vs $128 \times 128 \times 1 = 16,384$.

Let ε be this encoding function, such that:

$$z = \varepsilon(x) \tag{2.1}$$

where: x = input space vector and z = latent space vector

This function will need to be learnt by training over a suitable number of samples.

Forward Process

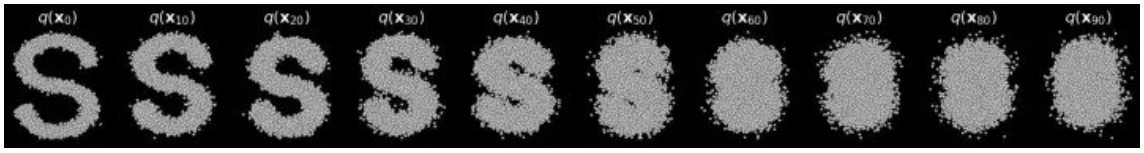


Figure 2.5: Forward diffusion process [18]

Figure 2.5 graphically shows the forward process. We start with a fully constructed image and over a series of steps, (here 100 [18]), apply Gaussian noise. Since each step is only dependent on the output from the previous, we can define this process as a Markov chain, so it doesn't require any training. By the time we reach the final step, the aim is for the image to be complete noise and have little to no resemblance to the original image.

A benefit of the Markov Chain property is that we do not need to apply a chain of operations to figure out the image at noise step 78 say. Instead, using the reparameterisation trick, we can apply all the noise in one step, saving computing resources.

Let $q(z_{1:T}|z_0)$ be this approximate posterior, then:

$$q(z_t|z_{t-1}) := \mathcal{N}(z_t; \sqrt{1-\beta_t} z_{t-1}, \beta_t \mathbf{I}) \tag{2.2}$$

where:

- $\beta_1, ..., \beta_T$ is a variance schedule with $\beta \in (0,1)_{t=1}^{T}$, which ensures that z_T is nearly an isotropic Gaussian for sufficiently large T
- $\mathcal{N}(z_t; \sqrt{1-\beta_t} z_{t-1}, \beta_t \mathbf{I})$ is a normal distribution with mean $\sqrt{1-\beta_t} z_{t-1}$ and covariance $\beta_t \mathbf{I}$ ($\mathbf{I}$ is the identity matrix)

From the reparameterisation trick,

$$q(z_{1:T}|z_0) := \prod_{t=1}^{T} q(z_t|z_{t-1}) \tag{2.3}$$

This leads to a series of increasingly noisy samples $z_1, ..., z_T$ being produced, and no training is required for this step.

Reverse Process

We now need to learn how to get back to the starting image z_0 from z_T.

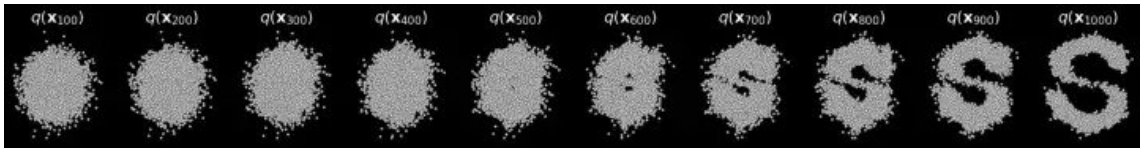


Figure 2.6: Reverse diffusion process [18]

Figure 2.6 shows the results of the reverse diffusion process. We can see that the final image isn't an exact reconstruction of the input image, but it is close enough to infer correctly. In this process, we predict the noise at each step and subtract it from the previous image. We repeatedly do this until we arrive at what we think the original image was. This process is not as straightforward as the forward process. We require knowledge of all previous gradients and thus will need a neural network to learn these.

Let $p_\theta(z_{t-1}|z_t)$ be the function this neutral network predicts, then:

$$p_\theta(z_{t-1}|z_t) := \mathcal{N}(z_{t-1}; \mu_\theta(z_t, t), \Sigma_\theta(z_t, t)) \tag{2.4}$$

shows that the model is trying to learn the mean and covariance (μ_θ and $\Sigma\theta$) of the normal distribution which goes from z_t to z_{t-1}.

The joint distribution $p_\theta(z_{0:T})$ is the underlying reverse process:

$$p_\theta(z_{0:T}) := p(z_T) \prod_{t=1}^{T} p_\theta(z_{t-1}|z_t) \tag{2.5}$$

Starting with pure Gaussian noise $p(z_T := \mathcal{N}(z_T, 0, \mathbf{I})$ and ending up at a vector $\tilde{z}$ close to z_0.

Conditioning

Conditioning information is added to the reverse process since this is the heart of the diffusion model. We are able to condition on a variety of types of input data: text, images, graphs, semantic maps, etc, so long as we are able to encode this data into some vector representation.

For text, we can use techniques like Word2Vec, or transformer-based models like BERT or Open AI's CLIP (Chapter 7.1.1.1). Given that some function $w()$ exists for this transformation, and we have the conditioning text "txt", the reverse time transitions will change to:

$$p_\theta(z_{t-1}|z_t) := \mathcal{N}(z_{t-1}; \mu_\theta(z_t, t, w(\text{txt})), \Sigma_\theta(z_t, t, w(\text{txt}))) \tag{2.6}$$

The bigger changes are in the architecture itself, where cross-attention mechanisms are introduced to aid this process.

Decoding

The last step is apply the decoder on $\tilde{z}$ to obtain our output image. Like the encoder, this function is also learnt through training.

$$\tilde{x} = \varepsilon^{-1}(\tilde{z}) \tag{2.7}$$

where ε is the encoding function, and thus ε^{-1} (ε inverse) is the decoding function.

2.1.2.4 Architecture

The overall architecture of LDMs is show in 2.3. Let's specifically talk about denoising U-Net segment.

It is common to use a special architecture called a U-Net model in the reverse process. This model downsamples and upsamples the data, forming a U-shape and hence the name. It is an encoder-decoder network where the encoder part of the network is used to extract features from the input image, and the decoder part is used to generate an output image. In the middle we have so-called "bottle-neck" layers, which ensure the network only leanrs the most important information. A unique feature is how certain layers are connected by skip connections, allowing for the decoder to have access to the high-level features extracted by the encoder, which helps to preserve the spatial information and fine details of the input image.

As outlined in Figure 2.7, U-Net architectures typically consist of a series of convolutional, max-pooling and upsampling layers. The convolutional layers extract features from the input image, max-pooing layers reduce the spatial resolution and increase the receptive field, and the upsampling layers increase the spatial resolution after the bottleneck.

Cross attention is a type of attention mechanism which are commonly used to help deep learning models to focus on the most relevant parts of inputs when making a prediction. In DDPMs, it is specially used to help design the skip connections paths, allowing for a fine spatial recovery in the U-Net decoder by filtering out non-semantic features. Essentially, it helps the model to focus on the most relevant parts of the input data and use more contextual information while ignore irrelevant parts.

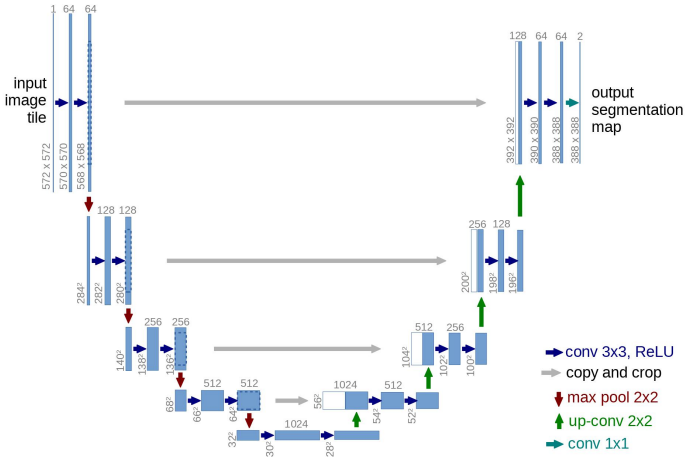


Figure 2.7: U-Net architecture [22]

2.1.2.5 Training

To train the entire LDM, we need to train both the VAE and diffusion model parts:

VAE

VAEs can be trained using a combination of two losses: a reconstruction loss and a Kullback-Leibler (KL) divergence loss[23]. The reconstruction loss measures how well the model can reconstruct the input data, i.e. how close the encoded then decoded result is to the ground truth. The KL divergence loss measures the difference between the model's latent distribution and a prior distribution, typically a standard Gaussian. This scores the latent space representation by comparing how close the two distributions are. It is preferable for both of these losses to be as close to zero as possible.

Here, $E_{q_\phi(z|x)}$ is the expected log-likelihood of the data under the decoder's distribution, essentially the reconstruction loss, and D_{KL} is the KL divergence:

$$\mathcal{L}(\theta, \phi) = E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \tag{2.8}$$

The goal of training is to find the parameters θ and ϕ that minimise this loss.

Diffusion model

The training process is just the reverse diffusion process. Authors of [19] found that predicting the noise component of an image at a given time step produced the best results. They arrived at the following objective:

$$L_{simple}(\theta) := E_{t,z_0,\epsilon}\left[\left\|\epsilon - \phi_\theta\left(\sqrt{\bar{\alpha}_t} z_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t\right)\right\|^2\right] \tag{2.9}$$

where:

- ϵ denotes the Gaussian noise added during the diffusion process.
- z_0 represents the initial sample before the diffusion process starts.

- The function $\phi_\theta(\cdot)$ is the denoising function that is parameterized by θ, the parameters of the model we aim to learn.
- t indicates the current time step in the diffusion process.
- $\bar{\alpha}_t$ is a pre-determined function dictating the noise level at each time step t.

The term $\sqrt{\bar{\alpha}_t} z_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ corresponds to the 'noisy' version of the sample at time t. The denoising function ϕ_θ takes this noisy version and the current time step t as inputs, and produces an estimate of the noise introduced by the diffusion process.

The loss function $L_{simple}(\theta)$ is the expected mean squared error between the true noise ϵ and the estimated noise $\phi_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)$, averaged over all time steps. The expectation $E_{t,x_0,\epsilon}[\cdot]$ is calculated over the distributions of time steps t, initial samples x_0, and noise values ϵ. We want to minimise this loss in order to effectively 'denoise' the samples.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 2.8: Training and sampling algorithms in DDPM [19]

Figure 2.8 shows the algorithm for both training and sampling. In training, we obtain an image from the training set and encode it. Then with a random timestep and Gaussian noise based on this timestep, we perform gradient descent using $L_{simple}(\theta)$. It is usual for diffusion models to take a large number of epochs (in the realm of hundreds of thousands) to train. With the large amount of computation per step, each epoch can take in the realm of tens of minutes, leading to hundred or thousands of hours training time in total. Therefore it is crucial to optimise and parallelise the training across many dedicated GPUs with large VRAM capacities.

Diffusion models can be specifically conditioned on certain categories during training [24]. Processes like ILVR [25] and score-based diffusion models [26] have shown it is possible to guide the generative process off some given reference. This is achieved by adding an extra latent variable value into the calculation after each training loop.

For sampling, the process begins with an initial vector z_T drawn from a standard multivariate normal distribution. The model then proceeds in reverse time order, starting from the final time step T and going back to the first time step. At each step, it modifies the current 'noisy' sample z_t using the denoising function ϕ_θ, and adds some noise that is scaled by σ_t. The noise levels at each time step t are controlled by predetermined functions α_t and $\bar{\alpha}_t$. This iterative application of the denoising function in reverse ultimately generates a sample z_0 that closely resembles those from the original data distribution. We perform this over a fixed number of timesteps, which can be modified at inference time, and importantly, we use the learnt parameters θ from training.

2.1.3 Stable Diffusion

Stable Diffusion [20], developed by StabilityAI, is an LDM used as a deep-learning text-to-image generative Artificial Intelligence model. This powerful model is capable of generating digital images from natural language descriptions, making it a revolutionary tool for digital art. It is a relatively recent concept in the field of generative modelling, introduced only in 2022, and it is primarily used to generate detailed images conditioned on text descriptions, though it can also be applied to other tasks such as inpainting, outpainting, and generating image-to-image translations guided by a text prompt.

Crucially, what makes it different to other state-of-the-art tools like DALL-E 2 7.1.1 and MidJourney 7.1.3 is that its source code and model weights have been released publicly, and it can run on most consumer hardware equipped with modest GPUs with at least 8 GB VRAM. This open-source nature has attracted a lot of community attention, leading to many tools (e.g. the browser interface Automatic 1111 [27]), and vast improvements (e.g. ControlNet 2.1.4) being developed quickly.

Under the hood, Stable Diffusion is simply an implementation of an LDM with specific adaptions in the U-Net architecture for the number, and sizes of the layers. Stability AI has computed and publically released several pre-trained Stable Diffusion checkpoints. Each has been trained on different image sizes, with different datasets, different encoders for text, and varying amounts of epochs.

The motivation behind the use of Stable Diffusion in this project stems from the advanced toolkit it provides, alongside the proven implementation of LDMs. There will be no benefit in implementing an LDM from scratch for this project, as this would instead obstruct the project's aim by taking away development time.

2.1.3.1 Finetuning

For Stable Diffusion, there are currently 4 mainstream methods of finetuning a pre-trained model to learn about and produce images on a new subject. These are Dreambooth [28], Textual Inversion, LoRA [29] and Hypernetworks. They all follow a similar style of creating a new unique identifier that the pre-trained model has not encountered before (e.g. "[V]") and then teaching the model to associate this identifier with a new custom subject. Figure 2.9 outlines this process clearly.

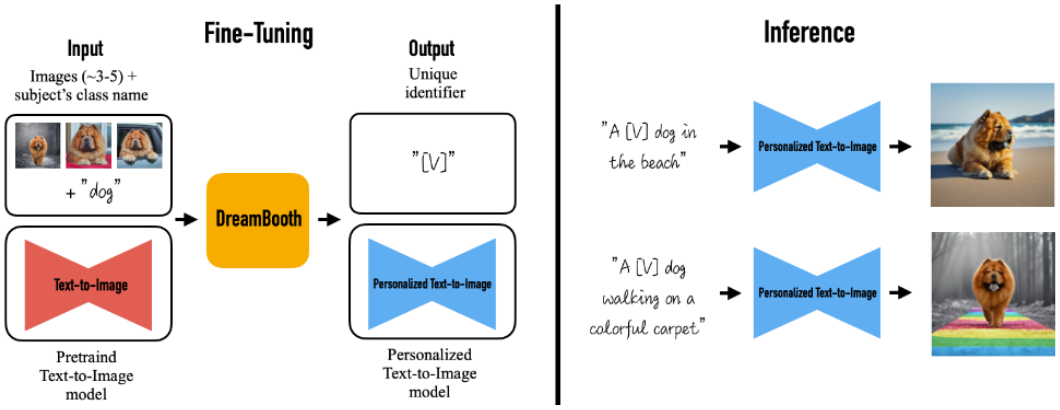


Figure 2.9: Dreambooth fine-tuning and inference [28]

With Dreambooth, in the training process, we perform gradient updates directly on the diffusion model. With textual inversion, instead of updating the model to understand the new identifier, we update the text encoding of the unique identifier to get closer to the visual representation of the object. The benefit here is that instead of our output being a large model, we have a small new embedding that has been learnt. With LoRA, we insert small layers into the pre-trained model and modify them on gradient updates. This approach trains faster whilst requiring less memory and producing an output smaller than Dreambooth. Finally, we have Hypernetworks, which very similarly to LoRA, adds small layers to the diffusion model. Crucially however, they sit one level higher and act as a model which outputs these intermediate layers. A downside with Hypernetworks is that we can not benefit from the mathematical optimisations which LoRA layers applied when they were manually constructed.

There is one other method called EveryDream, a general-purpose fine-tuning codebase. The follows a supervised learning approach by learning from the captions of each image, allowing for training multiple subjects and concepts. However, this is being developed by a single individual and there is a significant user base behind it.

2.1.4 ControlNet

ControlNet was introduced in "Adding Conditional Control to Text-to-Image Diffusion Models" [30] by Lvmin Zhang and Maneesh Agrawala in February 2023. It is a neural network structure that adds extra conditions to control diffusion models, bringing unprecedented levels of control to Stable Diffusion. Whereas previously Stable Diffusion provided minimal image-to-image control, ControlNet now presents a new architecture with numerous new control models.

The idea behind ControlNet is to provide an additional image as a condition to the diffusion model process, alongside the text. Importantly however, it aims to build on top of Stable Diffusion, to utilise its vast understanding of images and amazing image quality, essentially fine-tuning. Examples of these control images include edge maps, depth maps and line drawings, providing the diffusion process with different styles of input data.

2.1.4.1 Underlying Architecture

ControlNet works by adding trainable layers in parallel with Stable Diffusion's U-Net architecture. For each neural network block, we copy the weights into a "locked" copy and a "trainable" copy. The "locked" one preserves the model and the "trainable" one learns the additional condition.

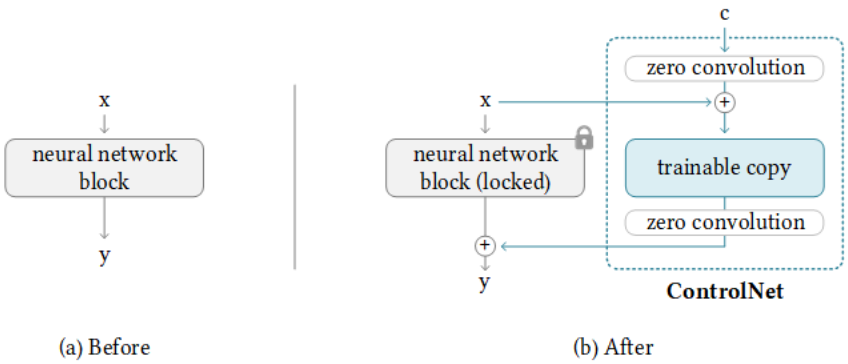


Figure 2.10: ControlNet parallel layers [30]

In Figure 2.10, we see the explained structure. x and y represent states before/after a block and c is the conditioning input. We have a special zero convolution operation which is a 1x1 convolution with both weights and bias initialised to zero. Overtime, these weights grow to optimised parameters in a learned manner. The original weights are then concatenated with these "zero-convolved" control weights, essentially making these zero convolution layers the bridge between Stable Diffusion and ControlNet.

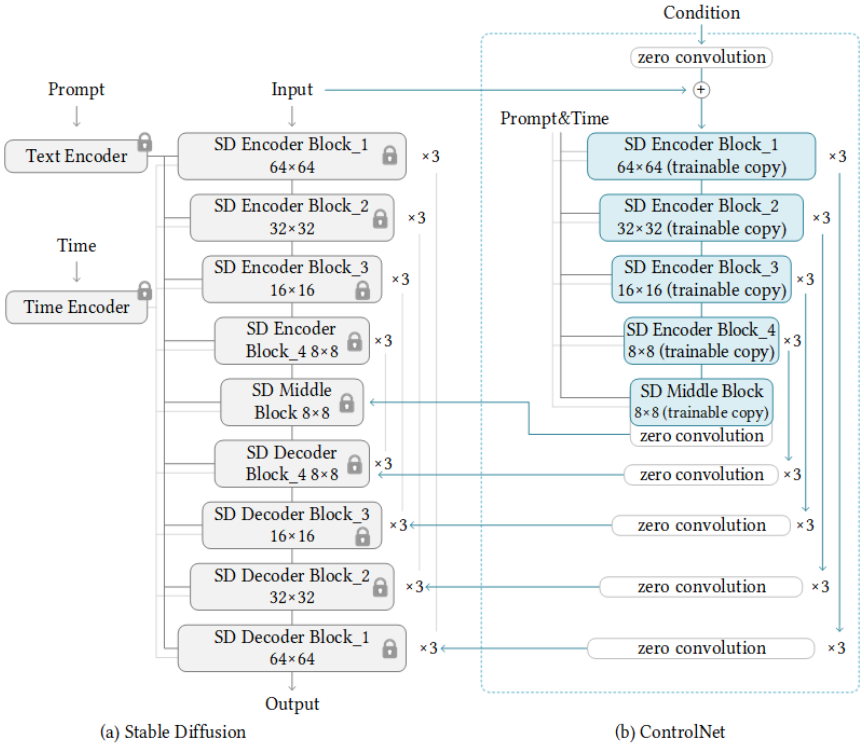


Figure 2.11: ControlNet in Stable Diffusion V1.5 and V2.1 [30]

Putting everything together, we get the overall architecture as shown in Figure 2.11. Since the zero convolution does not add new noise to deep features, the training is as fast as finetuning a diffusion model, compared to training new layers from scratch. Moreover, preserving the original weights allows for robust training on datasets of different scales, meaning ControlNet can condition on datasets with thousands of images even though Stable Diffusion itself required hundreds of millions.

2.1.4.2 Training

The authors of ControlNet have simplified the process of training a new ControlNet model (i.e. a new method of conditioning on images), by releasing documentation and training scripts in their repository. To do this, we first need to decide on the control method and prepare the conditioning dataset. This requires many triplets of the form (prompt, ground truth image, control image). The only requirement is that the control needs to be an image to comply with the released ControlNet architecture, as well as being the same shape as the Stable Diffusion model's input.

Next, we select a Stable Diffusion checkpoint to finetune and run a provided script to copy these weights into a new ControlNet checkpoint. This creates the parallel architecture shown in Figure 2.11. Finally, with some modifications, we can kick off a training script. Since no layer is being trained from scratch, the training is small-scale and therefore, low/mid-range GPUs could provide satisfactory performance.

2.1.4.3 Existing Models

When released, several control models were trained and provided by the authors. A selection of these are shown in Figure 2.12 below.

Figure 2.12: Comparison of six control models and the resulting images [30]

We have a source image of a house, and 6 control models: canny edge, HED, scribble, line, depth and a normal map. The images directly above these labels in Figure 2.12 are the control images which have been generated by a ControlNet annotator model, and have been passed into the ControlNet model alongside the text prompt "house". Each ControlNet model looks out for different input features and aims to replicate these in the generated images. For example, in the Canny Edge example, all the edges between objects in the source image are provided, allowing the generated image to retain these shapes and positionings, whilst allowing room for creativity.

Since its release, many new models being added (e.g. human pose, line art and multi-ControlNet models), and it presents a good solution to the spatial relationship problem, but importantly, only when you have a source image to ground this. This project focuses on providing additional spatial information through scene graphs.

2.2 Knowledge Representation

2.2.1 Scene Graphs

Scene graphs [15] are a type of structured representation of visual scenes, which are used to describe the objects and their relationships within an image or video. They consist of a set of nodes, each representing an object in the scene, and edges, representing the relationships between the objects, both possibly with attributes. As computer vision has advanced, we have become more interested in relationships between objects, image caption generation, visual question answering and image editing and retrieval. All these tasks require a high level of understanding and reasoning than just the initial image, which scene graphs provide.

2.2.1.1 Definition

Scene graphs can be represented in various ways, but a common representation is a graph, where the nodes represent objects, and edges represent relationships between the objects. The nodes can have attributes, such as object class, location, and size, and the edges can have attributes such as the type of relationship (e.g. "is-on" or "is-part-of"). Figure 2.13 shows a complex scene graph built off a source image.

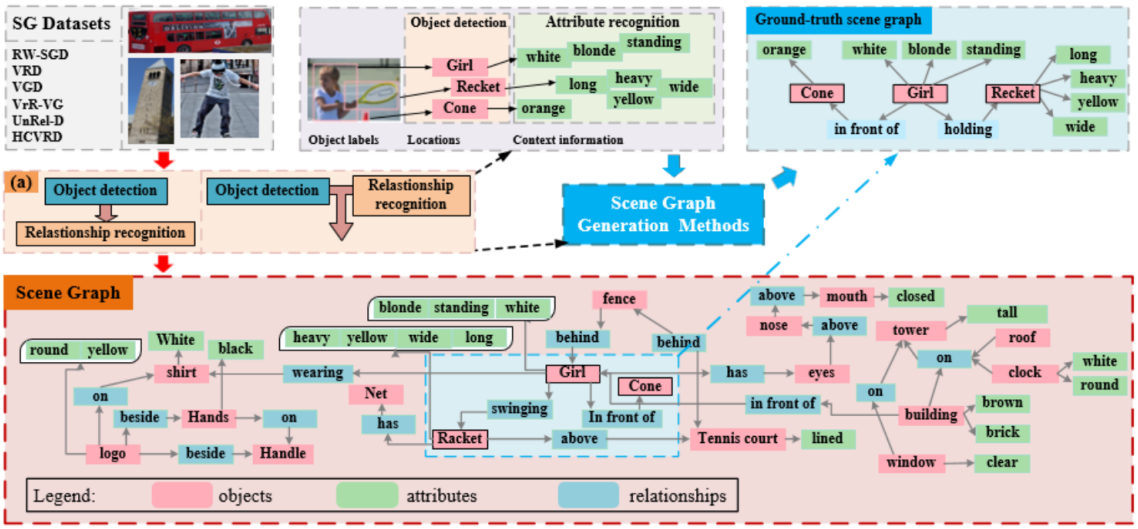


Figure 2.13: Example of scene graph construction [31]

We commonly process scene graphs using a graph convolutional network (GCN) [32], which is an architecture of neural networks oriented to process non-traditional non-grid-like data. We can manipulate the output of the network as we wish. A common output is a set of scores or probabilities for each node indicating the likelihood of belonging to a particular class, though more advanced approaches analysing different edge types can also be followed.

In terms of applications, they can be used in image and video captioning, visual question answering, scene understanding and video understanding. They can also be used to improve the performance of object detection and semantic segmentation. This project aims to use them as a means of storing and conveying additional information (spatial relationships) about a given prompt.

2.2.1.2 Generation

Scene graphs are typically generated through a pipeline of several steps, which include object detection, semantic segmentation, and relationship extraction:

- Object Detection: The first step in generating a scene graph is to detect the objects in the image or video. This can be done using a variety of object detection algorithms, such as Faster R-CNN, YOLO, or RetinaNet. These algorithms take an image or video as input and output a set of bounding boxes, each enclosing an object of a certain class.
- Semantic Segmentation: The next step is to perform semantic segmentation, which is the process of labelling each pixel in an image with a class label, such as "person", "car", or "tree". This step provides more detailed information about the objects in the scene, such as their shape and size.
- Relationship Extraction: After the objects and their properties have been identified, the next step is to extract the relationships between them. This can be done using various algorithms, such as graph convolutional networks (GCNs) or visual relationship detection (VRD) networks. These algorithms take the objects and their properties as input and output a set of relationships.
- Combination: Finally, the information from all the previous steps are combined to generate a scene graph, with the objects and their properties are represented as nodes, and the relationships between them are represented as edges.

It's worth mentioning that the accuracy of scene graph generation depends on the accuracy of the object detection and semantic segmentation, because if these steps aren't accurate, the relationships and attributes of objects will be incorrect.

2.2.1.3 Canonical Form

We can observe that generating realistic visual scenes becomes challenging as the complexity of the scene graph increases. One limitation of current models is the inability to capture semantic equivalence in graphs. For instance, an image with a relation "cat to the left of dog" also implicitly has the relation "dog to the right of cat" but usually only one is present. We need a canonical graph representation.

These graphs capture all relationships that exist in a given scenario, which is helpful when it's not obvious to deduce these. Figure 2.14 below outlines this, where given a simple input scene graph (a), we can arrive at (c) through simple logic. Now any model that (c) is passed to can build a wider understanding of the scene.

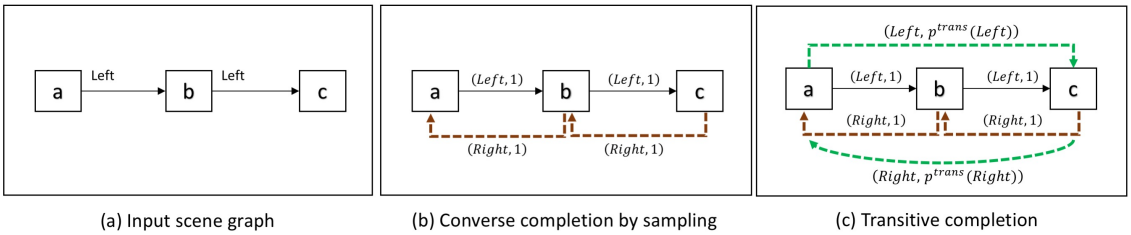


Figure 2.14: Canonical scene graph transformation [31]

2.2.1.4 Challenges and Limitations

Despite the advantages, multiple aspects make handling scene graphs challenging.

Generating a quality and correct scene graph from an image or a description can be challenging. Errors in the processes outlined in 2.2.1.2 can build up massively and lead to large inaccuracies. Another challenge is the inherent complexity and variability of scenes that can be represented. Scenes can contain numerous objects with various attributes, interacting in many different ways. Capturing all of this information in a scene graph can become difficult, especially when generalising.

Another significant limitation is that the generated images' quality heavily depends on the quality of the scene graph. If the scene graph is not accurate or does not capture all the necessary details, the generated image may not be as expected. To further this, we need large amounts of annotated training data to create models which can learn from this data.

We can circumvent some of these problems in this project by hand-crafting the uses of scene graphs. For instance, instead of designing a complex scene graph encoder, we can engineer a simpler representation to extract only the information we need. This could help any models focus more on the relationships we deem important.

Chapter 3

Overview of Approach

In the previous chapters, we explored the background of LDMs and the motivations for their use in this project. In this chapter, we will outline a novel method for building upon the current state-of-the-art solutions and conditioning diffusion models with scene graph information.

3.1 Problem Statement

The specific problem this project is trying to solve is how to inject additional information into the diffusion model image generation process, which can be used to learn and better understand the spatial relationships between the objects defined by the text prompt. Naturally, this spatial information resembles a graph structure, so we can make use of scene graphs to represent it. Therefore, the problem turns to: how can we incorporate scene graphs into the diffusion model architecture.

Breaking this down, here are some key criteria that any solution will need to meet:

Object Count Accuracy: The number of objects in the generated image should match the number of objects in the text prompt/scene graph.

Relationship Count Accuracy: The number of spatial relationships in the generated image should match those in the text prompt/scene graph.

Object Identity Preservation: The objects in the generated image should be clearly distinguishable.

Relationship Integrity: The relationships in the text prompt/scene graph are accurately represented in the generated images.

Scalable and Generalisable: The proposed method should be capable of handling many relationships and should be transferable across different datasets and models.

A solution which meets most, if not all of these objectives can be deemed to have successfully solved the project's aim.

3.2 Proposed Method

Based on all these criteria, this is the method we have come up with:

1. Create a custom dataset consisting of MNIST digits arranged in a grid shape, along with the corresponding text prompts and scene graphs.
2. Train a Stable Diffusion model using this dataset to act as a baseline to evaluate the performance.
3. Using ControlNet, train a new control model using the Stable Diffusion baseline and the scene graphs as conditioning input. Try two approaches for this:
   - Handcraft the scene graph into a tensor, and modify the ControlNet architecture to accept this as a control input.
   - Generate an image based on the scene graph information (essentially an annotator model) and use this as the control input.
4. Design an evaluation procedure and metric to score the generated images:
   (a) Create an MNIST digit classifier to analyse the generated images to detect the digits and relationships.
   (b) Construct an accuracy metric to score how well the relationships are respected in the generated images.

3.2.1 Architecture Diagrams

Let's visualise graphically the step of this project and provide some motivation:

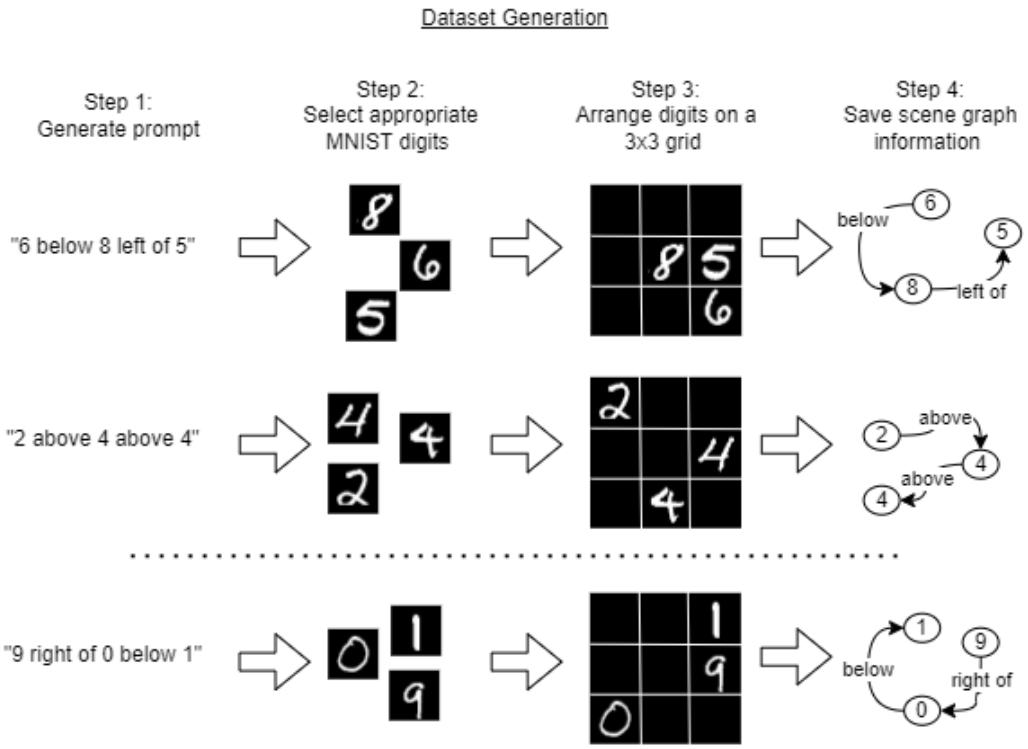


Figure 3.1: Dataset generation process

The motivation behind this project's approach is to focus the development work specifically on the question we are trying to answer. By extension, we want to reduce the complexity and number of external variables as much as possible, hence why we are creating a simpler dataset with MNIST digits arranged in a spatially determinant manner. This should be sufficient to provide a proof-of-concept solution.

Figure 3.1 provides an overview of how we generate the dataset. We use randomly crafted prompts following a pre-defined structure, and then select the appropriate MNIST digits, and arrange them on a 3x3 grid such that the relationships are met. We will discuss this process in more detail in Chapter 4.4.

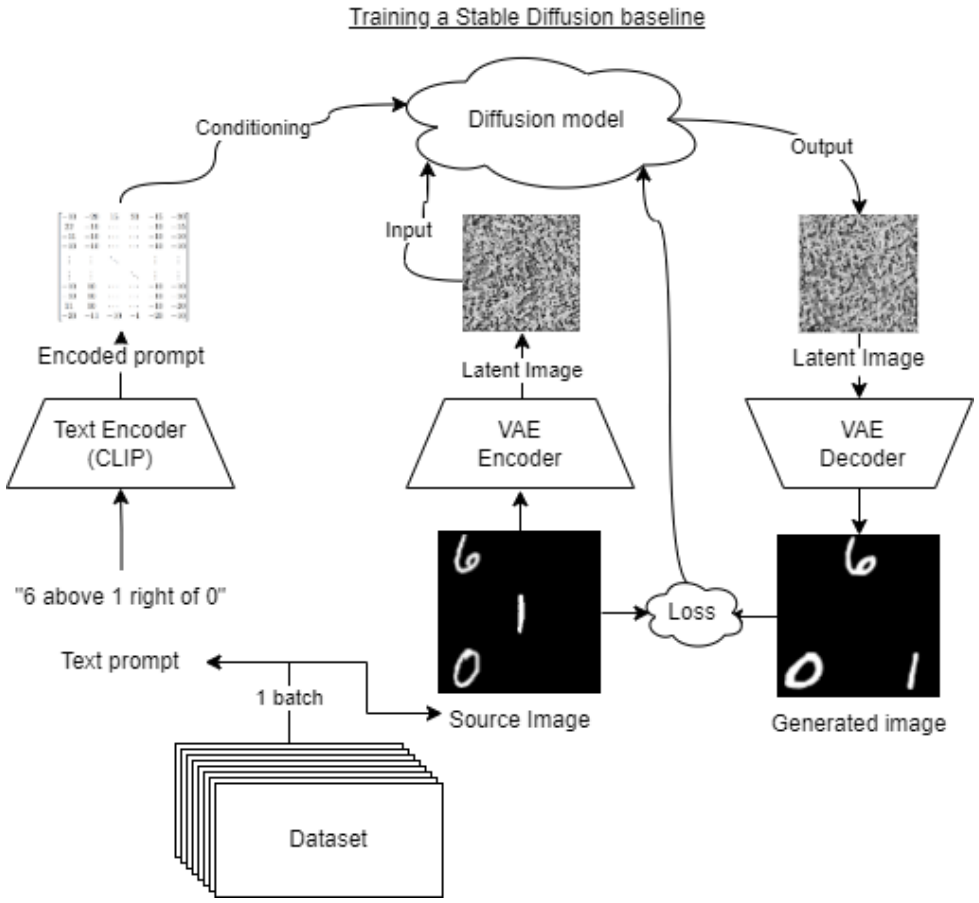


Figure 3.2: Stable Diffusion baseline training

To assess the improved spatial representation of our method, we require a performance baseline. This is achieved by training a Stable Diffusion model (see Figure 3.2) without architectural modifications, but with some configuration adjustments to reduce the training for this simpler task.

We sample from our dataset, taking batches of prompts and their ground truth images. These go through text and image encoders to create tensor representations, before being input to the diffusion model. The output is an image in the latent space, so after putting this through the image decoder, we can compare it with the ground truth to generate a loss and feed that back into the model.

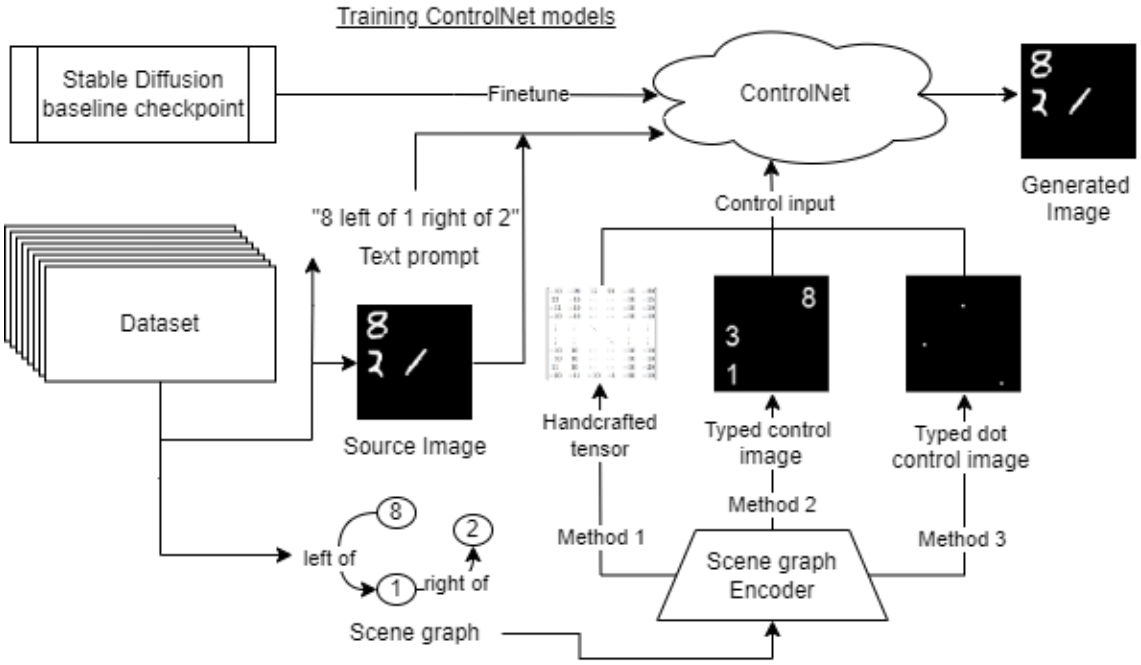


Figure 3.3: Training ControlNet models using 3 different methods

Next we have ControlNet, where we finetune the Stable Diffusion baseline model we have just created. Figure 3.3 shows this entire process.

We can experiment with different ways of modifying the architecture to inject the scene graph information into the diffusion model process. The "Scene graph Encoder" object in the Figure symbolises these different methods. Firstly we will try to encode the scene graph information in a custom tensor, then modify the ControlNet architecture to accommodate this. The idea behind this approach is to provide the spatial knowledge more directly in an attempt to reduce the difficulty for a model to pick this information up. Alternatively, we produce control images from the scene graph and train ControlNet accordingly. This method better aligns with ControlNet's intended use by providing images of the correct input dimension.

Finally, we need an evaluation process, which will need to be custom designed to focus on the spatial relationships in the generated images. This process is outlined in Figure 3.4.

Since we are using a fixed 3x3 grid structure of digits, we can train a classifier to identify the digits and their position in an image, and then calculate the all spatial relationships between them. This computed scene graph can be called the "Canonical scene graph" and we want to compare this with the ground truth scene graph to find which nodes and edges are contained. With this data, an accuracy metric can be designed and calculated for each ControlNet method. This can take into account existing/missing/surplus digits and relationships, and we can have several of these to focus on different aspects of the solution.

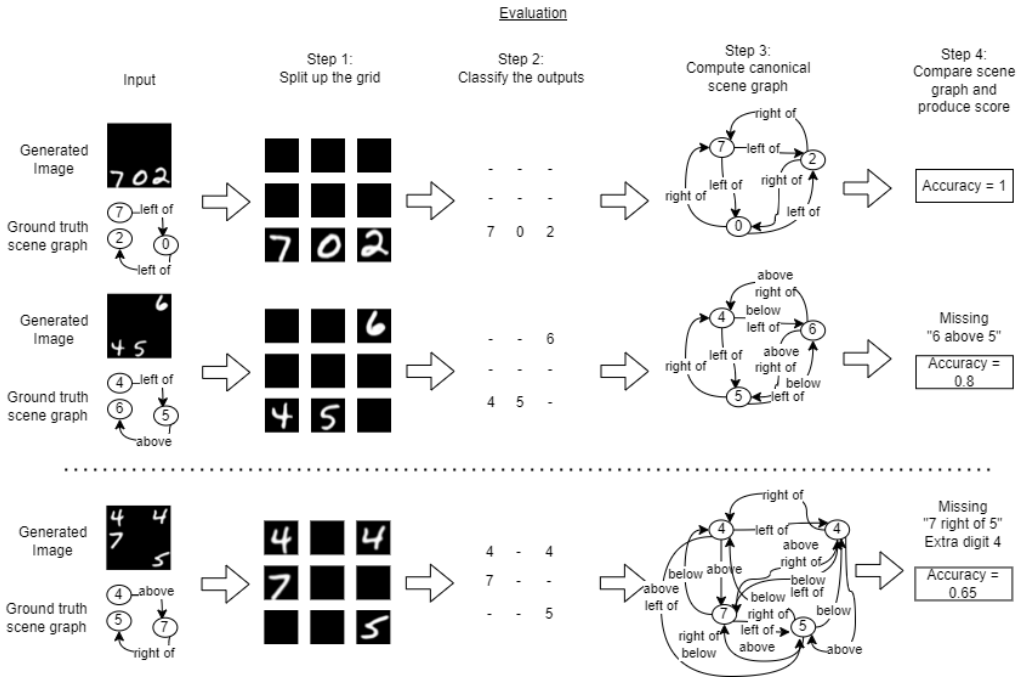


Figure 3.4: Evaluating generated images with accuracy metric

3.2.2 Method Justification

There are many reasons behind the designs we ultimately chose. We had to train an LDM from scratch since the Stable Diffusion checkpoints released by Stability AI have all been trained on very large datasets for many thousands of epochs. They have a large number of weights, leading to at least 7.2GB in size, and making it infeasible to work with on lower/mid-range GPUs. Additionally, these checkpoints are notoriously bad at producing images of text or digits, from the sparsity in the training data. But we couldn't fine-tune an existing model to understand digits since all finetuning methodologies work by training the model to associate a unique identifier with a certain object, not multiple identifiers for multiple objects.

Scene graphs were chosen because they provide a structured and intuitive way to represent spatial relations between objects in an image. Alternative representations could be used, such as semantic maps or direct text prompts, but scene graphs offer a good balance between expressiveness and simplicity. We didn't directly map them to images since this will be much more difficult to implement, requiring heavy modification to the U-Net structure. Also, we will be unlikely to incorporate these changes with existing Stable Diffusion checkpoints, thus requiring extensive additional training, unlike with the finetuning nature of ControlNet. We will assume the scene graphs are given, and will leave generating them as future work.

We selected to work with MNIST since its's a simple and well-understood dataset that allows us to focus on testing our methodology. We wanted to eliminate all other external variables to make it easier to identify changes made by the proposed method. There is nothing limiting the project's architecture from handling different datasets in principle, it would just need to be trained on these. When evaluating we aim to train a well-established and high-performing MNIST digit classifier since any inaccuracies in digit recognition could indeed affect our results.

Chapter 4

Technical Setup

This chapter delves into the project's technical foundation, outlining the setup processes performed before being able to train any models. We discuss the origins, structure, and limitations of the repositories used, the decisions made in scene graph structuring, data preprocessing techniques, and the modifications required for metric logging in the Stable Diffusion and ControlNet repositories. This overview offers insight into the infrastructure underpinning our experiments and findings.

4.1 Codebases

This project builds upon 2 main open-source repositories, Stable Diffusion and ControlNet. Let's discuss their general structure, required setup and any limitations.

4.1.1 Stable Diffusion

The repository is available on GitHub: `https://github.com/CompVis/stable-diffusion`. The repository includes all the code defining the VAE and diffusion model structure, code for training and testing both of these, as well as scripts for inference. Included also are several examples `.yaml` configuration (config) files for training the autoencoder and LDM, which describe the models evaluated in the Stable Diffusion paper [20]. These config files can be adapted to make the models smaller.

Setup

To get started, we can follow the instructions in the `README.md` to set up a conda/Python virtual environment using the packages in `environment.yaml`. We chose to use Python's `venv` module since it was simpler to get working on our development machines. To test the environment, we can run the `download_first_stages.sh` and `download_models.sh` shell scripts under `scripts/` to download and extract some pre-trained autoencoders and LDMs. Then we can run `txt2img.py` to run a loop through inference to generate some images based on a provided prompt. Successful generation will mean the environment is set up correctly.

Difficulties

The main difficulty of working with this codebase is the lack of documentation and instructions, be that through README files or comments in the code. There is no official guide to training models from scratch, nor are there any third-party sources due to the recent nature of this work. This meant that the workings of the codebase had to be understood ourselves from the ground up, by analysing the flow of execution and seeing which functions get triggered. Furthermore, despite the large number of pull requests and GitHub issues, there has been no activity from the authors on this repository since November 2022.

To add to the difficulty, PyTorch Lightning, a lightweight wrapper for the PyTorch library, was used by the authors for training, meaning that most of the usual training boilerplate code was masked away. This ultimately resulted in more upfront development time being spent thoroughly understanding the codebase's inner workings.

4.1.2 ControlNet

The repository is available on GitHub: https://github.com/lllyasviel/ControlNet. Similar to the Stable Diffusion repository, this repository includes code defining the VAE and LDM's structure, but we also get the code contributed by the ControlNet paper [30]. This includes a `ControlledUNetModel` and a `ControlNet` class under `cldm/`, and an updated `DDIMSampler` class to incorporate control inputs.

The pre-trained ControlNet models are available on HuggingFace: https://huggingface.co/lllyasviel/ControlNet. The authors have also included a gradio-based inference script for each of the pre-trained models. This provides a simple yet proficient interface for using and configuring these control models.

Despite the lack of comments in the code, there is importantly a first-party training tutorial, where the authors outline how to train a new control model on top of the Stable Diffusion v1.5 checkpoint. This is explained in `docs/train.md`, and many scripts are provided to aid the training process: `tool_add_control.py`, `tutorial_train.py` and `tutorial_dataset.py`.

Setup

Setup is similar to with Stable Diffusion, there is an updated `environment.yaml` file containing the packages required for this codebase. We chose to install these under a new Python virtual environment in case of any package conflicts. Next, we download the `control_sd15_canny.pth` model then ran the `gradio_canny2image.py` file to test out the Canny edge detection control model.

Difficulties

Compared to Stable Diffusion, this repository was much easier to understand and build upon, though a lot of this might be due to the existing familiarity with Stable Diffusion at this point. However, we did need to add metric logging, checkpointing and modify the Image logger to be able to record and save the trained models. Fortunately, we could adapt the Stable Diffusion code to achieve most of these.

4.2 Development Machines

Throughout development several different machines were used to train, test and evaluate the models explained in Chapter 5. The most important computing resource required was GPU VRAM, to facilitate larger batch sizes and reduce overall training/sampling times. The specs for the machines used are shown in Figure 4.1.

CPU	RAM	GPU	VRAM
Intel Xeon W-2145 16 core 3.70GHz	32GB	NVIDIA RTX 2080 Ti	12GB
Intel Xeon W-2123 8 core 3.60GHz	32GB	NVIDIA GTX Titan X	12GB
Intel Core i7-7700K 8 core 4.20GHz	16GB	NVIDIA GTX 1080	8GB
Intel Xeon ES-1620 8 core 3.70GHz	16GB	NVIDIA Quadro P4000	8GB
Intel Xeon Silver 4116 24 core 3.00GHz	256GB	NVIDIA Tesla A30	24GB
AMD EPYC 7702P 64 core 3.35GHz	256GB	NVIDIA Tesla T4	16GB
Intel Xeon Silver 4116 24 core 3.00GHz	256GB	NVIDIA Titan XP	12GB

Table 4.1: Specifications of machines used throughout development

When training larger models near the beginning of the project, it was beneficial to use machines with large GPU VRAMs since CUDA Out Of Memory (OOM) Error" was a common occurrence with batch sizes over 4. However, most of the training was performed on the Nvidia RTX 2080 Ti machines. We did observe a 1.5-2x performance increase when sampling using the 16-core machines, arguably due to the more CPU-heavy workload of switching between images to sample.

4.3 Metric Logging with Weights & Biases

Weights & Biases (wandb) is a powerful tool designed to help track and visualise progress during the training of machine learning models. It offers a platform to effortlessly monitor metrics like loss values over extended training periods, thereby enabling swift issue identification and resource optimisation. The wandb workspace for this project is accessible `https://wandb.ai/krishan3168/final-year-project`.

wandb was integrated into both the Stable Diffusion and ControlNet models. In the Stable Diffusion model, the existing setup allowed easier logging of metrics by shifting to wandb as the default logger. In addition, the `ImageLogger` object was modified to correctly denormalise and log sampled images while training.

For ControlNet, the wandb integration was more rudimentary but followed similar principles. Wandb logging was initiated, and the `ImageLogger` object was amended to include the `_wandb()` function. Control images were also logged to aid the monitoring of training with visualisations.

Overall, integrating wandb into our workflow has greatly streamlined the process ofmonitoring and debugging our models. It has provided a level of transparency andease that would have been challenging to achieve with traditional logging methods.

4.4 Dataset Generation

This section discusses in more detail how we generated the dataset for our experiments. Figure 3.1 from Chapter 3.2.1 illustrates an overview of the process, which can be summarized in three steps: the crafting of random prompts, selection and arrangement of MNIST digits, and preprocessing of the image data.

4.4.1 Scene Graph Structure

Our scene graph structure is determined by four cardinal relationships - "above", "below", "left of", and "right of". We have chosen these relationships because they are intuitive, applicable, and fundamental to any spatial layout. These relations also map clearly onto a 2D image plane, making them a natural choice for arranging MNIST digits on a grid.

The prompts we craft follow a predefined structure where digits are in specific relationships with other digits. The left-associative relationships take the form:

$$\textit{digit} \quad \textit{relationship} \quad \textit{digit} \quad \textit{relationship} \quad \textit{digit} \tag{4.1}$$

where:

- *digit* can be one of $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$
- *relationship* can be one of $\textit{above}, \textit{below}, \textit{left of}$, or $\textit{right of}$
- $(\textit{digit}, \textit{relationship}, \textit{digit})$ is a spatial relationship triplet

Examples of prompts include: "3 left of 2 above 5", "4 below 5 below 1" and "9 right of 0 above 0". For simplicity's sake, we have chosen to investigate with 3 digits and 2 relationships, forming 2 spatial relationship triplets, however this architecture can easily support more.

4.4.2 Generation

To generate the dataset, we developed a Python script that implements the above-mentioned procedure. The script (under `scripts/mnist_dataset_gen.py`) starts off by downloading the train and test MNIST dataset available in `torchvision.datasets.MNIST`. Next, we create a random prompt with 3 digits by randomly selecting 3 digits from 0-9 and randomly selecting 2 of the 4 available spatial relationships. Combining these together in our prompt structure gives us our random prompt.

Now we need to create an image representing the prompt we have. For this, we will use a simple 3x3 grid representation as this will provide a sufficient means of depicting all of our relationships Since each MNIST image is 28x28 pixels, this grid will be 84x84 pixels in dimension.

A rule we introduced at this stage was that to obey a spatial relationship, you cannot be on the same level horizontally or vertically. Given we have "2 below 6 left of 3", the 2 and the 6 cannot both be in the same row nor can the 6 and the 3 be in the same column. This was decided to make the relationships less ambiguous.

We tried developing a mathematically optimal solution to this problem but, the naive approach ended up being easier to understand and quicker to execute overall, so let's discuss that approach. Figure 4.1 shows a possible sequence of execution.

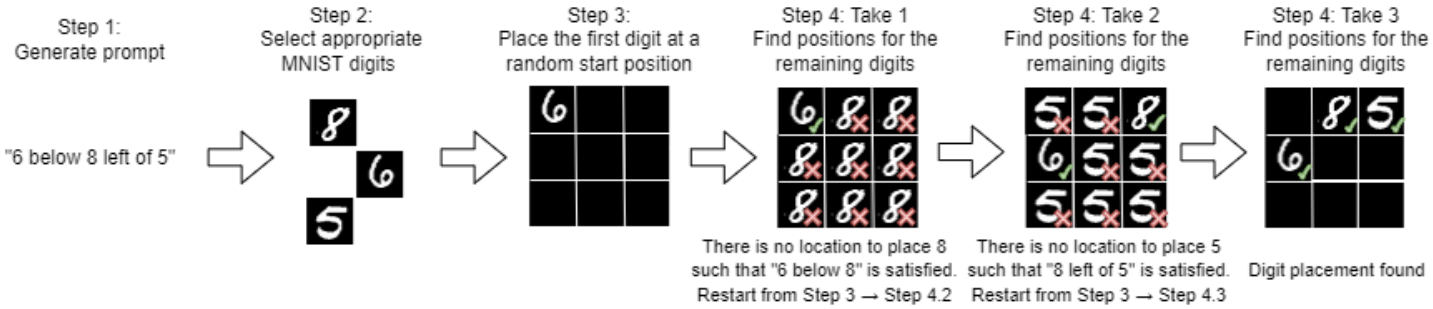


Figure 4.1: Digit placement process in image generation

First, we use the MNIST dataset object to randomly pick images of the digits we have in our prompt. Next, we randomly select a start position and place our first digit there, in the example shown in Figure 4.1 it would be 6. Then, we calculate all the possible positions where we can place our next digit such that the spatial relationship holds. If this list of positions is empty, then we conclude that the current placement of digits cannot be taken forward, and we start again with randomly placing the first digit. This is what happens in Figure 4.1 in Step 4 Take 1, since there is no square where 8 can be placed. Otherwise, if there are positions, then we place the digit and continue. An infinite loop is not possible here since any combination of 3 digits and 2 relationships can be represented in a 3x3 grid.

This stage of dataset generation is critical as it directly affects the quality and diversity of the training data. Ensuring a robust and diverse dataset is essential to train our models effectively. We generated a training dataset of size 9000, a validation dataset of size 1000 and a test dataset of size 1000, all of greyscale images.

Scene graph data is saved textually in the filename of each generated image. A filename comprises a unique counter, the prompt, and a grid representation where '-' signifies empty space. For instance, "counter_6 below 8 left of 5_-856——.png" represents the image in Figure 4.1. Duplicates are permitted as each prompt can be correctly represented in multiple ways with different MNIST images too.

4.4.3 Preprocessing

Before we can train models using our datasets, we first need to pre-process the images. This involves resizing, type conversion, and normalisation. We first upscale the 84x84 pixel images to 128x128 pixels using bicubic resampling, which enhances image detail and optimises memory use. Although we experimented with 256x256 pixel upsampling, we found the resulting model size excessive. The images are then transformed from the PIL.Image object into a PyTorch tensor. Following this, we normalise pixel intensity values from the range [0,255] to [-1,1] to accelerate convergence speed. An important consideration is to ensure consistency in normalisation and denormalisation. Inaccurate denormalisation can falsely present an output as incorrect. Initially, we normalised images to the range 0-1, but further debugging and codebase analysis revealed the need for the range to be [-1,1].

Chapter 5

Implementation

Chapter 5 delves into the detailed aspects of our implementation journey. Here, we dissect the processes, decisions and struggles faced in training our baseline model, followed by an in-depth analysis of our modifications to ControlNet. The chapter offers a practical lens on our methodology and provides the foundation for the forthcoming evaluation discussion.

5.1 Training the Baseline

Let's begin by discussing the process of training the Stable Diffusion baseline model. Figure 3.2 from Chapter 3.2.1 illustrates the overall method.

5.1.1 Motivation

Briefly, let's discuss the motivation for training the baseline from scratch as opposed to using one of the finetuning mechanisms explained in Chapter 2.1.3.1.

The pre-trained Stable Diffusion models struggle with producing text and digits in images. Figure 5.1 shows the completely unrelated images which are produced when asking about digits. Due to this behaviour, we need to teach Stable Diffusion about digits and relationships but due to the unique identifier approach in finetuning, we cannot enforce the distinction between each digit and relationship. Henceforth, training from scratch is the only realistic way for us to teach it about digits.

Figure 5.1: Stable Diffusions bad performance on digits

5.1.2 VAE

When starting to train the VAE, our first steps were to check the performance of pre-trained autoencoders released by Stability AI.

5.1.2.1 Pre-trained Models

Even though these models were designed for three channel images and our dataset was only greyscale (single channel) to reduce model complexity, a sufficiently well-performing autoencoder would have a superior latent space representation, and save development time. We tried the 4 KL autoencoders models downloaded by `scripts/download_first_stages.sh`, but their performance was poor on our dataset.

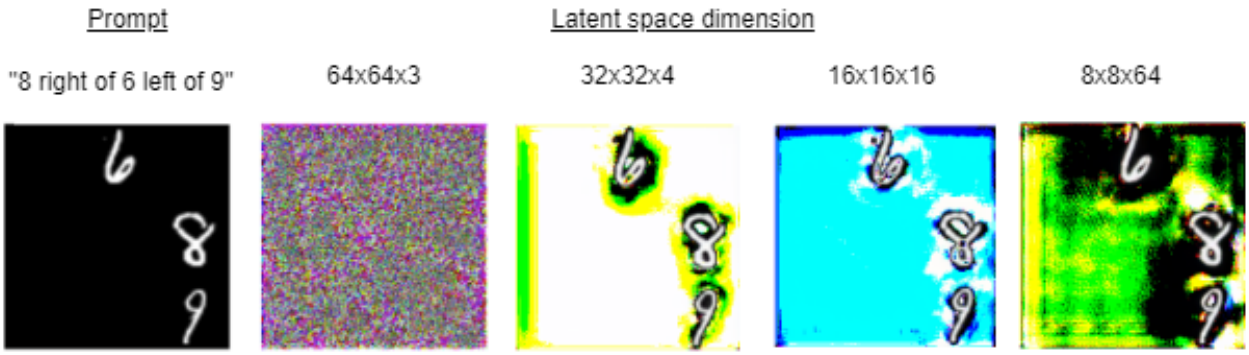


Figure 5.2: Pre-trained autoencoder performance on our dataset

Figure 5.2 shows the result of applying one of our dataset images on 4 pre-trained autoencoder models. The result should show a replication of the input, and so should ideally be indistinguishable from the input. However, it ranges from noise to multi-coloured backgrounds, all with a very high reconstruction loss. The poor performance could be explained due to the original ImageNet datasets not having many greyscale-like images to learn from. Regardless, we need to train from scratch.

5.1.2.2 Configuration Files

As explained in Chapter 4.1.1, the autoencoder is trained using a config file to specify all the specific parameter values. Figure 5.3 provides an example of this. The Stable Diffusion codebase includes the definition of 2 types of autoencoders: KL Autoencoder (using KL loss) and VQ Autoencoder (using the VQ approach which brings a discrete latent space representation). We chose to use the KL Autoencoder to focus on having both good image reconstruction and latent space representation.

Whilst trainig the autoencoder, we went through many iterations of hyperparamter tuning and debugging the codebase to build a better understanding of the inner workings. We knew we could decrease the size of the autoencoder to have a smaller model overall since we had less image data to encode. Let's discuss what each configuration parameter controls:

`base_learning_rate`: The initial learning rate for the optimizer. A critical parameter that determines how much the weights are adjusted during training.

`monitor`: The metric to monitor for model performance and focus the loss on

```yaml
model:
    base_learning_rate: 4.5e-6
    target: ldm.models.autoencoder.AutoencoderKL
    params:
        monitor: "val/rec_loss"
        embed_dim: 4
        lossconfig:
            target: ldm.modules.losses.LPIPSWithDiscriminator
            params:
                disc_start: 10001
                kl_weight: 0.1
                disc_weight: 0.5
                disc_in_channels: 1
        ddconfig:
          double_z: True
          z_channels: 4
          resolution: 128
          in_channels: 1
          out_ch: 1
          ch: 32
          ch_mult: [ 1,2,4,4 ]
          num_res_blocks: 2
          attn_resolutions: [ ]
          dropout: 0.1
          tanh_out: True
```

Figure 5.3: Autoencoder configuration file achieving best performance

`embed_dim`: Size of the embedding vector. Dimensionality of Encoder's output

`disc_start`: Specifies the epoch number at which the discriminator loss will start to be considered in training.

`kl_weight`: Weight balancing KL divergence loss with reconstruction loss.

`disc_weight`: Weight balancing importance of discriminator loss with other losses.

`double_z`: Indicates whether to double the latent variables during training.

`z_channels`: Number of channels in the latent space.

`resolution`: Spatial resolution (height and width) of the input images.

`in_channels`: Number of input channels to the model.

`out_ch`: Number of output channels from the model.

`ch`: Base number of channels for the convolutional layers in the model.

`ch_mult`: Multiples applied to the base channel number (`ch`) for each layer in the model. This determines the growth of channels in the model's layers.

`num_res_blocks`: Number of residual blocks in the model, helping to alleviate the vanishing gradient problem in very deep models.

`attn_resolutions`: Specifies the resolutions at which to apply self-attention in the model. An empty list implies no self-attention is applied.

`dropout`: This is the dropout rate, a regularization technique to prevent overfitting.

`tanh_out`: Indicates whether to apply a tanh activation function to the model's output. This ensures that the output values are between -1 and 1.

5.1.2.3 Early Models' Results

We began by choosing a resolution of 256, a base channel number of 64 and sticking with the defaults from Stable Diffusion for the rest. We also chose to name the models we trained sequentially for ease of analysis and comparison.

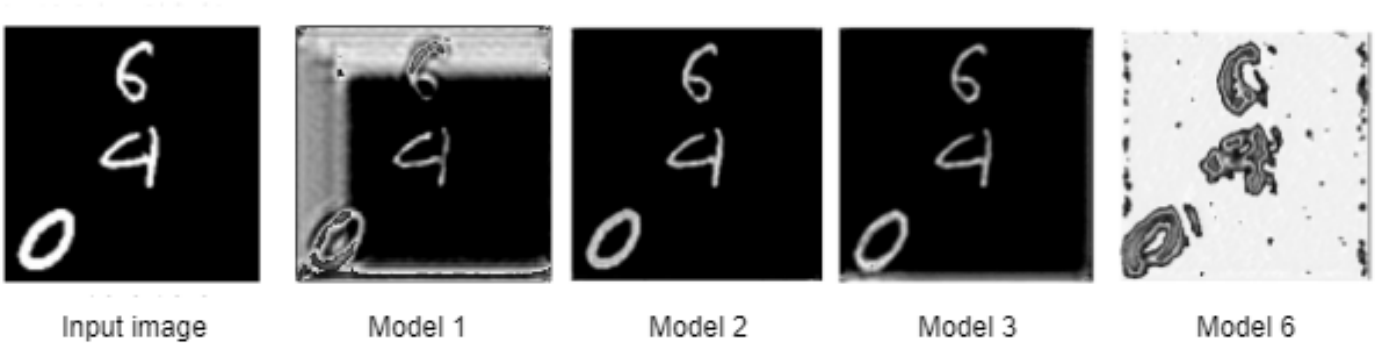


Figure 5.4: Performance of early autoencoder models

These models took an average of 20 minutes per epoch for our training set, but unfortunately did not produce good results, as shown in Figure 5.4. As evident, the reconstruction loss for all these was very high. During this time, the `lossconfig` parameters where set to their default of `disc_start: 50001`, `kl_weight: 1.0e-06`, `disc_weight:0.5` (a future bug addressed in 5.1.2.5).

The issue here was an overall complex model and incorrect pixel normalisation. Images passed to the encoder were not between [-1,1] but were either mistakenly between [0,1] or [-0.5,0.5]. We addressed this, and reduce the resolution to 128 and base channels to 32 to reduce model intricacy.

5.1.2.4 Output Layer Bug

Fixing the previous problem resulted in much more recognisable outputs with a lower reconstruction loss of around 0.022 (for model 9) instead of the previous 0.58 (for model 6). We can see this improvement in Figure 5.5 however, there is still a noticeable problem with producing spotty/dotted reconstructions. We attempted many configurations, even trying 3-channel RBG in model 12, but to no avail.

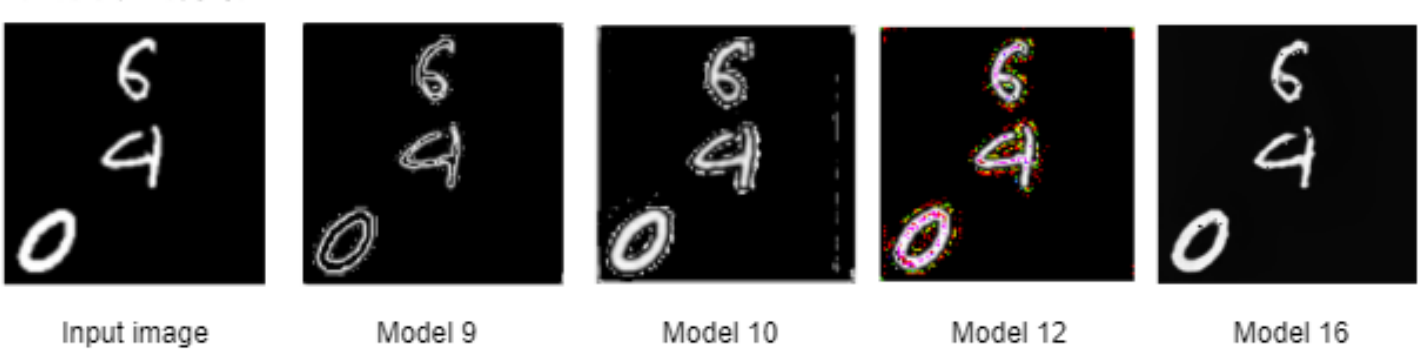


Figure 5.5: Unusual reconstruction behaviour with spots

After meticulous debugging, it was found this behaviour was caused by the lack of an activation layer on the model's output layer. The code was setup to perform a tanh activation however the respective config parameter was set to `False`, and so this operation was ignored. Once again, the lack of this layer caused a normalisation issue, where the latent space representation is not in the range [-1,1].

5.1.2.5 Balancing KL and Reconstruction loss

Following the previous fixes, we were now achieving very good performance, with reconstruction loss in the low 0.01 region. The problem now was with an extremely high KL loss (see Figure 5.6) which was not decreasing over time.

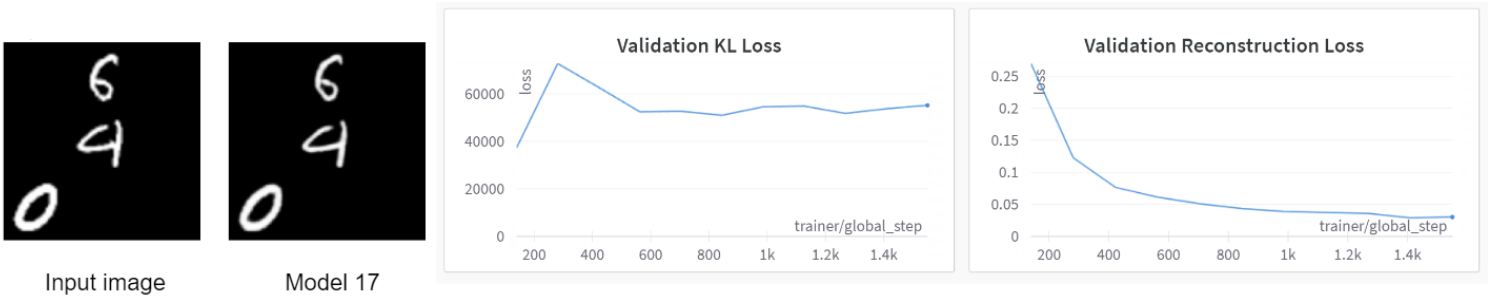


Figure 5.6: KL and Reconstruction loss for model 17

It is important to have a low KL loss since this metric describes how well the latent space is structured. Since the VAE will be used in training the LDM, a bad latent space representation can cause implications going forward. We found this behaviour was being caused by the very low `kl_weight` of 0.000001, telling the model to prioritise a lower reconstruction loss at the cost of KL loss.

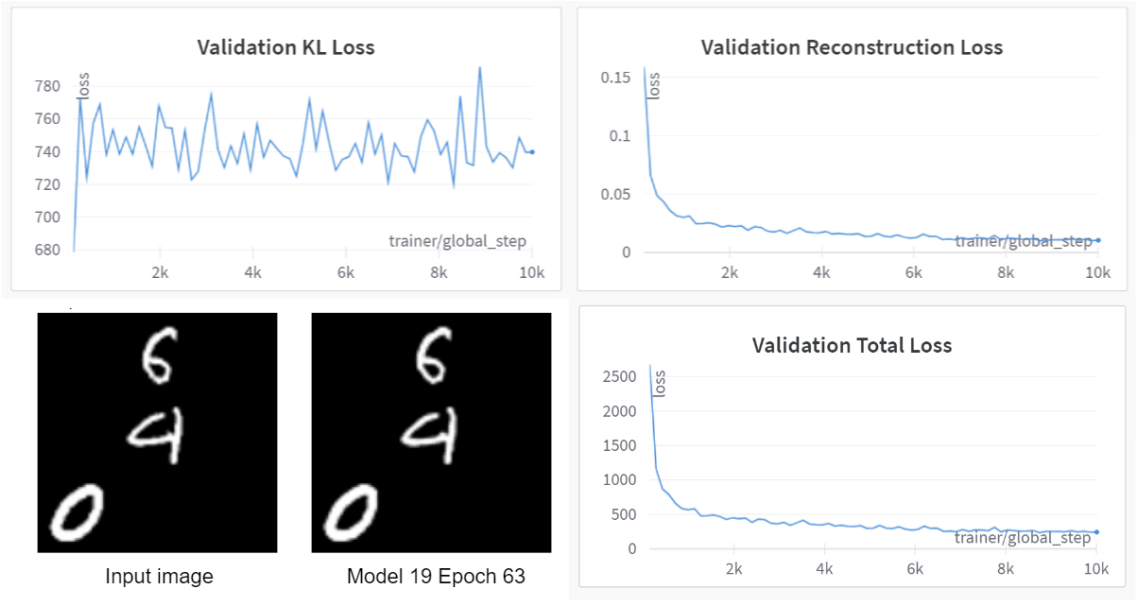


Figure 5.7: Best autoencoder (model 19) performance on validation set

We tuned this and the other `lossconfig` as hyperparameters, and found that moving `kl_weight` closer to 1 yielded significantly lower KL loss values, whilst keeping the reconstruction loss similar to before. We kept tuning until reaching Model 22, experimenting with `kl_weight` of 0.001, 0.1, 0.25, 0.5, 1, and finding the reconstruction loss to range between [0.009, 0.035] and the KL loss between [250, 5000]. We settled on `kl_weight`=0.1 (Figure 5.7) as it provided the best middle ground.

5.1.3 LDM

With our performant autoencoder, we can now embark on training the LDM. For encoding text, we will be using a frozen CLIP model 7.1.1.1. As before, we will be tuning the LDM parameters through a config file.

5.1.3.1 Configuration Files

Let's discuss the important parameters of the LDM config file shown in Figure 5.8.

```yaml
model:
  base_learning_rate: 5.0e-05
  target: ldm.models.diffusion.ddpm.LatentDiffusion
  params:
    linear_start: 0.00085
    linear_end: 0.012
    num_timesteps_cond: 1
    log_every_t: 200
    timesteps: 1000
    first_stage_key: image
    cond_stage_key: caption
    image_size: 16
    channels: 4
    cond_stage_trainable: false
    conditioning_key: crossattn
    monitor: val/loss_simple_ema
    use_ema: true

    cond_stage_config:
      target: ldm.modules.encoders.modules.FrozenCLIPEmbedder

    unet_config:
      target: ldm.modules.diffusionmodules.openaimodel.UNetModel
      params:
        image_size: 16
        in_channels: 4
        out_channels: 4
        model_channels: 64
        attention_resolutions: [4, 2, 1]
        num_res_blocks: 2
        channel_mult: [1, 2, 4]
        num_heads: 8
        use_spatial_transformer: true
        transformer_depth: 1
        context_dim: 768
        use_checkpoint: true
        legacy: False

    first_stage_config:
      target: ldm.models.autoencoder.AutoencoderKL
```

Figure 5.8: LDM configuration file for LDM model 15

`linear_start`, `linear_end`: Parameters for linear schedule used in LDM training.

`timesteps`: Number of timesteps in the forward diffusion process.

`first_stage_key`, `cond_stage_key`: Keys used in dataset for image and label.

`image_size`: The height and width of the input images to the model (latent space).

`channels`: Number of channels of latent space image.

`conditioning_key`: Either crossattn, concat, hybrid or adm for conditioning.

`use_ema`: Indicates whether to use Exponential Moving Average for model weights.

`model_channels`: Base number of channels in model, multiplied by channel_mult.

`channel_mult`: Specifies U-Net layer architecture in terms of model_channels.

`context_dim`: Dimensionality of the context vectors (i.e. text encodings).

5.1.3.2 Unstable Training

A hyperparameter we immediately tuned was `model_channels` since it controls the number of layers in the overall model, and thus has the largest impact on the model's complexity and training time. Training with `model_channels`=320 constitutes to 859MB of trainable parameters just in the diffusion model. Furthermore, it wasn't possible to train with this configuration on any of our development machines, even with a batch size of 1. We experimented with values of 160 and 96 but experienced trouble with the loss rising to and staying at 1 after some number of epochs.

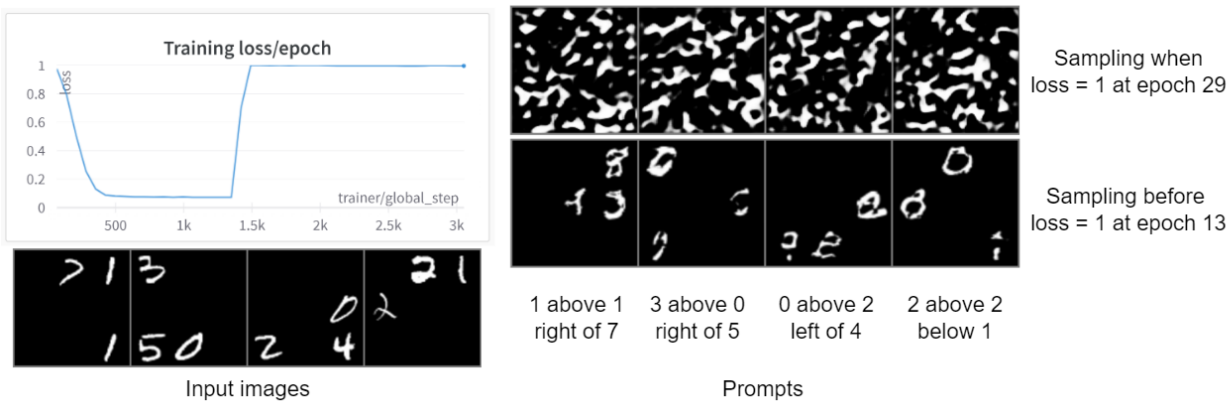


Figure 5.9: LDM loss hitting 1 and generating meaningless samples afterwards

Figure 5.9 shows the samples generated at epoch 13 and 28 of an LDM model. We have the ground truth images and conditioning text which are fed as inputs shown across the bottom. The generated samples at epoch 13 exhibited structural similarities and distributions that closely resembled the input data. However, by epoch 29, the model appeared to create predominantly noise-driven outputs. We found the cause of this behaviour to be a learning rate schedulur setting the learning rate too high far into the training. From then we used a fixed learning rate of 0.00005.

5.1.3.3 Training Smaller Models

Fortunately, after this fix, we arrived at a satisfactorily performant model trained for 30 epochs over 36 hours. The loss converged around 0.34 and the performance of this model on some prompts is shown in Figure 5.10. Although we now had a working model, we noticed that the size of it was quite large, 3.82GB.

This is problematic since ControlNet copies and trains upon Stable Diffusion layers, so if the baseline model is already quite large, the ControlNet models will be more difficult to train. To address this, we trained additional baseline models by decreasing the `model_channels` and the layers in `channel_mult`.

Our model number	model_channels	channel_mult	Epoch number	Training time	Size (GB)
Stable Diffusion	320	[1,2,4,4]	-	-	7.17
6	160	[1,2,4,4]	30	36 hours	3.82
12	64	[1,2,2,4]	152	33 hours	0.931
15	64	[1,2,4]	181	26 hours	0.983

Table 5.1: Architecture and size of baseline LDM models

Figure 5.10: Trained LDM model 6 sample results and loss curve

Table 5.1 shows our 3 best trained LDMs and the most important parts of their architecture. Model 6 used a batch size of 1 whereas models 12 and 15 used a batch size of 4. We will evaluate these models and compare their performance in Chapter 6, but briefly, models 12 and 15 outperform model 6 whilst being 4x smaller in size.

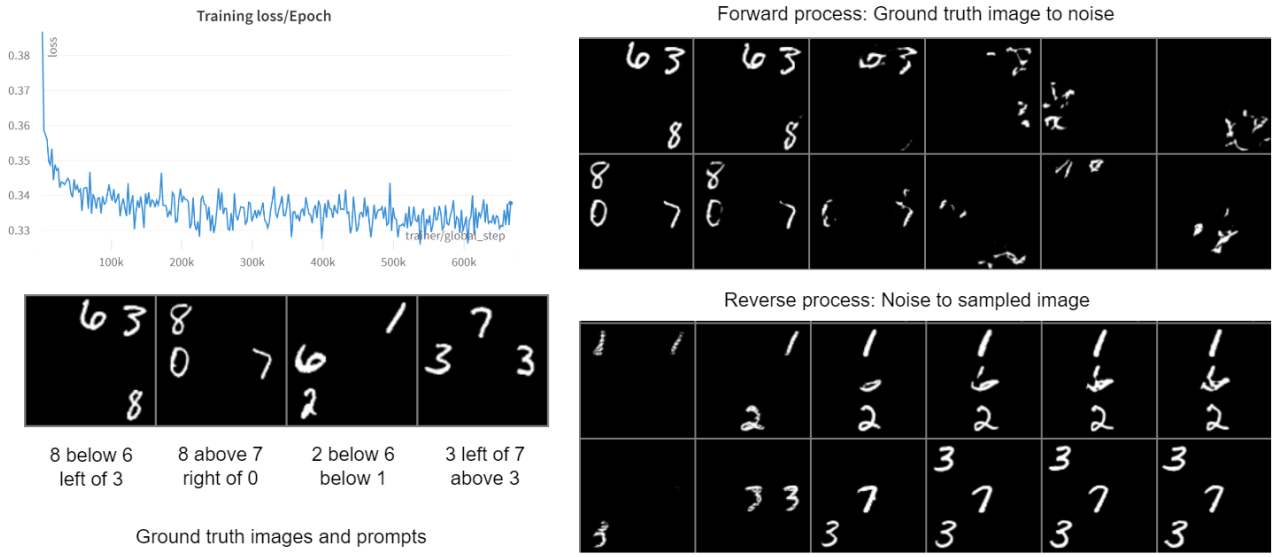


Figure 5.11: Trained LDM model 12 with forward/reverse diffusion process samples

Figure 5.11 above provides the training loss curve for model 12 together with images showing the execution steps of the forward and reverse diffusion process. Firstly, we can clearly see a conventional converging trajectory for the training loss over our large number of epochs. We can consequently be confident that there will be little benefit in training this model further.

The rows of images at the top show the forward diffusion process of taking the first 2 ground truth images and converting them to noise. The bottom rows show the reverse process and how images for the last 2 prompts are generated over several steps. We can see how the images slowly morph over time as the diffusion model applies its transformations to the noisy images.

5.2 Training ControlNet

As a recap, Figure 3.3 from Chapter 3.2.1 illustrates the overall method. To aid in training ControlNet models we had some additional setup to be performed.

5.2.1 Setup and Configuration Files

Firstly, we added the `ModelCheckpoint` callback from PyTorch Ligthning to save the 10 best checkpoints on a particular run. Without this, the script only keeps the last trained model. Next, we created Dataset objects (in `mnist_control_dataset.py`) to provide access to the dataset files and labels in a structured format. This class is also responsible for creating the control signals. Finally, we added a data module parser to the ControlNet configuration so the training, validation and test datasets could be specified in the config file, like with Stable Diffusion, instead of in the code.

The configuration files were largely similar to Stable Diffusions (Figure 5.8), except with an additional `control_stage_config` section for defining the specific ControlNet architecture. This will follow the same values trained on the baseline Stable Diffusion checkpoint being finetuned.

5.2.2 Handcrafted Scene Graph Tensor

The idea here is to convert the scene graph information into a custom representation which doesn't resemble an image. Let's discuss its structure:

5.2.2.1 Tensor Structure

As explained in Chapter 4.4.1, we have 4 spatial relationships and 10 possible digits. Each relationship triplet is formed of 2 digits and 1 relationship. Based on this we created a tensor with the shape 4x10x10 to represent this information. Figure 5.12 provides a visualisation of this 3D tensor, with filled green cubes representing values of 1.

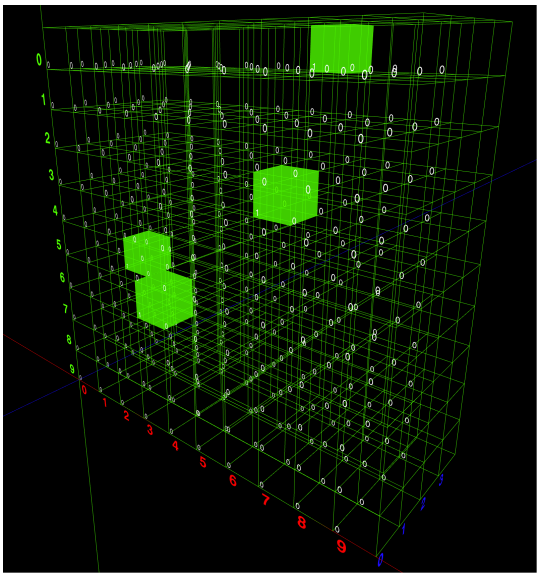


Figure 5.12: Tensor depicting canonical form of scene graph: "3 right of 6 above 0"

In this tensor, all other values are filled with 0, making it quite sparse. In the visualisation, the green and red axis represent digits 0-9 and the blue axis represents the spatial relationships with {0: "left of", 1: "right of", 2: "above", 3: "below"}.

While the 3D image may be difficult to interpret fully, one clear observation would be the existence of 4 values of 1 (green cubes) instead of just the 2 obtained from a given prompt. This is because we have represented the canonical form of the scene graph, while still respecting the local relationships, i.e. not allowing transitive relationships. This decision was made to keep the results clearer and simpler to analyse and evaluate, especially with our small grid space.

5.2.2.2 Projecting Tensor

Next, we modified the ControlNet architecture to accept the different control inputs we have created. We started this with a simple approach of using linear layers to project the 4x10x10 tensor to 4x128x128. This new shape was calculated from the ControlNet architecture's downsampling rate of 3, meaning to arrive at the latent space representation of 4x16x16, we needed to start off at $16 \times 2^3 = 128$, 4x128x128.

We flattened the 4x10x10 tensor into a 1D tensor of shape (400), then applied a linear layer projecting from $400 \rightarrow 65536$ ($4 \times 10 \times 10 = 400 \rightarrow 4 \times 128 \times 128 = 65536$), then reshaped this now 1D tensor with shape (65536) to 4x128x128. This linear layer contained $(400 \times 65536) + 65536 = 26,220,800$ parameters when adding the count of its weights and bias values, which constitutes to 100MB in additional size of the model.

We attempted a multi-linear layer approach ($400 \rightarrow (4 \times 64 \times 64 =) 16384 \rightarrow 65536$) however this quickly because infeasible as it required 1,080,328,192 parameters and 4.12 GB of additional space. We trialled several further approaches (e.g. GCNs in Chapter 2.2.1.1) but eventually settled on the technique explained in Chapter 5.2.3

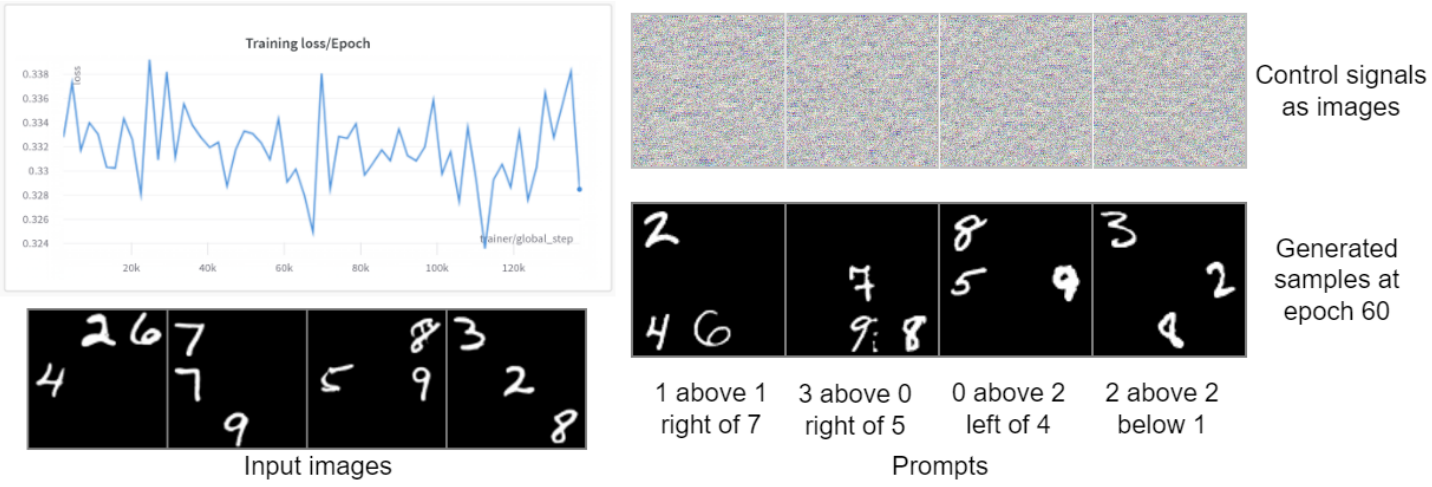


Figure 5.13: Trained ControlNet model with loss curve, samples and control images

Figure 5.13 shows results after 60 epochs of training using a single linear layer approach. The control images are the denormalised forms of the projected scene graph tensor. As we can see, it appears as just random noise and we will discuss the implications of this further during the evaluation.

5.2.3 Image-based Control Signals

5.2.3.1 Motivation

The main issue with the linear layer projection approach was that ControlNet was designed for image data, and so has strong biases towards local spatial information. It may be better at providing local information than a global structure. We however want to condition on global spatial information so we need to first translate this into local information. We can approach this using a similar strategy to prior work by following a 2 step process:

1. Apply an annotator model to recognise and process the spatial relationships in a given prompt, and predict where the digits will be placed. Create an image based on the arrangement.
2. Use the created image as the control signal to train the ControlNet model.

This means the task of learning how spatial relations are related to the image is delegated to some external, pre-trained model. We can handcraft this layout image generation process for now, to provide a proof-of-concept for this approach when using ControlNet alongside. Let's try two methods for this:

5.2.3.2 Typed Digits as Control

We start with control signals as typed digits (see Figure 5.15 for an example). For the images in the dataset, we generate these by replacing the MNIST digit images with ASCII versions of the digits, in the corresponding locations. At inference time, we employ the same method described in Chapter 4.4.2 to generate these.

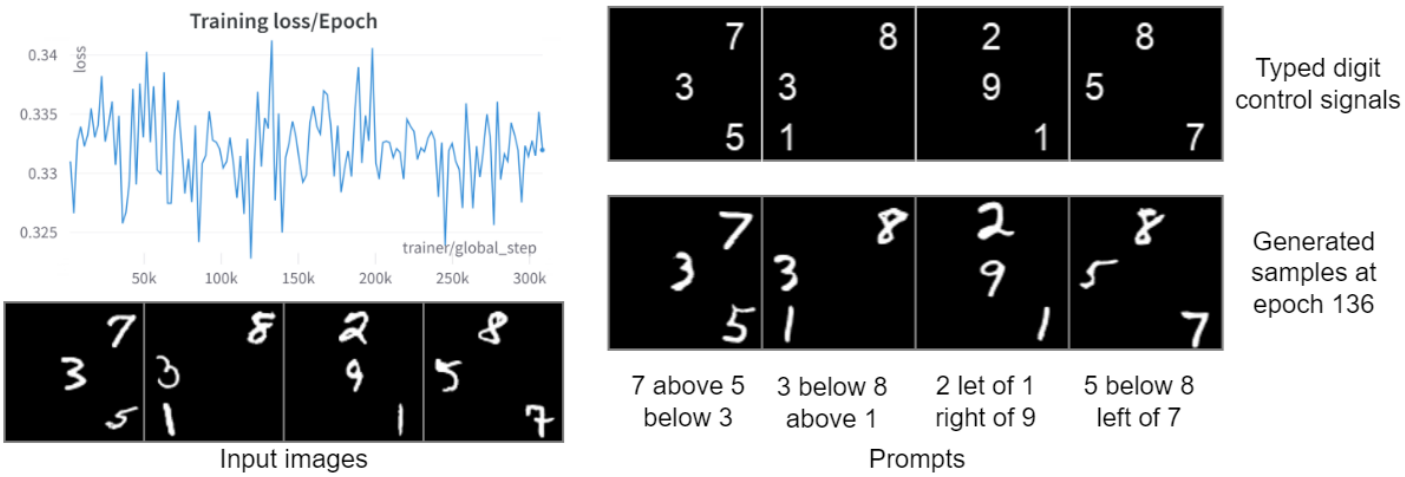


Figure 5.14: ControlNet with typed digits. Loss curve, samples and control images

Figure 5.14 shows some images sampled from a model trained using these typed digit control images, and the training loss. As expected, we observe the loss curve to start with a low loss already since we are finetuning and so are effectively working from the end of the loss curve in Figure 5.11. We can see that the generated samples adhere closely to the positions identified by the control images. There is still a level of creativity and understanding by the diffusion model since for the same typed control digit (e.g. 3), the model generates different images for it each time.

5.2.3.3 Dots as Control

Finally, we experimented with a more abstract provision of global information by replacing the digits in the typed digit control images with dots.

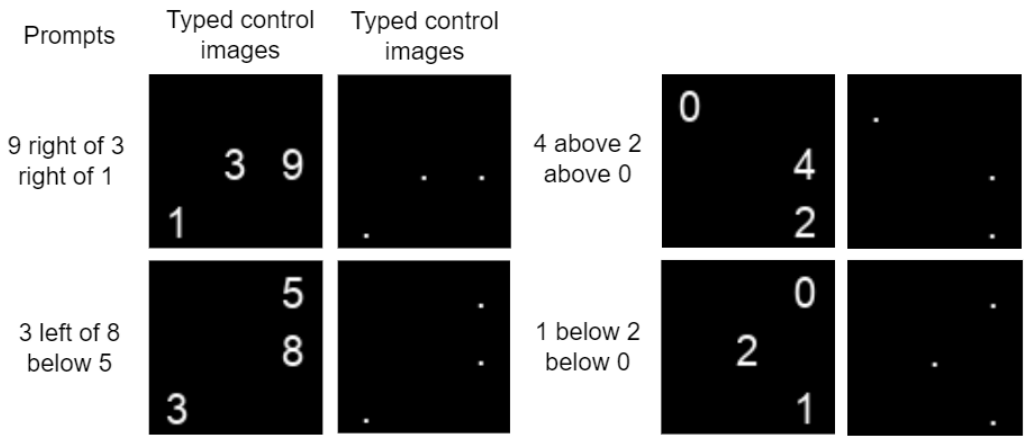


Figure 5.15: Trained ControlNet model with loss curve, samples and control images

The idea here was to test if there is a reliance on the actual digits in the control signal, or if the model has picked up and learnt from something else. It will also show how confident the model is in its understanding of the spatial relationships since the dots are providing a clue for the placement, but the diffusion model still needs to figure out which digit to place where.

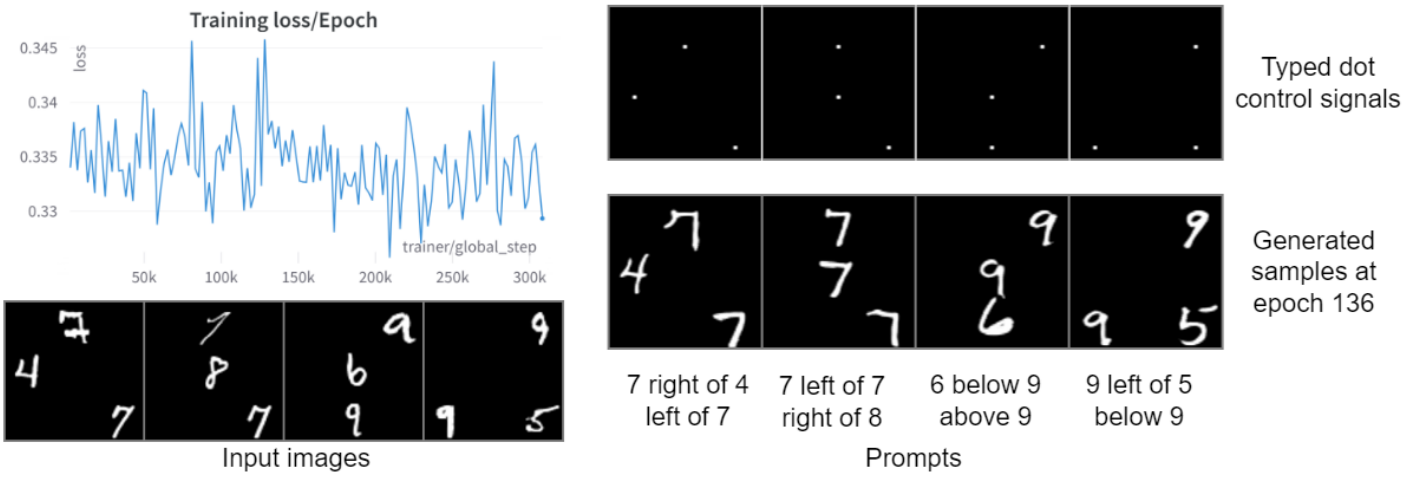


Figure 5.16: ControlNet with dotted images. Loss curve, samples and control images

We trained both models for over 26 hours until epoch 136. Figure 5.16 shows the samples we generated. It's clear the model has respected the positions of the dots in all these samples, but we do have a mistake in "7 left of 7 right of 8", as the digit 8 was never generated. Such errors, including missing and extra digits, have occurred in other models and will be discussed further in Chapter 6.

Training ControlNet models overall proved to be more efficient and straightforward compared to Stable Diffusion. Much of this efficiency stemmed from our better understanding of the codebase and lessons learned from past experiences.

Chapter 6

Evaluation

This chapter presents model evaluations and experiments. We explain our process for evaluation, starting off by describing the methodology, then outlining our evaluation metrics and discussing the results of our trained Stable Diffusion baseline and ControlNet models.

6.1 Methodology

To begin with, it's worth revisiting the evaluation process as a whole. Figure 3.2 from Chapter 3.2.1 illustrates the overall procedure well.

Our principal aim was to measure the efficacy of our models in generating realistic images based on the relationships and structure inherent in the custom dataset. To do this, given a generated image, we need a method of:

1. Calculating which digits are present
2. Computing the spatial relationships which exist between these digits

6.1.1 MNIST Digit Classifier

To achieve step 1, we can train an MNIST digit classifier. This is quite a simple and well-known task, so we quickly got a working model, with less than 5 minutes of training on the entire MNIST dataset. The initial architecture was 3 linear layers, with ReLU activation on the first two. We needed to add an additional label (label 10) for detecting blank (all-black) images. Given our generated image, we first downsample this to 84x84 pixels, then we can split this up into 9 images each 28x28 pixels, as shown in Figure 6.1. Most of the 9 images we have will represent no digit, and our classifier needs to recognise this accordingly, hence the need for label 10.

Testing this simple classifier on the MNIST test set yielded an accuracy of 96%. To further evaluate its performance, we devised a more realistic test of classifying our generated training and test datasets, for which we knew the ground truth digits from the associated prompts (filenames). Surprisingly, the classifier performed very badly in these datasets, achieving only an aerage accuracy of 82.18% (see Table 6.1).

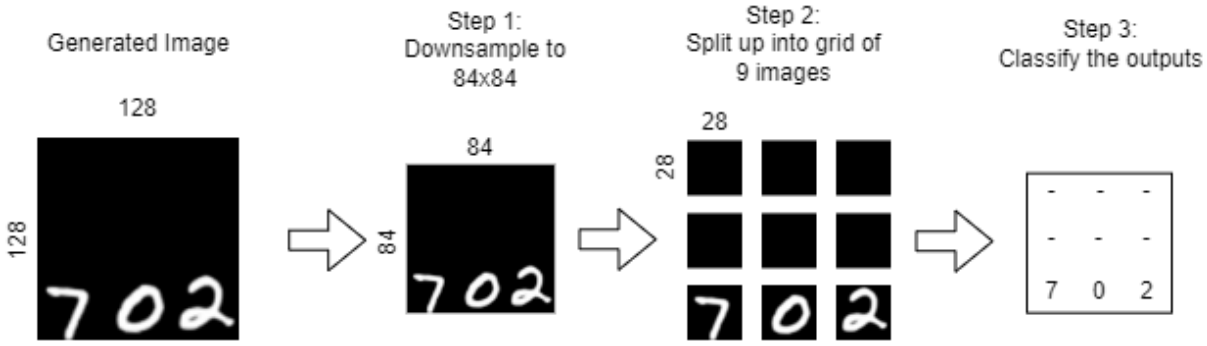


Figure 6.1: MNIST digit classification process

These underwhelming results were not acceptable since the classifier will form an integral part of the evaluation process. In an attempt to find a better solution, we tried using more advanced architectures, such as resnet18, and pre-trained models however, the problem lay with our tenth label. Any pre-trained model will only classifiy digits 0-9 meaning that given an image not containing a digit, it will still predict it to be one. Adding an initial finetuning layer on top of these models provided little benefit. Based on the idea of creating a probability distribution at the output, we took our simple network with linear layers once again, but added a softmax activation to the output layer. Remarkably, this alone provided a big increase in performance, with 98% on the MNIST testset, and 99.87% on our datasets.

Dataset	**Classifiers**	
	Linear layers	**Using softmax**
1	82.17%	99.87%
2	83.2%	99.9%
3	82.76%	99.8%
4	80.6%	99.9%
Average	82.18%	99.87%

Table 6.1: Accuracies for two MNIST digit classifiers on four separate datasets

Following from this superior performance, we will use the softmax version of the classifier in the rest of the evaluation.

6.1.2 Canonical Scene Graph Finder

Given the digits and their locations, we now need to compute all the spatial relationships between them. This process is quite methodical and involves comparing each digit with one another, and comparing their respective positions to calculate the spatial relationships between them.

We chose the canonical form since if correct, the ground truth scene graph will be a sub-graph as it only contains 2 relationships. It wouldn't be fair to just compare the digit positions between the generated image and ground truth, since we can generate digits in different positions, yet still adhere to and follow the requested spatial relationships.

6.2 Accuracy Metrics

To numerically evaluate our results, we composed 2 accuracy metrics: Strict and Inclusive accuracy.

Algorithm 1 Strict Accuracy

1: **procedure** StrictAccuracy($grnd\_truth$, gen_img)
2: $accuracy \leftarrow 0$
3: $grnd\_truth\_dgt \leftarrow digits(grnd\_truth)$
4: $grnd\_truth\_rel \leftarrow relationships(grnd\_truth)$
5: $gen\_img\_dgt \leftarrow MNIST\_digit\_classifier(gen\_img)$
6: $gen\_img\_rel \leftarrow canonical\_scene\_graph(gen\_img, gen\_img\_dgt)$
7: **if** $grnd\_truth\_dgt.length \neq gen\_img\_dgt.length$ **then**
8: **return** $accuracy$ ▷ accuracy $= 0$
9: **if** $grnd\_truth\_dgt \notin gen\_img\_dgt$ **then**
10: **return** $accuracy$ ▷ accuracy $= 0$
11: **for** r in $grnd\_truth\_rel$ **do**
12: **if** $r \in gen\_img\_rel$ **then**
13: $accuracy \leftarrow accuracy + 0.5$
return $accuracy$ ▷ accuracy is either 0.5 or 1

Strict accuracy awards points only to correct relationships, and assigns 0 if there are mistakes with the observed digits. We use the digit classifier and canonical scene graph creator from Chapter 6.1 in lines 5 and 6.

Algorithm 2 Inclusive Accuracy

1: **procedure** InclusiveAccuracy($grnd\_truth$, gen_img)
2: Same setup as lines 2 to 6 in Algorithm 1
Accuracy for digits
3: $extra\_digits \leftarrow grnd\_truth\_dgt.length - gen\_img\_dgt.length$
4: $missing\_digits \leftarrow grnd\_truth\_dgt \cap gen\_img\_dgt'$
5: $correct\_digits \leftarrow grnd\_truth\_dgt \cap gen\_img\_dgt$
6: $accuracy \leftarrow correct\_digits - extra\_digits - missing\_digits$
Accuracy for relationships
7: $matching\_rels \leftarrow grnd\_truth\_rel \cap gen\_img\_rel$
8: $accuracy \leftarrow accuracy + matching\_rels.length$
9: $accuracy \leftarrow \frac{accuracy-(-9)}{5-(-9)}$
return $accuracy$ ▷ Normalised accuracy from range [-9,5] to [0,1]

Inclusive accuracy awards points for each correct digit in the generated image, and deducts points for missing digits, and extra digits (when 4 or more digits are generated here). We score a point for each correct relationship we observed too. Finally, the accuracy is normalised from the range [-9,5] to [0,1]. This metric is more considerate of the generated image by analysing and valuing multiple parts of the finished product. Furthermore, we can isolate and individually analyse the digit and relationship accuracies with this Inclusive accuracy algorithm.

6.3 Comparison of Models

Let's discuss the results of evaluating all our trained models on held-out test sets.

For each scene graph in this 1000-image test set, we generated 3 batches of images using our models with the seeds fixed. We kept other inference configuration variables fixed too, including: `ddim_steps=50`, `n_samples=8` and `scale=7.5`, where `scale` refers to the unconditional guidance scale. For ControlNet generation, `strength=1` was used to control the influence weighting of the control image on the final result.

For batch 1 `seed=42`, batch 2 `seed=79637` and batch 3 `seed=92923` across all results.

6.3.1 Performance of Baseline Models

	Datasets	Models											
		Model 6				Model 12				Model 15			
		1	2	3	Avg	1	2	3	Avg	1	2	3	Avg
Inclusive Accuracy	Dataset 1	88.35	88.87	88.82	88.68	92.88	93.14	93.35	93.12	92.87	92.3	93.26	93.02
	Dataset 2	88.71	88.63	88.79	88.71	92.96	93.09	93	93.02	92.84	92.9	92.81	92.85
	Average	88.53	88.75	88.81	**88.70%**	92.92	93.12	93.18	**93.07%**	92.86	92.6	93.04	**92.94%**
Strict Accuracy	Dataset 1	35.47	37	37.59	36.69	54.08	55.59	57.03	55.68	61.23	62.13	63.22	62.19
	Dataset 2	36.62	36.61	36.43	36.55	55.43	56.73	55.81	55.99	61.32	61.76	60.96	61.34
	Average	36.05	36.81	37.01	**36.62%**	54.76	56.16	56.42	**55.84%**	61.28	61.95	62.09	**61.77%**

Table 6.2: Performance of different baseline models on multiple datasets. All values represent an accuracy percentage. Datasets each contain 1000 images/scene graphs.

In Table 6.2, we see inclusive and strict accuracy values for the 3 models we discussed in Chapter 5.1.3.3 and showed in Table 5.1. As a reminder, Model 6 is the largest in size of the three, but has trained for the fewest epochs despite having the most overall training time. This was due to its batch size of 1 compared to the other models' batch size of 4.

We can see that Model 6 has been outperformed by both Model 12 and 15 when looking at the overall averages (shown in bold) across both datasets and all batches. This is surprising considering it has the most complex U-Net architecture, meaning it has the capacity to theoretically sport more patterns and better understand the task. We predict the reason for this difference lies in the training setup. Model 6's batch size of 1 meant it took much longer to train per epoch compared to Models 12 and 15, at just over 1 hour per epoch. Comparing the training loss in Figures 5.10 and 5.11 shows that there is still room left for Model 6's training to converge.

Despite this, we need not train Model 6 any further since Models 12 and 15 have shown that its capable of generating better images in a smaller model with less training time. Between themselves, it isn't clear cut which the better model is, since Model 12 outperforms in the inclusive accuracy, but Model 15 has a larger margin in the strict accuracy. Going forward, we plan to fine-tune and evaluate methods on both of these models. This strategy is beneficial as it allows for a broader perspective in case of troubles that are specific to a particular model.

6.3.2 Performance of Handcrafted Tensor Approach

	Datasets	Models							
		Model 12				Model 15			
		1	2	3	Avg	1	2	3	Avg
Inclusive Accuracy	Dataset 1	90.7	91.1	90.91	90.9	91.26	91.44	91.38	91.36
	Dataset 2	90.81	90.91	90.89	90.87	91.22	91.39	91.27	91.29
	Average	90.755	91.005	90.9	**90.885%**	91.24	91.415	91.325	**91.325%**
Strict Accuracy	Dataset 1	47.34	4.01	47.66	48	58.06	58.33	58.37	58.25
	Dataset 2	47.41	48.14	48.29	47.95	57.4	58.58	57.91	58.15
	Average	47.375	26.075	47.975	**47.975%**	57.73	58.455	58.14	**58.2%**

Table 6.3: Performance of handcrafted tensor approach

Table 6.3 shows the results of finetuning baseline models 12 and 15 using ControlNet and the handcrafted tensor approach described in Chapter 5.2.2. Comparing this to the baseline averages in Table 6.2 shows that this approach has performed worse than the baseline. There are several possible reasons for this outcome. As discussed in Chapter 5.2.3.1, ControlNet has strong biases towards local spatial information and our projection of the tensor was not mimicking this structure well. This is clear from Figure 6.2 showing the visualisations of the control signals at the start and end of training. Ideally, we should see some evolving to a learnt representation but both images appear to be similarly noised data.

Figure 6.2: Comparison of control signals at the start and end of the training, showing that little to no information has been visibly learnt over this entire time.

Infact, as it's worse than the baseline, this suggests the additional control has confused the model more than it helped. A possible way of fixing this would be to avoid upscaling the tensor to match the expected input size, and instead more deeply modify the ControlNet architecture to provide the tensor in its original form. This would be more difficult, but it would reduce the modifications required to accept this data.

6.3.3 Performance of Typed Digits Approach

	Models							
	Model 12				Model 15			
	1	2	3	Avg	1	2	3	Avg
Inclusive Accuracy	89.95	59.99	90.23	**90.06%**	89.24	89.49	89.52	**89.42%**
Strict Accuracy	51.92	52.41	53.02	**52.45%**	53.75	54.77	55.49	**54.67%**

Table 6.4: Comparison of typed control approach for Models 12 and 15

Table 6.4 displays the results of evaluating the typed control signals model trained using ControlNet. Surprisingly, comparing this with the baseline averages in Table 6.2 shows this method has performed worse across both accuracies. We hypothesised this approach to outperform both the previous results however it seems the model has struggled to learn the semantic information we provided.

We designed the control signals to be images of the digits' positions, which is a high-level representation of this data. Upon further consideration, this is more to do with the image's semantics (i.e. what digits to include) as opposed to just concerning the spatial relationships. Based on the images produced by this model (examples shown in Figure 6.3), we can observe a subtle following of the control. For instance, the 6 mostly remains in the bottom-centre position as instructed in the control, same with the 3 in the top example. We are however observing more errors than before, possibly indicating the diffusion model has got worse by training on not useful data.

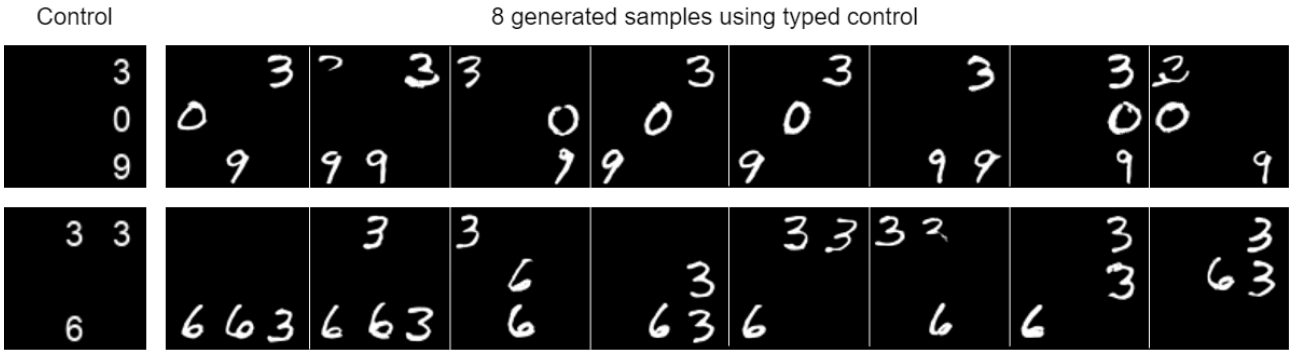


Figure 6.3: Samples generated using typed control method.
Prompts are: "3 above 0 above 9" and "3 above 6 left of 3".

6.3.4 Performance of Control Dots Approach

	Models							
	Model 12				Model 15			
	1	2	3	Avg	1	2	3	Avg
Inclusive Accuracy	94.33	94.31	94.32	**94.32%**	94.95	94.95	95.02	**94.95%**
Strict Accuracy	74.25	73.97	74.21	**74.14%**	78.04	78.04	78.14	**78.06%**

Table 6.5: Comparison of control dots approach for Models 12 and 15

Table 6.5 provides the evaluation results of the control dots methods. We are pleased to report that our results have surpassed the established baseline performance. Model 12 has scored 1.25% better in the inclusive accuracy, and 18.31% better in the strict accuracy. Its a similar story with Model 15. These results mean the model is better/more consistent at following the prompt to respect the spatial relationships. The dots are acting as low level information, which the ControlNet model is good at picking up. We can tell the model has truly understood the meanings of the relationships since the control signal merely provided a hint of the placement, but did not disclose which digit should appear where, that choice is up to the diffusion model.

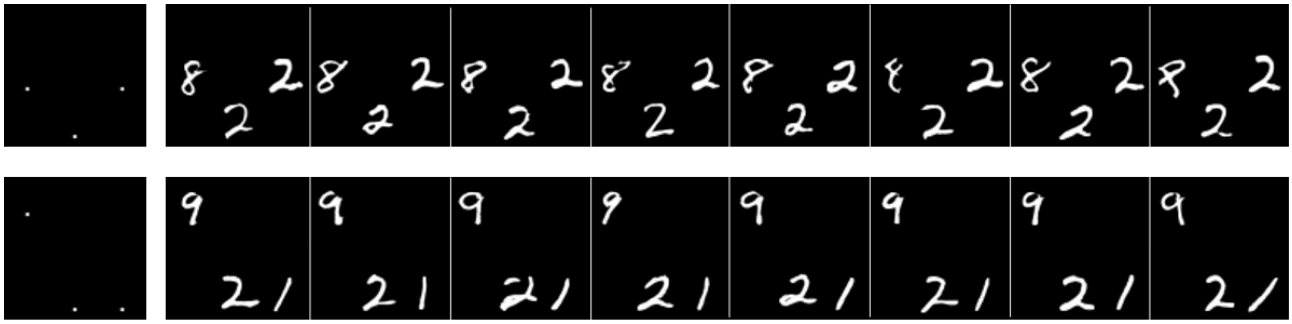


Figure 6.4: Samples generated using dots control method.
Prompts are: “8 left of 2 left of 2” and “2 below 9 left of 1”.

Figure 6.4 illustrates how much stronger the control signal is in this method. We can see that most, if not all, of the samples generated follow the instructed positions of the dots. Sometimes the digits are placed in different positions still obeying the dots and the spatial relationships. Missing digits still appear, albeit at a reduced rate than before, and extra digits appear much less

6.4 Discussion of Findings

	Models	Methods			
		Baseline	Handcrafted	Typed	Dots
Inclusive Accuracy	12	93.07%	90.885%	90.06%	94.32%
	15	92.935%	91.325%	89.42%	94.95%
Strict Accuracy	12	55.835%	47.975%	52.45%	74.14%
	15	61.765%	58.2%	54.67%	78.06%

Table 6.6: Comparison of different methods for Models 12 and 15

Table 6.6 conveniently compares the averages across all 4 of our methods. Reading across the table, we can clearly see the jump in performance for our dots method, outperforming all other methods and importantly the baseline too.

We hypothesised the typed and dots method would outperform the baseline, with the typed having superior performance. The results show this to be flipped. Upon reflection, this makes sense as ControlNet is better suited to understanding low-level information like the dots, rather than the high-level semantic information of the typed digits. We can observe this in the great performance of ControlNet Openpose models, which use lines and dots to understand the skeletal structure of a human, to create images with the same positions.

The handcrafted tensor approach has the potential to more directly influence relationships however, we can conclude that this will require a deeper modification of the architecture. Also, ControlNet may not be the best finetuning method for this approach, as we have understood it to fundamentally be provided images as control.

Overall we have found success in our 2-step process of scene graph-based annotating followed by conditioning. These insights serve as a firm foundation for future exploration and development of more efficient and powerful control methods in this field. We will provide a greater overview of this in our concluding Chapter 8.

Chapter 7

Related Work

The emergence of AI-assisted art generation has led to a new and exciting field of creativity. With machine learning algorithms, artists can now generate new forms of art that were previously impossible to create. A lot of these models have been invented and released in mid-to-late 2022, making them very new and relevant at the time of writing. In this chapter, we will explore existing methods for diffusion-based image generation, scene graph-based image generation and approaches which combine both, similar to what we have achieved in this project.

7.1 Diffusion-based Image Generation

7.1.1 DALL-E 2

DALL-E 2 [33, 34] is OpenAI's AI system that can create realistic images and art from a description in natural language. It has an encoder-decoder model where text is given, encoded into machine input, processed and then fed through a decoder to turn into a visible image. Figure 7.1 shows a high-level overview of the design.

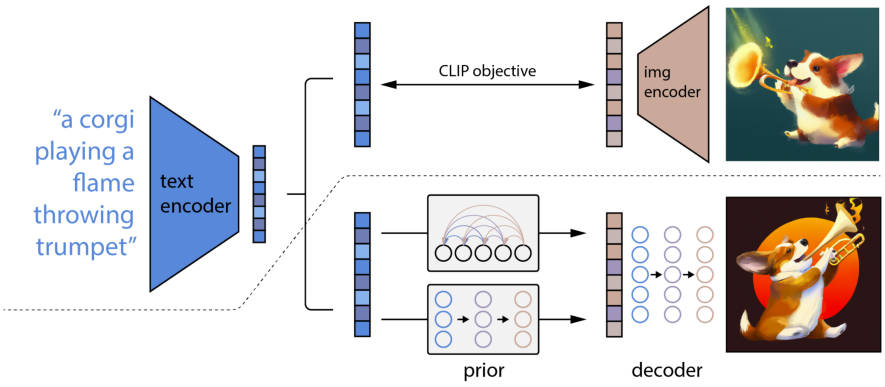


Figure 7.1: High-level overview of DALL-E 2 text-conditional image generation [34]

DALL-E 2 learns using a dataset of images/captions pairs using a two-stage model:

1. A prior which generates a CLIP image embedding given a text caption
2. A decoder which generates an image conditioned on the image embedding

Diffusion models are used for the decoder and a mix/choice of autoregressive or diffusion models can be used for the prior (see Figure 7.1), but the latter are more computationally efficient and produce higher quality samples.

7.1.1.1 CLIP

Contrastive Language-Image Pre-training (CLIP) is a representation learner for images. It uses a contrastive learning approach where the model is trained to distinguish between images and captions that match and those which do not. The model learns a general understanding between images and captions, which can be finetuned for specific purposes. In DALL-E 2, the CLIP image embedding directly influences the generated image, and is what extracts the data from the text prompt.

CLIP embeddings are desirable in many ways: they are robust to image distribution shift, have impressive zero-shot capabilities, and have been fine-tuned to achieve state-of-the-art results on a wide variety of vision and language tasks.

A key advantage of CLIP is that it embeds images and text into the same latent space, allowing us to apply language-guided image manipulation (i.e. modifying an image based on the new text). We perform the text diff by getting a text embedding of the final image, as well as the caption of the current image, then taking their difference and normalising. The CLIP latent space provides us with the ability to semantically modify images by moving in the direction of any encoded text vector [34]. One useful side effect of encoding and decoding images is that we can observe which features of an image CLIP recognises and/or discards.

7.1.1.2 DALL-E

DALL-E [35] is the official predecessor of DALLE-2. It is also a text-to-image generation model, and its approach is based on training a transformer that autoregressively models the text and image tokens as a single stream of data. This transformer was OpenAI's GPT-3 [36]. The key difference is that diffusion models are not used.

For training, a discrete VAE is taught to compress each 256x256 RGB image into a 32x32 grid of image tokens. The reduces the context size for the transformer by a factor of 192 without a large degradation in visual quality. Byte-pair encoded text tokens are then concatenated with the image tokens, before being applied to autoregressive transformer models. Overall, we maximise how closely the captions match the images and tokens.

7.1.1.3 GLIDE

GLIDE [37] is the unofficial predecessor of DALLE-2 [34], and is what initially made the major leap to diffusion models to build off the work of DALL-E [35]. GLIDE is a generative model based on text-guided diffusion models, and is what introduces CLIP guidance and classifier-free guidance [24]. DALLE-2 is comparable in quality but with greater diversity and realism. GLIDE made the jump to perform image inpainting for powerful text-driven image editing, letting us insert new objects, add shadows and reflections, change styles etc. It can also convert basic line drawings into realistic photos, similar to ControlNet's scribble modal. GLIDE's powerful zero-shot capabilities help approach unseen material and complicated circumstances.

7.1.1.4 Analysis of Performance

Despite all its successes, a critical analysis of DALL-E 2's performance [38] shows that some overlooking is performed to not quite produce the correct image for certain prompts. This analysis paper first outlined what meaning we should apply to success? If the goal is to generate candidate images that a graphic artist will choose from, then a system's success can be measured based on the quality of the best result. However, if we expect much higher standards (e.g. critical real-time applications), then we need to be satisfied with the results more frequently.

Overall, the conclusions can be summarised as:

- Visual quality of generated images was stunning, needing very little effort to create something striking.
- Impressive insights during image generation by succeeding in applying many diverse artistic styles, with extraordinary fidelity and aptness. Realistic styles are almost always physically plausible, and non-realistic conform to the particular norms of the style.
- In scenarios with a small number of objects and relations (less than 3), the language abilities seemed quite reliable. But sometimes with more, the relations don't correspond to the correct object.
- Often results were incomplete and relationships between entities are particularly challenging. DALLE-2 fails to associate specific spatial properties to objects An experiment using the Winoground benchmark [39] tested the visuolinguistic compositional reasoning of current state-of-the-art vision and language models. They found that none of these, including CLIP, does significantly better than chance.
- Anaphora (repetition of a word or phrase) and context-introducing phrases are challenging. Numbers are poorly understood, text/digits are hard to produce, and negation was problematic. Common sense reasoning also failed in their tests (e.g. an image of an elderly man's parents didn't have old-aged parents).

7.1.2 Imagen

Imagen [40, 41] is Google's version of text-to-image generation, which also generates very high-quality and high-resolution images given only a description of the scene. It works using a text encoder to convert the caption into a numerical representation, followed by a diffusion-based image generator, and finally a series of super-resolution models to upsample the output.

The text encoder used is the encoder network of T5 [42], which is a language model released by Google in 2019. It is trained for translation, sentence acceptability/similarity and summarisation, making it quite different to CLIP for DALL-E, since that is trained just on image captions. When asked for specific spatial relationships, its performance is comparable to DALL-E 2, but it has produced a state-of-the-art FID score of 7.27 on the COCO dataset without being trained on it.

7.1.3 MidJourney

Midjourney [43] is an independent research lab that produces a proprietary AI program. Its user interface is built around Discord, where users talk and interact with a bot in order to generate images. It is proficient at adapting actual art styles to create combinations of objects the user desires. A particular task it excels at is creating environments with fantasy and sci-fi scenes and dramatic lighting.

Like with DALL-E 2 and Imagen, MidJourney's algorithms and codebases are closed-source and not publically available to build on top of. This also makes it difficult to analyse how they incorporate the prompt information into the generation.

7.2 Scene Graph-based Image Generation

Approaches have been made to generate images directly from scene graphs. Let's explore the most successful of these:

7.2.1 PasteGAN

PasteGAN [44] defines a semi-parametric method of generating images from scene graphs. Its aim is to directly tackle the problem that image generators only guarantee image-level semantic consistency. Spatial arrangements of the objects and pair-wise relationships are defined by the scene graph, and object appearances are determined by the given object crops. If the crops are not provided, then an inbuilt crop selector proposes to pick the most-compatible crops from their external object tank. GANs we used for the image generator. This method has been evaluated and performed successfully on the Visual Genome [45] and COCO-Stuff [46] datasets.

7.2.2 Sg2Im

Sg2Im [47] was one of the first approaching of image generation from scene graphs. Like PasteGAN, it proposed a method of faithfully reproducing the user's prompt of complex sentences by explicitly reasoning about objects and their relationships. Graph convolution is used to process the input graphs, and use to predict a scene layout. This is an image consisting of bounding boxes and segmentation masks of where the objects should be rendered. This layout is finally converted to an image using a cascaded refinement network. This network is trained adversarially against a pair of discriminators to ensure realistic outputs. Similarly, the approach was validated using the Visual Genome [45] and COCO-Stuff [46] datasets, since these consist of images and the corresponding scene graphs they were generated from.

7.2.3 WSGC Method

The WSGC method is introduced in paper [31], which discusses the challenges of generating realistic images of complex visual scenes. The authors propose a 2 stage process of first using the canonical form of a scene graph representing the scene to predict a bounding box layout, and then conditioning a diffusion model on this layout to generate images. WSGC is the method proposed for learning equivalent relationships and constructing a canonical scene graph (shown in Figure 7.2).

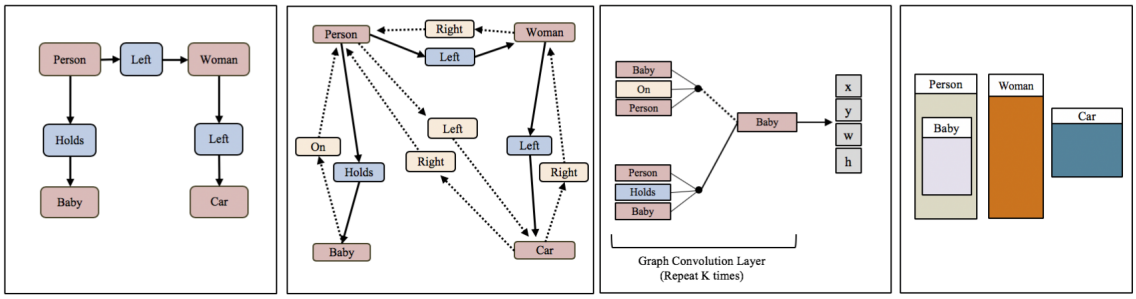


Figure 7.2: Scene graph to layout architecture [31])

A weighted scene graph to canonicalise, and then a GCN to predict bounding boxes for the nodes in the scene graph. This paper actually achieved image generation from these scene graphs by passing the generated scene layout into a pre-trained LostGAN network. However, the results had the usual downsides of GANs, and do not compare in FID scores to images from diffusion models.

7.3 Diffusion + Scene Graph Image Generation

At the start of this project, this section would have been empty, but lots of work has been happening in this field over the course of the previous 8 months. Let's discuss the work developed in parallel to this project.

7.3.1 SGDiff

SGDiff [48] is a recent piece of work which combines diffusion models with scene graphs. The authors acknowledge that image generation from scene graphs alone is challenging due to the difficulty of aligning nodes and connections in the graph with objects and relations in the images. Mapping nodes to bounding boxes is not perfect, and some relationships (e..g "eating and looking at") are hard to model spatially. Ambiguity in relationships can also hinder training with extraneous information.

SGDiff proposes to learn the scene graph embeddings by directly optimising their alignment with images. They pre-train an encoder to extract both global and local information from scene graphs that can predict the corresponding images. During training, random areas of images are masked and the model learns to reconstruct them, a form of learning global information. The final product allos for semantic modification of generated images by changing the scene graphs.

7.3.2 SceneGenie

SceneGenie [49] in a way improves upon the WSGC method [31] by conducting a scene graph based on the text prompt, then using a graph convolutional network to create 2 images: a bouding box layout and a mask image representing segmentation information. The authors train a diffusion model conditioned on both these images and the text, although it seems peculiar to still include the prompt in the loss since the 2 images have extracted all the information provided by the text.

They also use CLIP embeddings to acquire bounding box and segmentation map information. The improved image fidelity from diffusion models has helped this method surpass many previous benchmarks. For the guidance generation, they are using off-the-shelf semantic segmentation and object detection models.

7.3.3 DiffuScene

DiffuScene [50] introduces a system for indoor 3D scene synthesis based on a novel scene graph denoising diffusion probabilistic model. This model generates 3D instance properties stored in a fully-connected scene graph. For each graph node, it retrieves the most similar object geometry, which is characterised by various attributes such as location, size, orientation, semantics, and geometry features.

The authors trained a diffusion model to determine the placements and types of 3D instances given a scene graph. The proposed method can be applied to various downstream applications, such as scene completion, scene arrangement, and text-conditioned scene synthesis. The key accomplishment is the consistently well-performing spatial understanding of the models to place objects in the correct relative 3D positions.

7.3.4 LLM-grounded Diffusion

The paper [51] introduces a novel two-stage text-to-image generation method which enhances diffusion models with reasoning capabilities using off-the-shelf pre-trained large language models (LLMs). Like the previously mentioned papers, the authors aim to address the difficulties with prompts that require spatial or common sense reasoning, including negation, numeracy and attribute assignment.

In the first stage, an LLM is adapted to be a text-guided layout generator through in-context learning. Given an image prompt, the LLM outputs a scene layout with several bounding boxes, each with its own corresponding individual descriptions.

In the second stage, a novel controller is introduced to guide a diffusion model in generating images based on the layout. Both stages use frozen pre-trained models without optimising any parameters in the LLM or diffusion model.

With the recent advancements in LLMs, this paper proposes a very reasonable next step in the advancement of diffusion models. Our largest problem with these complex models is how they understand and interpret the input we give, but LLMs are great at reading and understanding human input and creating different representations of information from this. It does not seem far-fetched to hypothesise that LLM-derived text encodings or some similar knowledge representation will soon be a popular way of conditioning diffusion models.

Chapter 8

Conclusion

In this chapter, we will recap the research, talk about any limitations of decisions made, and then briefly touch upon ethical considerations. We will outline future work which can build off this project and end with some final thoughts.

8.1 Recap of Research

In this project, we set out to investigate methodologies for making diffusion models better at understanding spatial relationships. At the end of this project, we have a much better grasp of how information is used throughout the diffusion process, and what possibilities exist for improving the ways that diffusion models learn and get conditioned. We have finished with several models, which we have open-sourced and made publically available, which improve upon the baseline performance for our task of better spatial relationships following.

With this consideration, the project has been a success. Reflecting back on our problem statement in Chapter 3.1, we see that our solution has met all the required needs. We have measured a combination of the object and relationship count accuracy, and shown the performance to exceed the baseline with our dots method, object identity preservation and relationship integrity has been analysed by our accuracy metrics, and we have created a method with great potential to scale and generalise by building on top of ControlNet.

The two-stage process we devised in our typed and dots control methods have proven to be successful by significantly outperforming our Stable Diffusion baselines as discussed in Chapter 6.3. Some limitations of this method do exist, with the main being the annotating process required to generate these control signals. With our simple 3x3 grid MNIST dataset, we can compute the control signals quite quickly on even a low-performance machine due to the finite combinations. The complexity of this quickly grows with real-world images containing a much larger range of objects and relationships. To scale and generalise, modifications should be made to how we create and represent the control information.

Despite that, our overall aim of creating a proof-of-concept in improving the spatial understanding of diffusion models has been successful and we have contributed to this cutting-edge field with our novel approach using ControlNet.

8.2 Ethical Considerations

There are several ethical, legal, professional, and societal issues briefly faced by diffusion models, but the majority are generic and do not apply to our datasets.

The misuse of generated images alongside the use of copyrighted training images is a large problem in image generative models. Since we have used our custom dataset of images generated with the freely available MNIST dataset, and since our models can't produce recognisable images other than these digits, this does not concern us.

Bias could be introduced by the scene graphs or text prompts, and potentially passed through to be learnt by the diffusion models. Again, our inputs are very structured and finite, making it difficult to speculate beyond the specific task. Any malicious user will need to make significant changes to convert this research codebase into a tool for generating fooling images with a misrepresentation of reality.

8.3 Future Work

This project has explored and built only the foundations of this research in the field, there are many more exciting areas which can be further explored and investigated.

Firstly, we can begin training on a large, more complex dataset now that we have shown a successful proof-of-concept. This can start with the COCO-Stuff and Visual Genome datasets, which already come with attached scene graphs. For the control signals, bounding boxes and segmentation masks similar to SceneGenie (Chapter 7.3.2) can be considered, only this time using the more robust ControlNet finetuning platform. These conditioning styles are exactly what ControlNet excels at, meaning a huge potential for state-of-the-art performance in spatial consistency awaits.

Additionally, we can try to generate the scene graph from the text prompt itself, rather than relying on its separate existence. LLMs can possibly be used to perform this initial stage, before processing the graph further with graph neural networks. In fact, modern powerful LLMs can potentially take control of a larger share of the conditioning of diffusion models. Exploring how LLMs can manipulate and transfer human input to a diffusion model may be the next ground-breaking step in this field.

Finally, there is the interesting question of semantic relationships and how they play a role. We observed with our typed control signals that semantic information was difficult to learn from however, success would mean a complete representation of all data in scene graph form, making the text prompt unnecessary. Furthermore, this could result in more realistic and contextually appropriate scenes, at which point it would be useful to evaluate visual quality using Inception Score or the FID metric.

In conclusion, this project not only establishes a stepping stone for future explorations in the field of text-to-image generation using deep learning, but also serves as a testament to the transformative potential of artificial intelligence. As we continue to innovate and refine these techniques, we move closer to a future where the border between natural language and visual understanding is blurred, opening an array of possibilities that are as limitless as the human imagination.

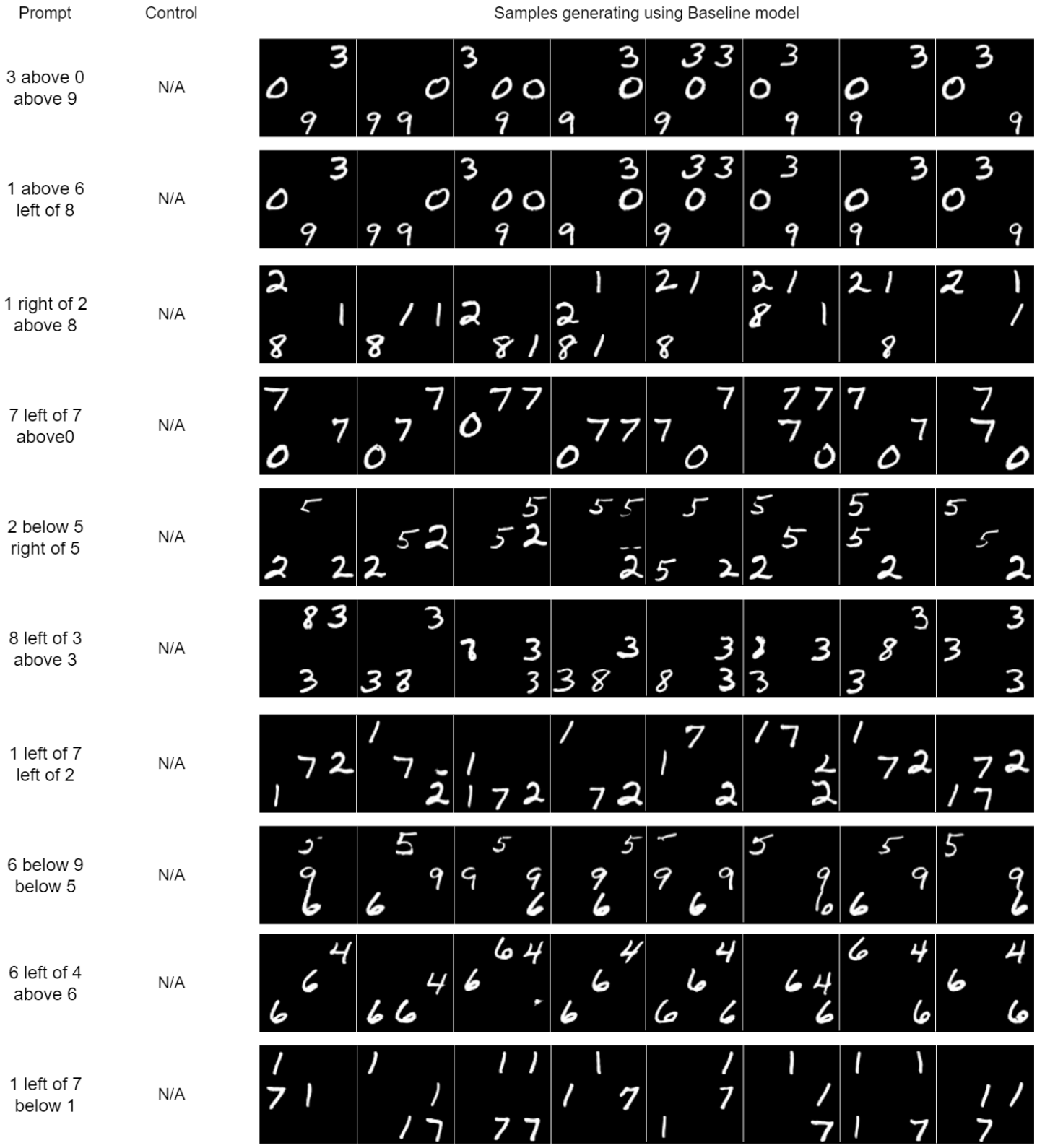
Appendix A

Images Produced by Models

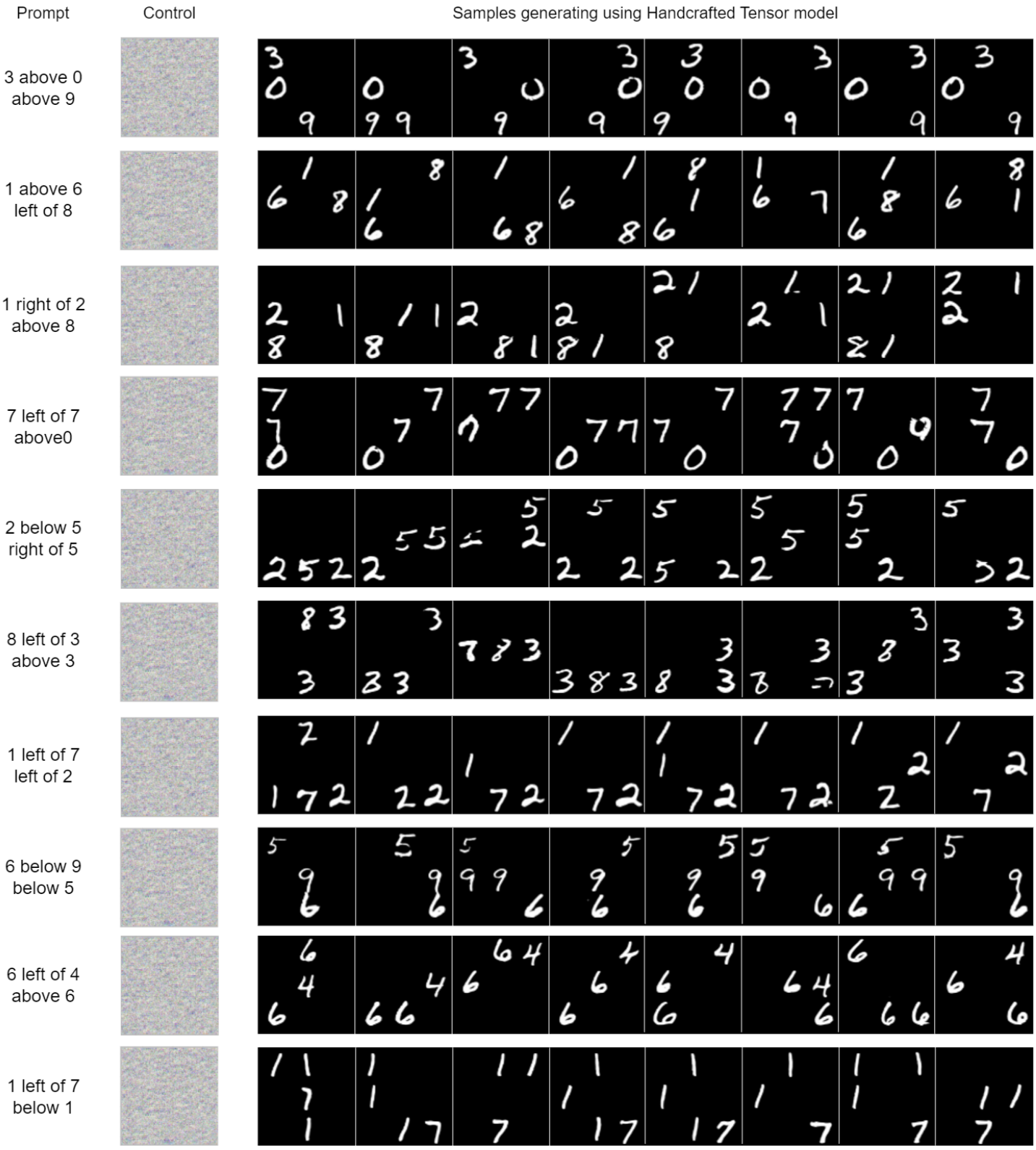
The appendix that follows provides a visual representation of the performance of the four different methods explored in this project: the Baseline Approach, the Handcrafted Tensor Approach, the Typed Control Method, and the Control Dots Method. Ten distinct samples, each accompanied by its respective prompt, are showcased for each method, all generated by Model 15 and using the same `seed=42`.

These illustrations serve to illuminate the strengths, weaknesses, and unique characteristics of each method. By examining these samples, readers will gain an understanding of how these methods operate in practice and will be able to discern the qualitative differences among them. These images also support the evaluation discussed in Chapter 6.

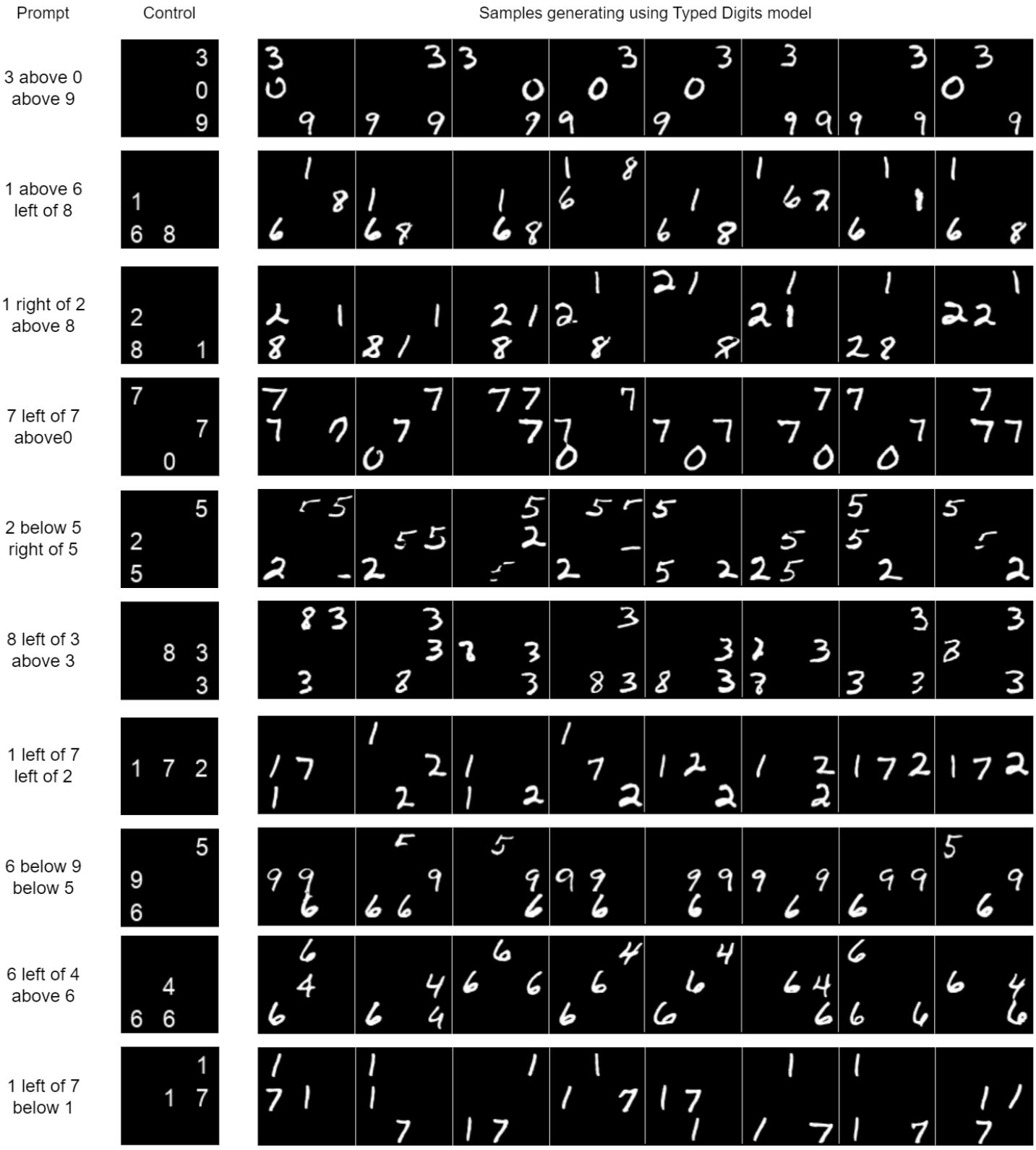
A.1 Baseline Method



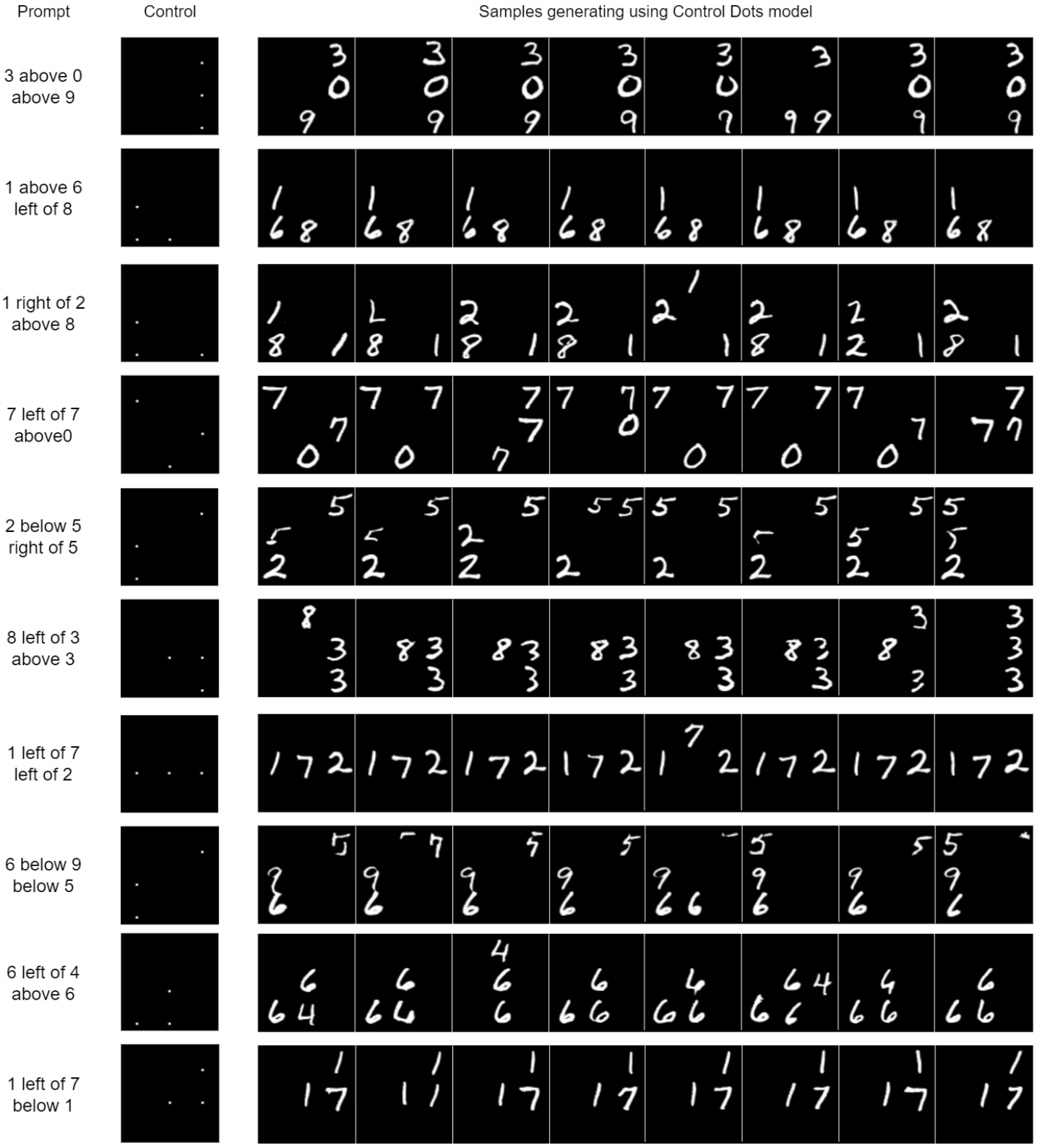
A.2 Handcrafted Tensor Approach



A.3 Typed Digits Approach



A.4 Control Dots Approach



Bibliography

[1] Harold Cohen. The further exploits of aaron, painter. *Stanford Humanities Review*, 4(2):141–158, 1995.

[2] Chris Garcia. Harold cohen and aaron—a 40-year collaboration - chm, 2016. URL https://computerhistory.org/blog/harold-cohen-and-aaron-a-40-year-collaboration/.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Science Robotics*, 3:2672–2680, 6 2014. ISSN 10495258. doi: 10.48550/arxiv.1406.2661. URL https://arxiv.org/abs/1406.2661v1.

[4] Lilian Weng. From gan to wgan. *lilianweng.github.io*, 2017. URL https://lilianweng.github.io/posts/2017-08-20-gan/.

[5] Daniel Herr, Benjamin Obert, Matthias Rosenkranz, and Haiyang Chen. Challenges and corresponding solutions of generative adversarial networks (gans): A survey study. *Journal of Physics: Conference Series*, 1827: 012066, 3 2021. ISSN 1742-6596. doi: 10.1088/1742-6596/1827/1/012066. URL https://iopscience.iop.org/article/10.1088/1742-6596/1827/1/012066https://iopscience.iop.org/article/10.1088/1742-6596/1827/1/012066/meta.

[6] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 10 2017. doi: 10.48550/arxiv.1710.10196. URL https://arxiv.org/abs/1710.10196v3.

[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *7th International Conference on Learning Representations, ICLR 2019*, 9 2018. doi: 10.48550/arxiv.1809.11096. URL https://arxiv.org/abs/1809.11096v2.

[8] David Alvarez-Melis, Vikas Garg, and Adam Tauman Kalai. Why gans are overkill for nlp. 5 2022. doi: 10.48550/arxiv.2205.09838. URL https://arxiv.org/abs/2205.09838v1.

[9] Lilian Weng. From autoencoder to beta-vae. *lilianweng.github.io*, 2018. URL https://lilianweng.github.io/posts/2018-08-12-vae/.

[10] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12:307–392, 6 2019. ISSN 19358245. doi: 10.1561/2200000056. URL https://arxiv.org/abs/1906.02691v3.

[11] Imant Daunhawer, Thomas M. Sutter, Kieran Chin-Cheong, Emanuele Palumbo, and Julia E. Vogt. On the limitations of multimodal vaes. 10 2021. doi: 10.48550/arxiv.2110.04121. URL https://arxiv.org/abs/2110.04121v2.

[12] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Towards deeper understanding of variational autoencoding models. 2 2017. doi: 10.48550/arxiv.1702.08658. URL https://arxiv.org/abs/1702.08658v1.

[13] Lilian Weng. Flow-based deep generative models. *lilianweng.github.io*, 2018. URL https://lilianweng.github.io/posts/2018-10-13-flow-models/.

[14] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 5 2016. doi: 10.48550/arxiv.1605.08803. URL https://arxiv.org/abs/1605.08803v3.

[15] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Yingxia Shao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *Comprehensive Survey of Methods and Applications*, 1:39, 9 2022. doi: 10.48550/arxiv.2209.00796. URL https://arxiv.org/abs/2209.00796v9.

[16] Ryan O'Connor. Introduction to diffusion models for machine learning, 2022. URL https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/.

[17] Lilian Weng. What are diffusion models? *lilianweng.github.io*, Jul 2021. URL https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[18] J. Rafid Siddiqui. Diffusion models made easy. understanding the basics of denoising... | by j. rafid siddiqui, phd | towards data science. 5 2022. URL https://towardsdatascience.com/diffusion-models-made-easy-8414298ce4da.

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 2020-December, 6 2020. ISSN 10495258. doi: 10.48550/arxiv.2006.11239. URL https://arxiv.org/abs/2006.11239v2.

[20] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. pages 10674–10685, 12 2021. ISSN 10636919. doi: 10.48550/arxiv.2112.10752. URL https://arxiv.org/abs/2112.10752v2.

[21] Aman Chadha and Vinija Jain. Diffusion models. *Distilled AI*, 2020. https://aman.ai.

[22] Weihao Weng and Xin Zhu. U-net: Convolutional networks for biomedical image segmentation. *IEEE Access*, 9:16591–16603, 5 2015. ISSN 21693536. doi: 10.48550/arxiv.1505.04597. URL https://arxiv.org/abs/1505.04597v1.

[23] Thushan Ganegedara. Intuitive guide to understanding kl divergence | by thushan ganegedara | towards data science, 5 2018. URL https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-understanding-kl-divergence.

[24] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. 7 2022. doi: 10.48550/arxiv.2207.12598. URL https://arxiv.org/abs/2207.12598v1.

[25] Jooyoung Choi, Sungwon Kim, Yonghyun Jeong, Youngjune Gwon, and Sungroh Yoon. Ilvr: Conditioning method for denoising diffusion probabilistic models. *Proceedings of the IEEE International Conference on Computer Vision*, pages 14347–14356, 8 2021. ISSN 15505499. doi: 10.48550/arxiv.2108.02938. URL https://arxiv.org/abs/2108.02938v2.

[26] Georgios Batzolis, Jan Stanczuk, Carola-Bibiane Schönlieb, and Christian Etmann. Conditional image generation with score-based diffusion models. 11 2021. doi: 10.48550/arxiv.2111.13606. URL https://arxiv.org/abs/2111.13606v1.

[27] Andrew. Stable diffusion webui automatic1111: A beginner's guide - stable diffusion art, 3 2023. URL https://stable-diffusion-art.com/automatic1111/#comments.

[28] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. 8 2022. URL https://arxiv.org/abs/2208.12242v2.

[29] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. 6 2021. URL https://arxiv.org/abs/2106.09685v2.

[30] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. 2 2023. URL https://arxiv.org/abs/2302.05543v1.

[31] Roei Herzig, Amir Bar, Huijuan Xu, Gal Chechik, Trevor Darrell, and Amir Globerson. Learning canonical representations for scene graph to image generation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12371 LNCS: 210–227, 12 2019. ISSN 16113349. doi: 10.48550/arxiv.1912.07414. URL https://arxiv.org/abs/1912.07414v5.

[32] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 9 2016. doi: 10.48550/arxiv.1609.02907. URL https://arxiv.org/abs/1609.02907v4.

[33] OpenAI. Openai dall-e 2 webpage, 2022. URL https://openai.com/dall-e-2/.

[34] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. 4 2022. doi: 10.48550/arxiv.2204.06125. URL https://arxiv.org/abs/2204.06125v1.

[35] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. 2 2021. doi: 10.48550/arxiv.2102.12092. URL https://arxiv.org/abs/2102.12092v2.

[36] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020-December, 5 2020. ISSN 10495258. doi: 10.48550/arxiv.2005.14165. URL https://arxiv.org/abs/2005.14165v4.

[37] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. 12 2021. doi: 10.48550/arxiv.2112.10741. URL https://arxiv.org/abs/2112.10741v3.

[38] Gary Marcus, Ernest Davis, and Scott Aaronson. A very preliminary analysis of dall-e 2. 4 2022. doi: 10.48550/arxiv.2204.13807. URL https://arxiv.org/abs/2204.13807v2.

[39] Tristan Thrush, Ryan Jiang, Max Bartolo, Amanpreet Singh, Adina Williams, Douwe Kiela, and Candace Ross. Winoground: Probing vision and language models for visio-linguistic compositionality. pages 5228–5238, 4 2022. ISSN 10636919. doi: 10.48550/arxiv.2204.03162. URL https://arxiv.org/abs/2204.03162v2.

[40] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. 5 2022. doi: 10.48550/arxiv.2205.11487. URL https://arxiv.org/abs/2205.11487v1.

[41] Ryan O'Connor. How imagen actually works, 6 2022. URL https://www.assemblyai.com/blog/how-imagen-actually-works/.

[42] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of

transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 10 2019. ISSN 15337928. doi: 10.48550/arxiv.1910.10683. URL https://arxiv.org/abs/1910.10683v3.

[43] Jonas Oppenlaender. The creativity of text-to-image generation. 11:192–202, 5 2022. doi: 10.1145/3569219.3569352. URL http://arxiv.org/abs/2206.02904http://dx.doi.org/10.1145/3569219.3569352.

[44] Yikang Li, Tao Ma, Yeqi Bai, Nan Duan, Sining Wei, and Xiaogang Wang. Pastegan: A semi-parametric method to generate image from scene graph. *Advances in Neural Information Processing Systems*, 32, 5 2019. ISSN 10495258. doi: 10.48550/arxiv.1905.01608. URL https://arxiv.org/abs/1905.01608v2.

[45] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123:32–73, 2 2016. ISSN 15731405. doi: 10.48550/arxiv.1602.07332. URL https://arxiv.org/abs/1602.07332v1.

[46] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1209–1218, 12 2016. ISSN 10636919. doi: 10.48550/arxiv.1612.03716. URL https://arxiv.org/abs/1612.03716v4.

[47] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1219–1228, 4 2018. ISSN 10636919. doi: 10.48550/arxiv.1804.01622. URL https://arxiv.org/abs/1804.01622v1.

[48] Ling Yang, Zhilin Huang, Yang Song, Shenda Hong, Guohao Li, Wentao Zhang, Bin Cui, Bernard Ghanem, and Ming-Hsuan Yang. Diffusion-based scene graph to image generation with masked contrastive pre-training. 11 2022. doi: 10.48550/arxiv.2211.11138. URL https://arxiv.org/abs/2211.11138v1.

[49] Azade Farshad, Yousef Yeganeh, Yu Chi, Chengzhi Shen, Björn Ommer, and Nassir Navab. Scenegenie: Scene graph guided diffusion models for image synthesis. 4 2023. URL https://arxiv.org/abs/2304.14573v1.

[50] Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Scene graph denoising diffusion probabilistic model for generative indoor scene synthesis. 3 2023. URL https://arxiv.org/abs/2303.14207v1.

[51] Long Lian, Boyi Li, Adam Yala, Trevor Darrell, and Uc Berkeley. Llm-grounded diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models. 5 2023. URL https://arxiv.org/abs/2305.13655v1.