Modelling Additive Manufacturing Processes via Graph-Conditioned Diffusion Models

Rupert Menneer

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Clare College August 2024

For Molly

Declaration

I, Rupert Menneer of Clare College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Software Declaration: Standard Python libraries were used e.g. *Pytorch, NumPy, Matplotlib.* In particular, *Pytorch Geometric* was heavily utilised. GitHub Repositories: `github.com/lucidrains/imagen-pytorch` and `https://github.com/NVlabs/edm` were used for baseline diffusion model architectures.

Data Declaration: As part of the project a 3D printing dataset was collected at the Institute of Manufacturing (IfM). The majority of this work is not my own. Most of this work can be attributed to Christos Margadji and the prior research of the Computer Assisted Manufacturing (CAM) group.

Word Count: 14,717

Rupert Menneer
August 2024

Signed:

Rupert . M

Acknowledgements

I would like to extend my thanks to my supervisors, Dr. Sebastian Pattinson and Christos Margadji, without whom this project would not have been possible. I was fortunate to benefit from our weekly meetings, which provided a space for reflective and thoughtful discussions. I am especially grateful to Christos Margadji for his hands-on approach to data collection.

Additionally, I would like to express my gratitude to everyone involved in organising the MLMI course, as well as to my cohort for their collaboration and support throughout this challenging but rewarding year.

Abstract

Additive Manufacturing (AM) has the potential to revolutionise manufacturing across various sectors. By building parts layer-by-layer, this method allows for complex and custom geometries at a substantially lower cost. However, AM processes are complex and can be unreliable, which leads to defects in the final product. Traditionally, human operators oversee the process by applying corrections through trial and error, yet this approach is costly and prone to errors. Deep learning has recently been applied in the detection, forecasting and compensation of these errors. In particular, monitoring the process with vision sensors provides insightful data that can be utilised by machine learning models. One such framework, *diffusion modelling*, has proven to be a powerful conditional image generator.

This project proposes a method to forecast layers before they are printed via generative diffusion modelling, which can then be used for error prediction and avoidance. By creating graphs that represent the printing process parameters, the aim is to predict the final layer image. In essence, this method proposes a graph-to-image generative pipeline that combines recent advances in Graph Neural Networks (GNNs) with generative diffusion models. A new architecture is introduced by fusing a convolutional neural network (CNN) with GNNs via a novel representation. This representation is coined the *dual-graph* as it considers two graphs: one which represents the target image, the other which represents conditioning variables. We demonstrate this to be a highly effective approach, and show its application in an AM setting.

To facilitate the project, a first-of-its-kind dataset was collected that contains over 2500 process parameter graphs and print layer image pairs. To achieve this, we introduce a new strategy that aids in the semi-automatic collection of such data. This work could enhance the AM process by accurately reconstructing layer images, in turn preventing future defects or minimising the effect of existing ones. This could improve the viability of the method for industrial or recreational use.

Table of contents

1 Introduction **1**

1.1 Motivation 1

1.2 Approach 3

1.3 Contributions 5

1.4 Thesis Outline 6

2 Related Work **7**

2.1 Deep Learning for Additive Manufacturing 7

2.2 Methodological Background 10

2.2.1 Diffusion Models 10

2.2.2 Graph Neural Networks 17

3 Method **20**

3.1 Data 20

3.1.1 Space-Time Graphs 20

3.1.2 Data Collection & Pre-Processing 22

3.1.3 Synthetic Data Generation 30

3.2 Conditioning Images with Graphs 31

3.2.1 Learnable Render using Line Lookup 32

3.2.2 Dual-Graph GNN 34

3.3 Graph-to-Image Diffusion Model 39

3.4 Validation Procedures 41

4 Results **44**

4.1 Experimental Setup 44

4.2 Synthetic Data 45

4.3 Evaluating Real-World Data: Initial Findings 48

4.4 Further Improvements & Ablation Studies 50

4.5 Final Model 54

4.5.1 Generated Samples 55

4.5.2 Case Study: Process Parameter Optimisation 58

5 Conclusions **62**

5.1 Discussion 62

5.2 Future Work 63

5.2.1 Future Work in FDM 63

5.2.2 Applications in Other Domains 64

References **66**

Appendix **74**

A.1 Elucidated Diffusion 74

A.2 GNN Architecture Details 76

A.3 Further Implementation details 77

A.4 Model Probing 78

Chapter 1

Introduction

1.1 Motivation

Additive Manufacturing (AM) is the method of creating objects by iteratively adding material, typically layer-by-layer such as in 3D printing. This method allows for complex and bespoke parts to be made at a low cost, which could lead to advances in many industries such as aerospace or medical device design (Haghiashtiani et al. [2020], Najmon et al. [2019]). The most common approach: *Fused Deposition Modelling* (FDM) heats a material filament before extruding it through a nozzle head at precise locations on a surface (Ngo et al. [2018]). However, this approach commonly suffers from defects which can significantly degrade the mechanical, dimensional and functional properties of the final part. Traditionally, human operators must estimate the cause of the issue and restart the process with adjusted parameters (Mohamed et al. [2015], Brion et al. [2022]). Yet this not only expensive, but they often fail to navigate the process' complexities, meaning that errors are still common. This has hindered its adoption in industrial-scale production (Baechle-Clayton et al. [2022]).

Manufacturing defects commonly observed in FDM can include: adhesion defects, extrusion defects, blobbing or stringing (see Figure 1.1). The causes of these defects include: complex or unstable geometries (e.g. a single dot is less likely to adhere to the print bed and intricate details can be compromised), inappropriate parameters (e.g. low or high flow rate) or mechanical failures (e.g. belt slippage). The root causes complexly interplay with one another adding to the complexity of minimising errors. For example, Figure 1.2 shows the significant difference in output when printing the same geometry with different the flow-rate.

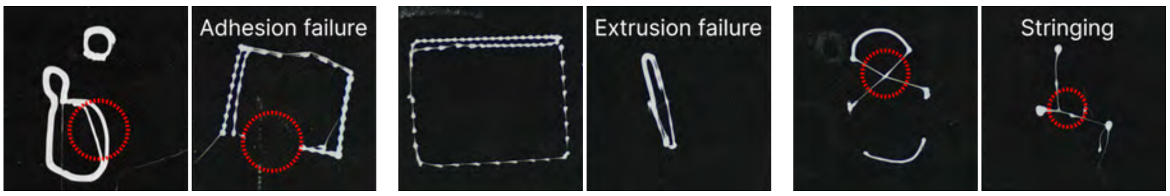

Figure 1.1: **Example of Common FDM Errors.** Real images exhibiting common defects. Local errors are highlighted in red circles. Adhesion failure is the inability of the printed layers to properly bond to each other or the print bed. Extrusion defects occur when material is not properly deposited e.g. the flow rate is too slow. Stringing results in undesired strands as the nozzle head moves to different parts of the print.

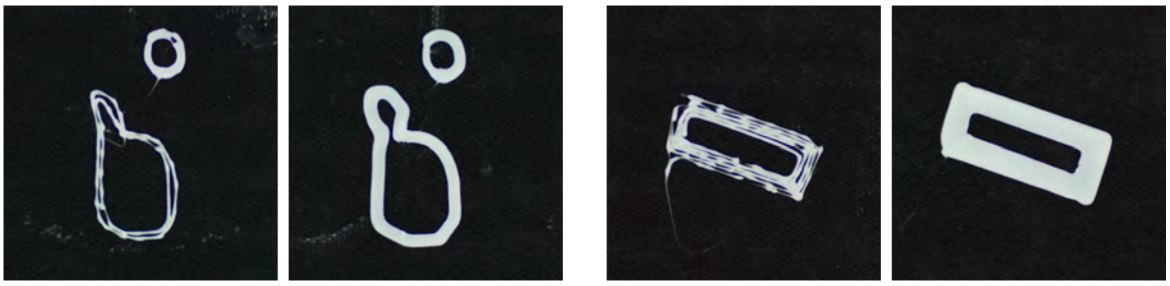

Figure 1.2: **The Effect of Different Process Parameters.** The same input geometries printed with different flow rates.

To this end, if it were possible to forecast and avoid errors, this would have significant benefits for its applicability. Ideally, such an approach would be able to forecast potential errors before they occur, be applicable in-situ, and understand how different printing parameters influence the physical end product. One area of active research focuses on using on-board sensors to monitor the process. Particularly, vision sensors provide rich data and have allowed recent advances in deep learning to be applied to error detection and prevention (Straub [2015], Cunha et al. [2021], Brion and Pattinson [2022b,a]).

The Institute for Manufacturing (IFM) has developed a framework that relates process monitoring data with spatial coordinates and control commands being executed at the instance of collection. They refer to this data as *space-time graphs*. This framework forms the foundation for this project's aim: to forecast the occurrence and location of defects in FDM processes, and to provide interpretable feedback to operators based upon input control commands.

1.2 Approach

In this work, a new method is introduced for predicting errors in FDM. This is achieved by learning the mapping between space-time graphs (e.g. input parameters) and the resulting layer produced. The graph indicates when, where and how the printer will deposit material. The resulting layer is captured via a camera mounted vertically above the print bed (see Figure 1.3). In order to achieve this, a commercially available 3D printer was retro-fitted. The chosen printer came equipped with a movable print-mill (i.e. a conveyor belt) which allowed for semi-automatic generation of a dataset consisting of graph and layer-image pairs. Since this is an unexplored research area, data collection focused on single-layer prints for both simplicity and collection speed. Moreover, if the first layer is printed successfully this decreases the chances of later failures (i.e a strong foundation minimises errors propagating to higher layers).

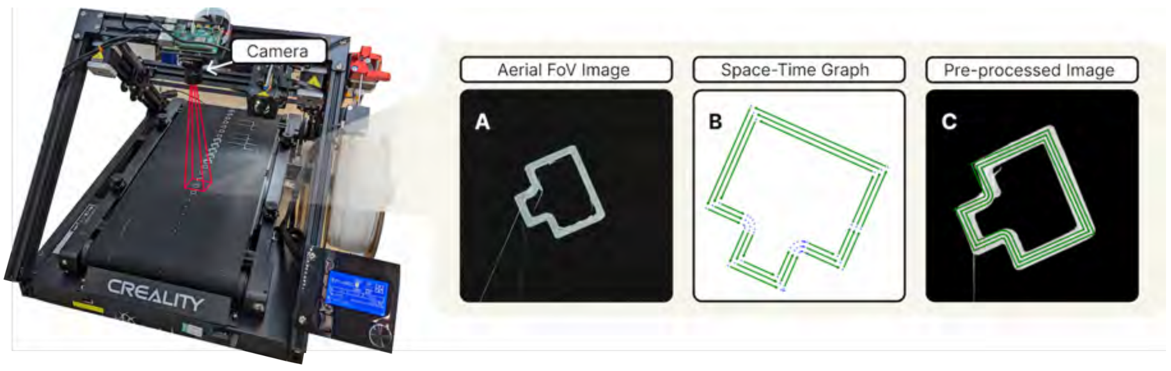

Figure 1.3: **Data Collection.** **A)** Layer images are captured with an aerial camera. **B)** The input space-time graph was used to print the layer image, this contains process parameters and spatial information. **C)** A pre-processed layer image.

The diffusion framework was selected to learn the mapping from inputs to resulting layer image. In recent years, diffusion modelling (Ho et al. [2020], Song and Ermon [2020]) has proven to be an extremely effective model for image generation. What makes these models particularly effective is how they break down the difficult problem of generation into many sub-problems. They achieve this by optimising a model to predict noise that has been used to corrupt a target image at varying noise levels. Once trained, new samples are generated by initialising the model with pure noise and conditioning variables; these are then iteratively denoised. This project proposes a graph-to-image pipeline by aiming to produce layer-image samples conditioned on space-time graphs.

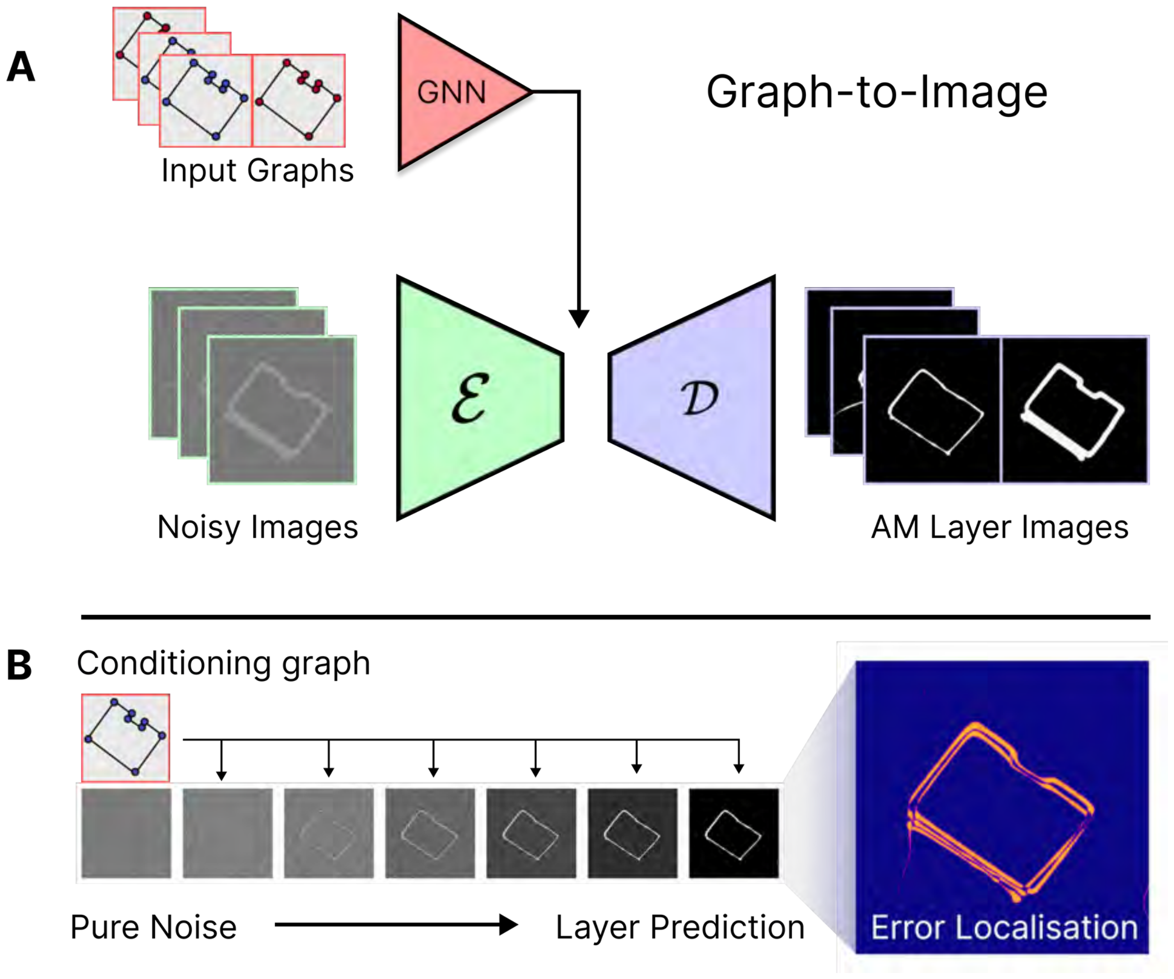

Figure 1.4: **A) Training Overview.** The input data consists of space-time graphs and layer image pairs. Noisy data is passed to a customised U-Net that allows conditioning images with graphs, principally through a GNN. **B) Inference Overview** New samples can be generated by starting from pure noise and the conditioning input graph. Error localisation is achieved by comparing expected output with the generated samples.

To train the generative pipeline, a U-Net model (Ronneberger et al. [2015]) is augmented with two novel computational units. These facilitate the task of transforming a space-time graph into latent image-space, and provide strong spatial conditioning signals to the model during denoising. The first of these units is referred to as a dual-graph GNN. This utilises the recent advances in geometric deep learning to process the space-time graph and its connection to the image. This takes into account that the underlying geometry contains important information required for error prediction. The second unit borrows ideas from classical rendering to produce a learnable 2D latent image representation of the graph.

The model was trained with real data captured by the printer's onboard vision sensors, as well as a larger synthetic dataset. Real data is time-consuming to collect, thus synthetic data allowed a head-start in model development, and the ability to pre-train on a related task. The real data was extensively pre-processed before being passed to the model. Most notably, images were registered i.e. transforming the camera coordinate system into the graph coordinate system via a transformation matrix. Due to belt slippage, mechanical inaccuracies and print failures, a robust affine matching system was developed in order to minimise coordinate discrepancies. See Figure 1.3 for the overview of data collection, and Figure 1.4 for the general approach.

Although existing machine learning techniques have aimed to tackle the problem of error prediction in FDM, none of them explicitly learn the mapping between process parameters and the resulting layer images. For example, previous approaches have attempted to classify whether a specific print will suffer from warping defects (Brion et al. [2022]), yet this is less interpretable than image outputs, and difficult to analyse for human operators. Some approaches have adopted a generative approach (using GANs) to predict complex microstructures in different types of AM (e.g. selective last melting Song et al. [2023]). Yet, no work currently utilises the modern advances in both diffusion modelling and graph neural networks to tackle the challenge in FDM. This approach has the potential to accurately reconstruct the resulting layer images from any given graph. This could allow operators to forecast likely errors before they print, ultimately making the process more reliable and minimising wastage. Additionally, such a model could be used to directly find the most stable process parameters for a desired geometry, which opens the door to automatic error compensation. Validation routes for the proposed framework rely on layer image reconstruction accuracy for unseen layer graphs.

1.3 Contributions

The main contributions of this thesis are as follows:

- **Novel Dataset & Collection Strategy**: a new AM dataset comprising of over 2500 graph and image pairs. This data is automatically collected by augmenting a commercially available 3D printer with vision sensors. We make this data freely available here.

- **First Graph-to-Image Diffusion Model and Novel Application for AM**: this work proposes the first application of diffusion models to predict images from spatially aligned graphs e.g. a graph-to-image generative pipeline. This approach is applied to predict layer images given AM process parameters represented as a space-time graph.

- **Novel Computational Units for Graph Conditioning**: several novel computational units are introduced in order to successfully train the graph-to-image generative pipeline. These units have application beyond AM and may be used with little to no modification for other graph-to-image tasks. Furthermore, it provides a new perspective on how spatial graphs can be used to condition images via a novel dual-graph representation. Although these units are integrated into a U-Net, the ideas introduced are directly transferable to other vision architectures. All project code is included here.

This project plans to submit these contributions as a conference paper to CVPR. Additionally, a follow-up work which extends the framework with nozzle-head video is being developed.

1.4 Thesis Outline

In **Chapter 2** existing literature is reviewed and presented. This section explores the existing work in deep learning for AM, and presents the fundamental theory required to understand the contributions proposed by this project i.e. diffusion, architecture choices and GNNs. **Chapter 3** explains the overall methodology, including data collection, pre-processing, synthetic data creation, the proposed novel computational units and the overall graph-to-image pipeline. Subsequently, **Chapter 4** presents the results of the pipeline, both qualitatively and quantitatively. Finally, **Chapter 5** summarises the project's findings, examines its impacts and explores potential avenues for future research. This section includes reflections on the project, both its merits and shortcomings.

Chapter 2

Related Work

This section outlines prior work, which is categorised into two distinct areas. In the first part, we review deep learning applied in the additive manufacturing domain and highlight gaps in research. The second part covers relevant methodological works including diffusion models, with an emphasis on conditional variants, model architectures and graph neural networks. By organising the related work in this manner, we first cover relevant work from an applied stand-point, before delving into relevant methodological background. This section provides a comprehensive literature review, and positions the contributions of this work as natural extensions both in terms of application and methodology. It is important for the reader to keep in mind the goal of this project when reviewing this section: to learn the graph to layer-image mapping in FDM, with the ultimate aim to forecast and prevent defects.

2.1 Deep Learning for Additive Manufacturing

Automatic Error Detection and Prevention

Additive Manufacturing (AM) processes are complex physical processes and are often unreliable, which has limited their adoption. Therefore, automatic error detection and prevention is an active research area. The types of defects are wide-ranging and can vary depending upon the parameters (see Figure 1.1 and Figure 1.2). For example Baechle-Clayton et al. [2022] notes that adhesion errors in FDM can be influenced by thermal inconsistencies, such as fluctuating heating/cooling cycles of the polymer filament, ambient temperature or improper print bed temperature settings.

Furthermore, extrusion temperature and speed are crucial in FDM printing. The temperature directly impacts the viscosity and adhesion of the filament (Zhu et al. [2022]), whereas speed directly impacts how much material is deposited, affecting its geometry and structural integrity. Although these problems are complex, vision sensors can automatically collect datasets. This makes this problem a strong candidate to be optimised and improved by modern deep learning techniques, and much work has shown success (Johnson et al. [2021], Jin et al. [2020], Gardner et al. [2019], Brion et al. [2022], Brion and Pattinson [2022a,b]).

Saimon et al. [2024] produced a recent literature review which covers how deep learning has been applied to AM. In particular, they highlight two main research problems: understanding the process-structure-property (PSP) relationship and analysing high-dimensional data for in-situ control. The PSP relationship aims to explore how printing parameters and materials affect the microstructure of the final object. PSP research focuses on mitigating defects via improving process parameters, whereas research for in-situ control analyses large volumes of sensor data to mitigate or detect defects during the process in real-time.

One example of research that aims to address the PSP problem is Jain et al. [2024]. By simulating the behavior of 3D lattice structures using graphs, they effectively capture the link between the process parameters, the resulting lattice structure, and its mechanical properties. Yet they rely on a small simulated dataset that was expensive to generate, and the resulting method still cannot be run in real time. Methods applied in-situ can be deployed more flexibly for automatic prevention of errors. One work that tackles the in-situ control problem would be Larsen and Hooper [2023]. They showed that Graph Neural Networks (GNNs) could be used to help detect defects during Laser-Powder Bed Fusion manufacturing. GNNs are a popular choice for this type of task due to the dependence on the underlying geometry. However, this work was limited to training on only a handful of geometries. Other work has shown success in FDM mid-print optimisation using vision sensors. Brion et al. [2022] made use of Convolutional Neural Networks (CNNs) to predict and prevent warping in FDM. They paused the print between each layer, and use a CNN to classify whether or not warping had occurred. They note that correcting for warp is a difficult challenge, as it may occur a long time after the print material has been deposited. Furthermore, it is dependent on a number of complex factors such as layer number. Nonetheless, their classification model allowed them to adapt the bed temperature, fans and printing speed to save prints that would warp without intervention. Some limitations of this work include the need for extensive labelled data, and the way that parameter updates were applied globally rather than to localised regions. It is challenging to interpret and localise output from classification models - generative models allow richer outputs such as images which may remedy this.

Generative Modelling in AM

Generative models have already been applied to AM. Saimon et al. [2024] highlight that Generative Adversarial Networks (GANs) have been used in many settings including: topology optimisation, manufacturability assessment and process monitoring and control (Hertlein et al. [2021], Almasri et al. [2022], Guo et al. [2021], Song et al. [2023]). Typically, GANs are applied during image-based monitoring to enhance low-fidelity images or to generate new synthetic data for training as in Li et al. [2022]. Yet in Song et al. [2023] they use a conditional GAN to predict the final complex pore microstructure for selective laser melting. The authors use a GAN auto-encoder model conditioned on the process parameters such as laser power or scanning velocity. However, GANs are notoriously unstable to train, and diffusion models have emerged as a stable alternative with a superior sample quality (Ho et al. [2020], Dhariwal and Nichol [2021]).

Diffusion models have recently been explored in the wider life and material sciences domain, for example they were recently used in AlphaFold 3 (Abramson et al. [2024]) to predict protein structure and their interactions. However, they have only been lightly explored in the AM domain. Ogoke et al. [2023] recently applied diffusion models in Laser-Powder Bed Fusion to map low-fidelity simulation information to high-fidelity versions (i. e. super resolution). Additionally, conditioned diffusion models were applied by Jadhav et al. [2023] to reduce the cost of finite element analysis when simulating stress distribution for a given geometry. These works both employ the U-Net CNN model architecture, which tends to be the most popular backbone for diffusion models (Ho et al. [2020]).

Highlighting Gaps in Research

Although prior work has proposed the use of diffusion modeling in manufacturing, to the best of my knowledge, no work has applied it to predict the final layer image in FDM. This project explores this gap in research. In particular, the approach allows a hybrid solution to PSP predictions and in-situ control, since images are generated on a per-layer basis, which in turn would allow for mid-print optimisation. In addition, this work does not require manual data annotation, avoided by directly learning the mapping between process parameters and resulting layer images. This allows us to collect a more diverse and interesting dataset containing many geometries with varying parameters. Moreover, by producing a high-resolution output, it possible to get highly localised and interpretable feedback for different parameters and geometries. Lastly, while graphs have been suggested as a natural choice for representing complex physical properties, no research has integrated spatially aligned graphs directly into a diffusion model for image generation.

In this section we outlined how deep learning has been applied in AM, focusing on research involving vision-based technology (Brion et al. [2022]), graph neural networks (Larsen and Hooper [2023]), and diffusion models (Ogoke et al. [2023]). In the following section, we elaborate on these key methodologies and discuss the relevant theory, all of which play a pivotal role in solving the task of generating final layer images from the printing process graph.

2.2 Methodological Background

This section condenses key methodological concepts, including diffusion models, architecture choices (e.g., UNet, ViT), and GNNs. These topics are inherently complex, the aim of this summary is to capture the essential principles and present the key mathematical foundations required for each topic. Please note that due to the breadth of each topic, this is not an exhaustive treatment but a focused discussion to facilitate understanding of the approach presented in the subsequent chapter.

2.2.1 Diffusion Models

Diffusion models, first proposed by Sohl-Dickstein et al. [2015], and later popularised by Ho et al. [2020] and Song and Ermon [2020] are a class of generative model that learn a mapping between two distributions. Diffusion models map between a complex data distribution (e.g. images) and a simple distribution such as isotropic Gaussian noise by defining a forward and reverse mapping process. The forward process slowly corrupts the data distribution by adding noise over time. At the end of this process, the corrupted data becomes indistinguishable from pure noise. In contrast, the corresponding reverse process transforms the noise back into the data by slowly removing the noise that was added.

Turning data into noise is trivial, yet turning noise into data is difficult, therefore deep learning is employed to approximate the denoising reverse process. Once trained, diffusion models can generate new samples from the data distribution by starting from noise and iteratively denoising with the trained denoiser. These models have shown success by turning the difficult problem of data generation into a series of much simpler supervised denoising problems. This has led to diffusion models surpassing GANs for image synthesis (Dhariwal and Nichol [2021]), and producing text-guided generated artworks that have captured the public imagination (Saharia et al. [2022], Balaji et al. [2023], Podell et al. [2023], Ramesh et al. [2022]).

Despite diffusion's ubiquity and impressive performance, many authors struggle to provide a clear and concise mathematical explanation of the framework (Turner et al. [2024]). This is partly down to the many interpretations and variants such as: the variational lower bound vs maximum likelihood perspective (Turner et al. [2024]), or the markov-chain vs differential equation perspective (Song et al. [2021]). In this vein, many works have tried to elucidate the design space of diffusion models; most notably the work of Karras et al. [2022], which made several interesting contributions such as using 2nd order solvers to speed up inference. In the following section I will provide a concise mathematical foundation on diffusion models from the same continuous time perspective used in Song and Ermon [2020] and Karras et al. [2022]. The continuous-time variant is chosen since it offers a natural way to trade off quality with inference speed, and tends to produce higher quality samples.

Mathematical Preliminaries

Consider a large number of training samples $\{\mathbf{x^1} \dots \mathbf{x^N}\}, \mathbf{x} \in \mathbb{R}^d$ drawn from an underlying data distribution $p_{data}(\mathbf{x}; \sigma_{\text{data}})$. It is logical that by iteratively corrupting this distribution with additive i.i.d Gaussian noise $\sim \mathcal{N}(0, \sigma^2\mathbf{I})$, over time for $\sigma_{\max} >> \sigma_{\text{data}}$ the distribution $p(\mathbf{x}; \sigma_{\max})$ is indiscernible from pure noise (as shown in Figure 2.1). This sequential operation outlines the forward process, which transforms the data distribution to the noise distribution.

Due to the mathematical nature of sequentially adding Gaussian noise, it is possible to sample a noisy image $\mathbf{x_t}$ analytically. At a given noise level σ_t, this is done by first sampling pure noise $\mathbf{x_0} \sim \mathcal{N}(0, \sigma^2_{\max}\mathbf{I})$ and combining it with the true image $\mathbf{x_1} \sim p_{data}$ with respect to a pre-determined noise schedule.

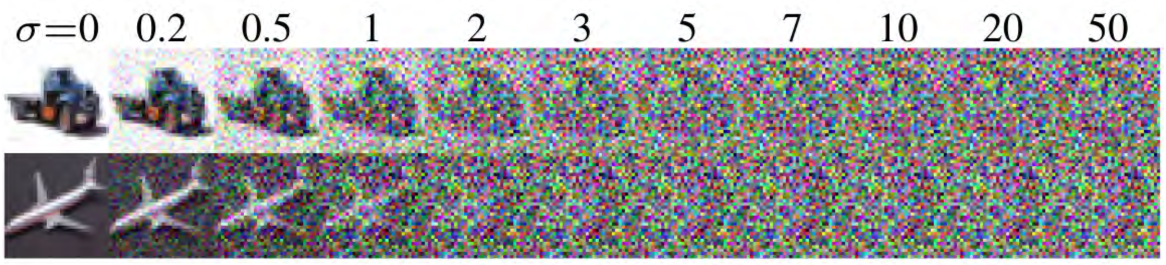

Figure 2.1: **Karras et al. [2022] Figure 1.a Noisy images drawn from** $p(\mathbf{x};\sigma)$. Images corrupted with varying levels of additive Gaussian noise (normalised for cleaner visualisation).

It is possible to formulate diffusion with either discrete or continuous levels of noise σ. For the continuous and more generalised case, we consider a continuous time variable $t \in [0,1]$ that indexes into the noise schedule. Such that for a given noisy sample $\{\mathbf{x_t}\}_{t=0}^{1}$, both requirements of $\mathbf{x_0}$ being noise and $\mathbf{x_1}$ being the data are met. With this requirement being met, we can model the diffusion process (the mapping from data to noise distribution) as the solution to a differential equation. In Song et al. [2021], they introduce both Stochastic Differential Equations (SDE) and Ordinary Differential Equations (ODE) variants, but find the SDE to produce higher quality samples. They propose the following Itô SDE:

$$d\mathbf{x} = \mathbf{f}(\mathbf{x},t)dt + g(t)d\mathbf{w} \tag{2.1}$$

Where $\mathbf{w}$ is a standard Weiner process (e.g. a stochastic process such as brownian motion), $g(t)$ is a scalar function which produces the diffusion coefficent at time t, and $\mathbf{f}(\mathbf{x},t)$ models the data drift coefficient over time. One manifestation of equation 2.1 could be a small decay of the data plus a small amount of random noise. Furthermore, Song et al. [2021] introduce a corresponding reverse-time SDE that maps noise back towards the data distribution:

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x},t) - g(t)^2\nabla_{\mathbf{x}}\log p_t(\mathbf{x};\sigma(t))\right]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}} \tag{2.2}$$

Here, dt is an infinitesimal negative timestep and $\bar{\mathbf{w}}$ is the reverse-time Weiner process. Crucially, this formulation relies on $\nabla_{\mathbf{x}}\log p_t(\mathbf{x};\sigma(t))$, which is known as the *score function* (Hyvärinen [2005]). Here, the score function is a vector field that points towards higher density areas of data for the given noise level. Therefore new data samples can be generated by simulating equation 2.2 starting from noise. Off-the-shelf numerical SDE solvers can be used to simulate this, for example Euler-Maruyama. See Figure 2.2 for an overview of these forward and reverse processes simulated.

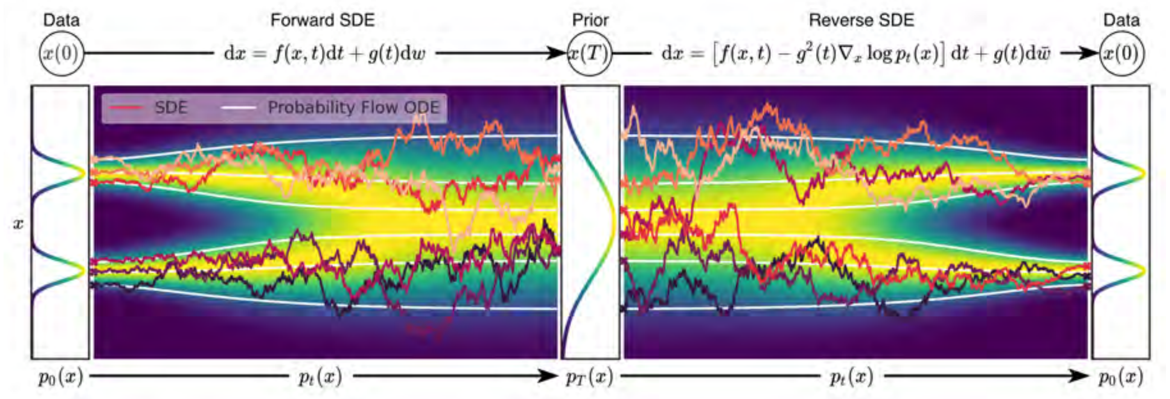

Figure 2.2: **Song et al. [2021]** **Figure F.2** An illustrive graphic of the mapping between the data and noise distributions. The left hand-side illustrates the forward process described by 2.1, whereas the right the reverse process described in 2.2.

The work of Karras et al. [2022] thoroughly investigates several aspects of the score-based diffusion formulation presented by Song et al. [2021]. This includes: 1) the importance of the noising schedule, 2) the performance gap between ODE and SDE variants, 3) the choice of solver and higher order solvers and 4) other practical improvements such as whether to predict the noise or the image. In their work they first explore the ODE variant, in this case the only source of randomness is the initial noise, and it allows a clearer examination of the process before stochastic sampling is re-introduced. This revealed that the noise schedule defines the shape of differential equation solution equations, and therefore errors induced by truncation error during sampling can be reduced by selecting an appropriate schedule. Secondly, they show that a 2nd order solver such as Range-Kutta produces a better computational trade-off. Additionally, they prove that the SDE variant produces higher quality samples since the stochasicity pushes the sample towards the desired marginal distribution at time t, actively correcting errors made in previous sampling time steps. Finally, they propose a network pre-conditioning connection that allows the model to predict either the image or the noise (or an adaptive mix of signal and noise). These improvements and practical considerations led to impressive performance increases in both training convergence, inference and sampling quality. Readers should refer to Appendix A.1 for a more detailed treatment of Elucidated Diffusion, the chosen diffusion variant.

Conditioning diffusion models on conditioning variables $\mathbf{c}$ changes the task from $p(\mathbf{x}; \sigma(t))$ to $p(\mathbf{x}|\mathbf{c}; \sigma(t))$. Typically, this is done explicitly through concatenation be it a text embedding, a low-res image or another signal. However, other common approaches either inject conditional signals through cross-attention or use guidance methods during sampling (Po et al. [2023]). Alternatively, the ControlNet paper by Zhang et al. [2023] achieved unprecedented spatial-conditional generation by incorporating an additional network alongside a large, pre-trained text-to-image diffusion model with frozen parameters. This underscores the potential of diffusion models in enabling advanced spatial conditioning for image generation.

In this section, a high level overview of the mathematical preliminaries for diffusion was introduced. In the next section, model architectures for diffusion are explored.

Architecture Choices

Traditionally, diffusion models (Ho et al. [2020]) employ encoder-decoder style CNNs such as the U-Net proposed by Ronneberger et al. [2015]. These models have a track-record of efficiently learning fine-grained details from relatively small datasets. They process input images via a CNN encoder-decoder connected via skip connections (see Figure 2.3). In diffusion, they are used to predict the added noise, the original image or some combination of the two (as in Karras et al. [2022]).

Alongside other generic advances in deep learning architectures, the U-Net has been specifically improved for diffusion by: making them more efficient at high resolutions, conditioning them for controlled generation and augmenting them with self-attention for improved fidelity (Ho et al. [2020], Dhariwal and Nichol [2021]). One work which encapsulates all three of these improvements is the Imagen paper (Saharia et al. [2022]), where the authors generate high-quality 1024x1024 images (see Figure 2.5) via a cascade of conditional diffusion models with U-Net backbones. Specifically, they improve the U-Net's memory efficiency with no performance degradation, by down-scaling images before the ResNet blocks and by using more ResNet blocks at lower resolutions compared to higher ones.

The Imagen paper is also a great example of conditional generation i.e. via text and low-res images. Conditional generation is most commonly done by explicitly providing the U-Net additional information. For example, in super-resolution the low-resolution image is provided to the model generally via concatenation to the noisy input (see Figure 2.3 for a basic overview). This approach provides the model with important spatial and semantic information throughout generation and has led to unprecedented success (see Figure 2.4 and Figure 2.5 for some examples).

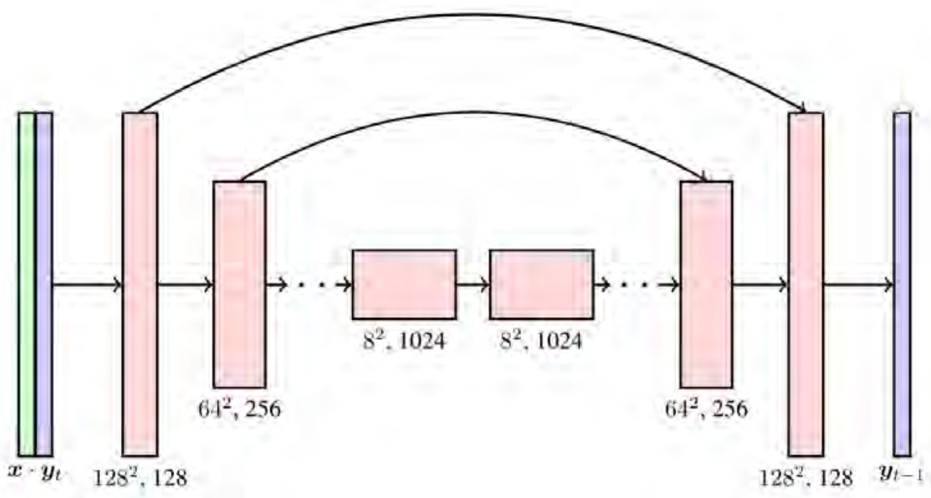

Figure 2.3: **Saharia et al.** [**2021**] **Figure A.1** A basic diagram of the U-Net architecture used for super-resolution diffusion. Input images are concatenated with a low-resolution conditioning image before processing. The encoder downsamples images into feature maps via ResNet blocks (He et al. [2015]), typically halving the resolution and doubling the channels at every scale. This down-sampled feature map is then passed to the decoder through a bottleneck, before being upsampled and processed by more ResNet blocks. Each corresponding scale in the encoder and decoder is connected via a skip connection which helps the network learn effectively at multiple scales.

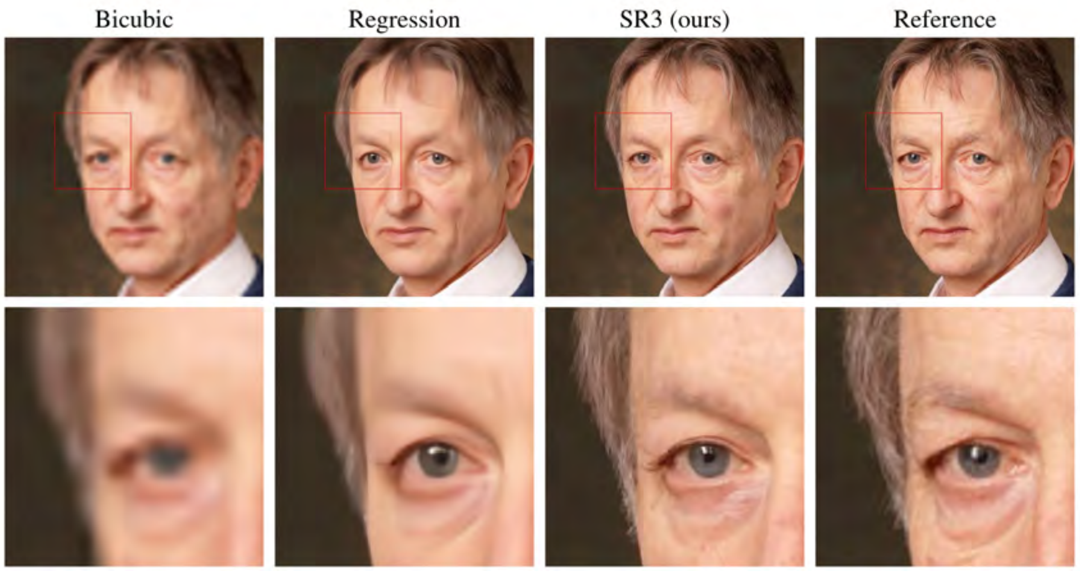

Figure 2.4: **Saharia et al.** [**2021**] **Figure F.4** [**top row only**] An example of conditional image super-resolution using diffusion, and comparisons to other methods.

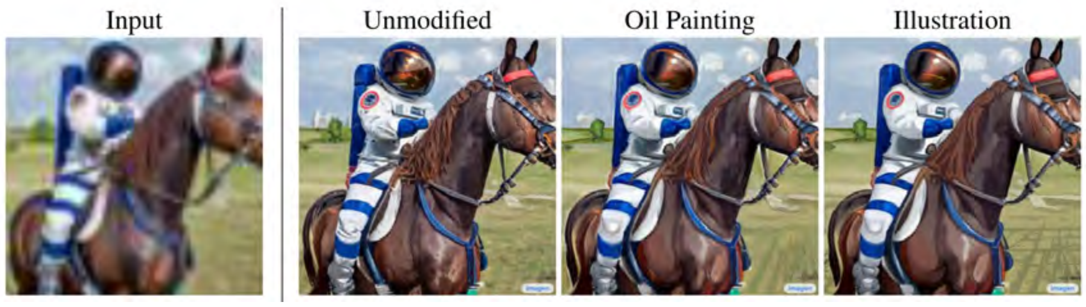

Figure 2.5: **Saharia et al. [2022] Figure A.12 [top row]** An example of conditional image super-resolution in the Imagen paper, each image is generated by providing a noisy low resolution image during generation alongside a text embedding.

It is important to observe that tasks such as super-resolution are ill-posed; there are many possible high-resolution images for each corresponding low-resolution counterpart. In purely generational tasks, models are often adapted to increase sample diversity (and therefore distribution coverage). For example, the Imagen paper corrupts the low-resolution image which increases sample diversity. Additionally, classifier-free guidance (Dhariwal and Nichol [2021]) is used, meaning the model only partially follows the text-conditioning which improves sample quality.

Many similarities can be drawn between super-resolution and predicting the final-layer image given process parameters. The complex molecular dynamics of FDM can result in behaviour that is very hard to predict. However, given suitable sensor information it should be possible to generate conditioned samples from this underlying distribution. However, this raises the research question of how exactly to condition images using space-time process parameter graphs.

A promising alternative to CNN-based architectures would be Dosovitskiy et al. [2021]'s vision transformers (ViTs), which were shown to be scalable image generators for diffusion in Peebles and Xie [2023]. Although there are many attractive properties to ViTs, in their basic form they suffer from quadratic input complexity which makes them difficult to scale to higher resolution without resorting to diffusion in latent space as in Peebles and Xie [2023]. Compressing images into latent space inevitably introduces errors, this is acceptable when the aim is improving visual quality (i.e if the target metric is FiD), but is less appropriate for reconstruction tasks.

2.2.2 Graph Neural Networks

Geometric deep learning is principally interested in extending deep learning techniques to non-regular data-spaces such as graphs. Just as CNNs excel by operating on local pixel neighbourhoods, Graph Neural Networks (GNNs) achieve strong results by operating on the local neighbourhoods of graph nodes. As Bronstein et al. [2021] state, geometrical approaches leverage low-dimensional structures (like grids in images or molecular properties) and their symmetries (e.g., translation, rotation), enforcing invariances critical for tasks like molecule classification. Additionally, this approach effectively handles high-dimensional and flexible data types, such as social network graphs with varying degrees of node connections. This geometric interpretation can applied to a wide-array of architectures, not just those that operate with graph data. For example, image processing with CNNs can be viewed through this lens as it introduces translational invariance via shared local weights (Bronstein et al. [2021]).

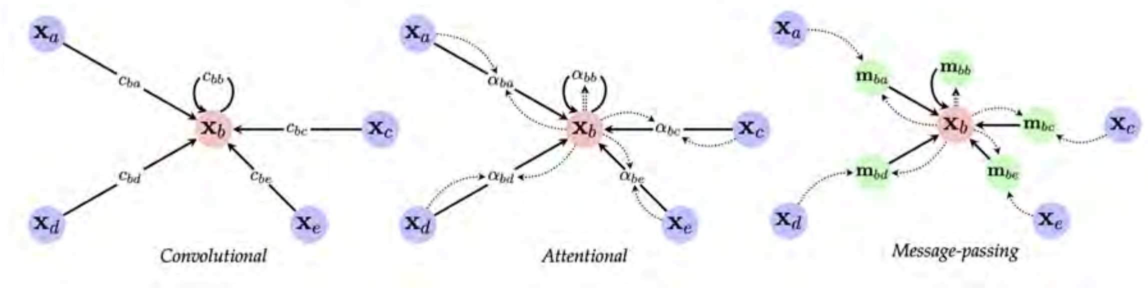

Figure 2.6: **Bronstein et al.** [**2021**] **Figure F.17** Modern GNN architectures can be broadly categorised into three types. The convolutional variant updates node features based on the features of neighboring nodes, weighted by a learned constant. The attentional variant uses an attention mechanism to update node features by considering the relationship between sender and receiver nodes. Lastly, the message passing variant computes vector-based messages between sender and receiver nodes to update their features.

As discussed in section 2.1, graphs are a natural choice for operating on space-time graphs that contain AM parameters. This is because the underlying geometry includes important information that is important to the task, and there are many desirable equivariances such as rotation and translation that should be enforced when computing with such input. Furthermore, they naturally handle different sized graphs without resorting to zero-padding. They have exhibited success in both the AM domain (Larsen and Hooper [2023], Jain et al. [2024]) and wider life and material science domain

(Abramson et al. [2024]) even with small datasets. This is often attributed to their explicit focus on not only feature nodes, but the relations between them.

One of the most significant contributions in this area is the work of Gilmer et al. [2017], who introduced a GNN design for quantum property prediction using small datasets. This work shows that the vast majority of GNN architectures fall into the same family of architectures known as: *message passing* GNNs. As Bronstein et al. [2021] discuss, GNNs broadly fall into one of three variants, each increasing in complexity: convolutional, attentional and message passing, see Figure 2.6 for an overview. Attentional and convolutional networks can be viewed as special cases of Message Passing Neural Networks (MPNNs) by applying specific restrictions.

All variants follow the same basic premise: an updated node embedding is computed with respect to the local neighbourhood. Permutation invariance - the order in which neighbours are considered is not relevant - is ensured by some permutation-invariant operator (defined with $\oplus$) such as averaging. In the message-passing variant, an updated node ($\mathbf{h}_u$) is calculated:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right) \tag{2.3}$$

Where ϕ is a learnable function and ψ is a learnable message passing function that computes the message from local neighbour x_v to pass to node x_u (Bronstein et al. [2021]).

Graph data is often represented by more than a single node or edge type. Heterogeneous graphs - those with more than one of type of node or edge - have also been explored in the literature. For example, Hu et al. [2020] introduced the Heterogeneous Graph Transformer (HGT) architecture that learns different transformations for each node and edge type in the graph (commonly known as a meta-path).

Furthermore, GNNs have also been used in the computer vision domain. Han et al. [2022] demonstrate how graphs can be effectively used in image classification tasks by splitting the image into patches and connecting them via nearest neighbours. This work showed that graphs can be powerful general architectures in the vision domain. GNNs have also been applied to diffusion models before, for example, Yang et al. [2022] and Farshad et al. [2023] utilise scene graph conditioning using diffusion. Both of these approaches adopt some form of graph encoder (typically graph convolution Kipf and Welling [2017]), to condition the diffusion process. It is worth noting that scene graphs are very different types of graphs when compared to space-time graphs. Scene graphs are designed to contain coarse structures of the scene images, as opposed to the fine-grained accuracy of space-time graphs.

Chapter 3

Method

This chapter explores the practical and theoretical approach of the project, which is split into three sections. The first part explains the dataset, the collection process, pre-processing and synthetic data generation. The next section investigates the potential strategies to condition images with spatially aligned graphs, and describes two proposals in detail. The final section outlines the model architecture along with proposed modifications that were required to facilitate graph-to-image generation.

3.1 Data

This section examines input space-time graphs and the data collection process, along with pre-processing and augmentation strategies. Due to time constraints, the volume of real data was limited, so much of the initial work focused on synthetic data. The generation of synthetic data is also covered in this section.

3.1.1 Space-Time Graphs

Space-time graphs for AM can be interpreted as a sequential list of instructions that are sent to a 3D printer. In their most basic form, each node is only connected to the subsequent node. Each command or 'node' in the graph indicates where and how the printer should deposit material, for example one hypothetical command may read {`x` : 0.5, `y` : 0.5, `z` : 0, `flow_rate` : 100, `active` : 1} indicating that the nozzle head should move from its current position to position `x=0.5,y=0.5` along the print bed with a flow rate of 100% the default calibrated value, where the active command allows the printer to move the nozzle head position with or without depositing material. It is easy to draw parallels with plotting software that renders lines between sequential coordinate pairs. However, there are complex physical interactions that underpin each stage of the AM process. This results in a mapping between space-time graph and layer images that is not bijective. See Figure 3.1 for some examples of this phenomenon.

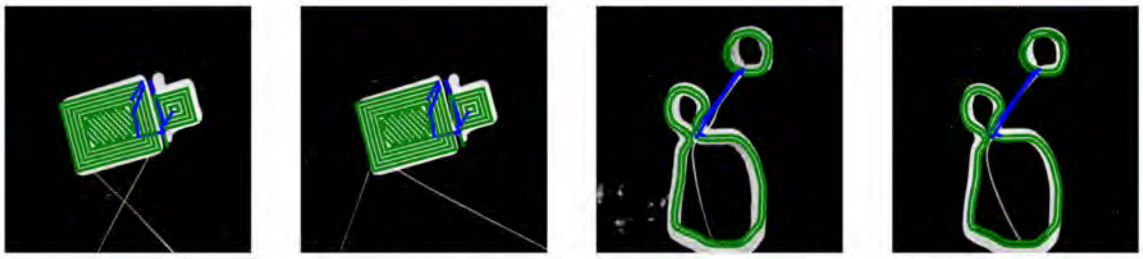

Figure 3.1: **Space-time Graphs Overlaid on Resulting Layer Images.** This figure shows two space-time graphs overlaid on the resulting layer images. Green lines show active lines, whereas blue indicate inactive lines where the nozzle is moved but no material is deposited i.e. the island in the right hand side images can only be formed through a connection via an inactive edge.

Representing the set of instructions as a graph is a natural choice because it exploits the underlying geometry and its associated symmetries (as discussed in 2.2.2). It also supports a variable number of nodes and edges without explicit padding. Given the number of nodes in graphs can vary significantly between prints, adopting a framework that accommodates this is advantageous (implementation details discussed next in 3.1.2).

As mentioned, this is an unexplored research area, so only graphs that represent a single-layer were investigated. However, the graph representation can trivially extend beyond this restriction. In fact, it allows a way to represent the projection of multiple layers in the form of overlapping nodes. A dataset of input graphs were provided by the CAM group, which was generated by taking slices from existing 3D models. These were used to generate a dataset of resulting layer images using a 3D printer. A secondary dataset was collected towards the end of the project for training a final model. To add variety, half of this data was created using new space-time graphs derived from Quick Draw - a line drawing dataset (Ha and Eck [2017]).

3.1.2 Data Collection & Pre-Processing

This section describes data collection using the graphs outlined in the previous section. The experimental set-up is as follows: the *Creality CR-30 3DPrintMill* 3D Printer was retro-fitted with an aerial-view camera *Raspberry Pi High Quality Camera with CS-mount 25mm/F1.2 lens* and a *Raspberry Pi 4 Model B 8 GB model* used for remote control. The stock 45-degree nozzle head was replaced with one perpendicular to the print bed, preventing other prints from being scraped during collection. A custom mount and software adjustments, such as multiplying y by cos(45), were required to accommodate this change. This setup was specifically configured in the climate-controlled IfM additive manufacturing laboratory. The printer's moving print-bed allowed for automatic printing of multiple layers before manual cleaning was necessary. Graphs were converted to g-code - the standard language for 3D printing - before being sent to the printer. In total an initial dataset of $\sim$ 1500 images were collected by a member of the CAM team. At a later stage I took over the collection process and collected $\sim$ 1000 additional samples. All data is made available here. Images were captured in batches of 55 prints, taking ~90 minutes per batch to run and reset. The captured images have a resolution of 1280x720. Each graph geometry was printed up to five times with varying flow rates to study the distribution of final layer images. The flow rate was randomly perturbed with offsets from an initial random value for each graph. Flow rate variations are defined as the percentage of material exiting the nozzle relative to the default settings established during calibration (this builds on existing work from IfM Margadji et al. [2024]). Therefore, the focus of this study is to understand the PSP relationship between input geometry, flow rate, and the characteristics of the final layer.

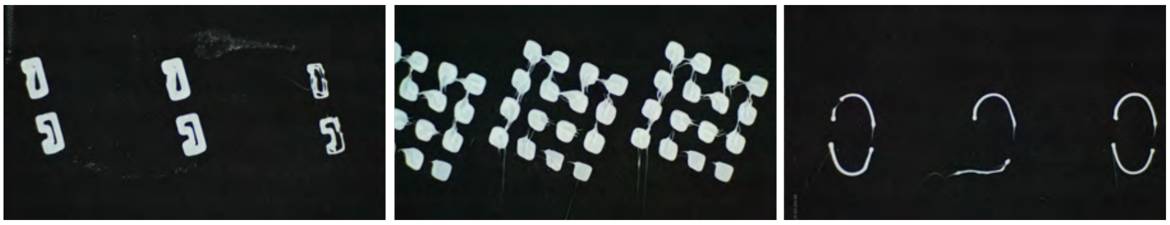
Figure 3.2: **Layer Images Before Pre-Processing.** Displays layer images prior to pre-processing; multiple neighbouring prints are in view and residue is visible on the print bed.

After collection, the images were pre-processed to remove residue and neighboring prints, and to align the graph and image coordinates. Without alignment, neighboring prints could appear in the images, and the target layer might be out of view. See Figure 3.2 to see what images looked like prior to pre-processing and registration.

To align the images with the graph coordinates, 2D affine transformation matrices were estimated. Affine matrices allow for scaling, rotation and translation. This makes the conservative assumption that perspective effects are negligible given the flat bed and aerial view. However, due to belt slippage and other mechanical inaccuracies it was not appropriate to apply a single affine matrix to all samples. A more sophisticated approach was required in order to minimise discrepancies. Yet, estimating robust affine matrices between the graph and its corresponding image is not trivial. This is because images can contain severe errors when compared to the graph; neighbouring prints share near identical features and belt slippage alters alignment mid-experiment (see Figure 3.3). These problems mean that traditional feature matching algorithms ultimately fail to correctly register images. This raised two issues: 1) how to find robust correspondences between the graph and the captured image, 2) how to safely handle failure cases.

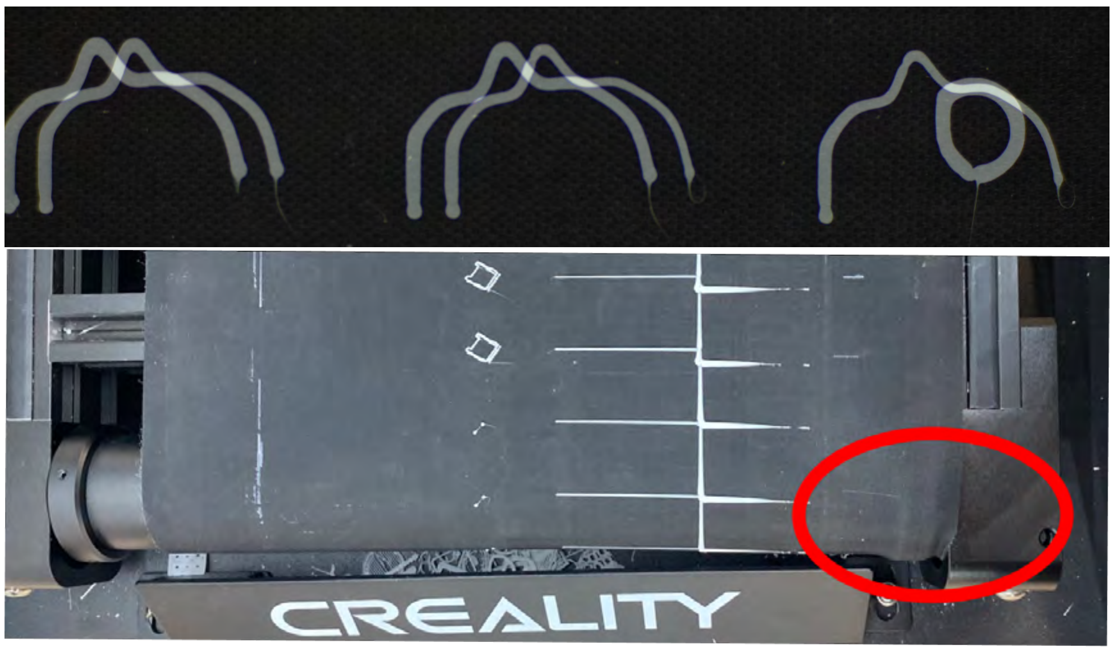

Figure 3.3: **Belt Slippage and Belt Jamming.** This top image overlays two subsequent prints on top of one another, notice that severe belt slippage can occur even between a single sample. The bottom image shows a belt jam, notice in the highlighted red region the belt has moved over the top of the belt roller.

A new approach to find robust correspondences between graph and imperfect layer images was developed. The approach aimed to find approximate outline shapes of each central print image and its corresponding graph, and then compared them with Enhanced Correlation Coefficient (ECC Evangelidis and Psarakis [2008]). Importantly, this produced a highly robust way to estimate scale, translation and orientation, even between two severely distinct images that represent the same underlying geometry. Other traditional methods are not explicitly compared to this approach because they result in a majority of catastrophic failures. An overview of this affine matching system is displayed in Figure 3.4 on the page below, algorithm details can be found in Algorithm 1.

Algorithm 1 Robust Image Alignment for Graph / Real Image pairs

```
Require: Unaligned Graph / Image Pairs: graph, real_image
 1: graph_image ← render(graph)
 2: real_image ← threshold(crop_and_pad(real_image))
 3: for image in {graph_image, real_image} do
 4:     morphological_close(image)
 5:     adaptive_threshold(image)
 6:     find_middle_contours(image)
 7:     fill_contour_outline(image)
 8:     blur(image)
 9: end for
10: affine_matrix, EEC ← FindAffineWithEEC({graph_image, real_image})
11: if EEC > threshold then
12:     affine_matrices ← store(affine_matrix)
13: end if
```

The second challenge was robustly handling failure cases. Although belt slippage has significant impacts on image translation, the system was robust in terms of scale and rotation. Therefore, a default scaling and rotation matrix was found using all available samples. This was done by filtering the set with RANSAC to remove outliers (Fischler and Bolles [1987]), and taking the average. To find more accurate translations a three-tier fall back approach was then implemented. Translations were calculated on a per-sample basis if the sample had high ECC (e.g. 90+). For samples below this threshold the translations of neighbouring prints from the same geometry were averaged. Finally, if that failed then the per experiment average was used i.e. the 55 neighbouring prints. Again RANSAC was used to remove outliers from this per-run mean. Approximately 88.7% of dataset samples are processed by the first method, 6.1% by the second stage and 5.2% by the third stage. To further analyse this approach Figure 3.5 displays translation predictions. We notice clear clusters displayed on an experiment/sample basis showing the validity of this method. Furthermore, in extreme

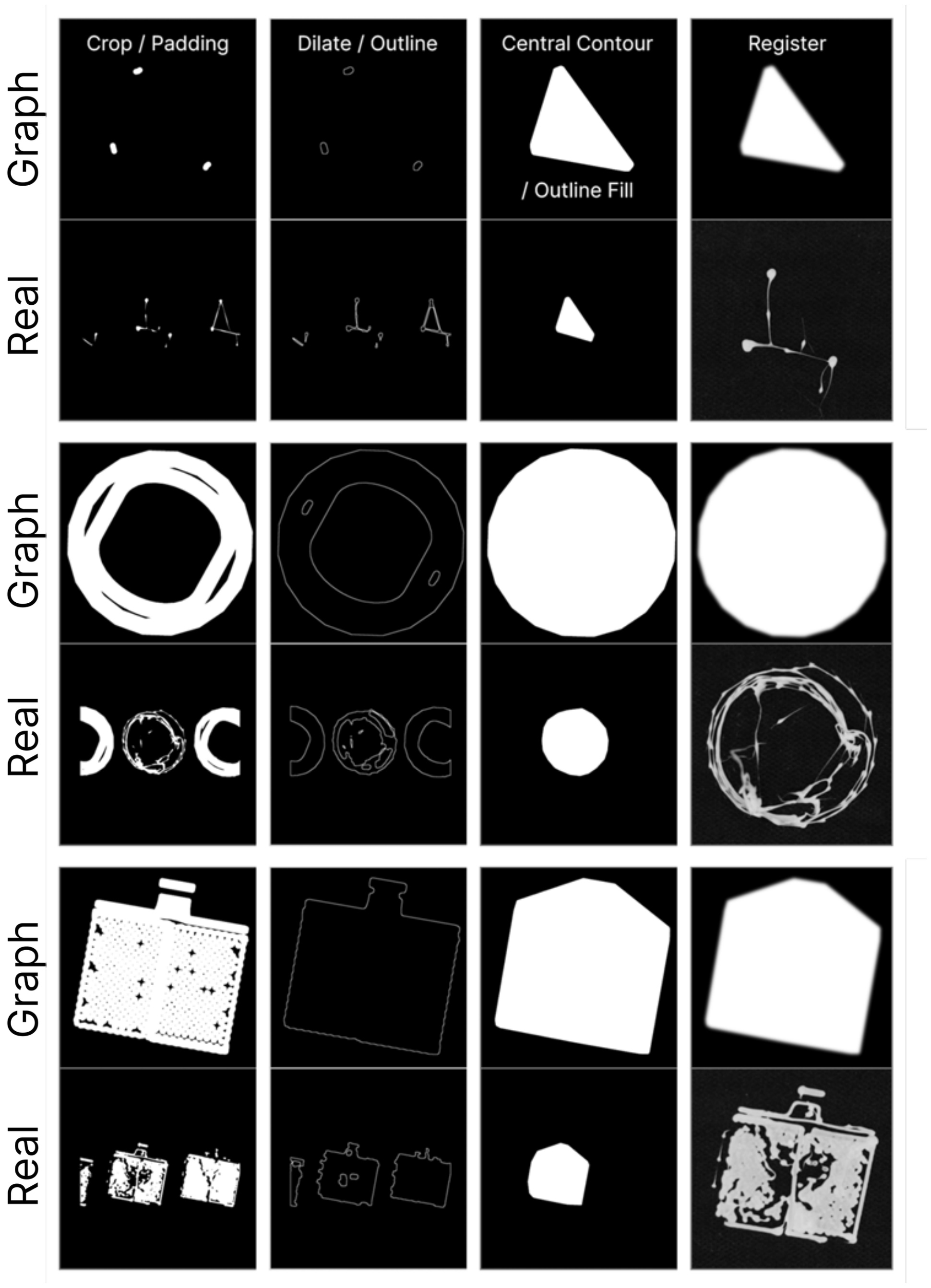

Figure 3.4: **Robust Affine Matching System.** Examples of how graph/print images are processed for affine estimation. Graphs are plotted to target renders, and real images are padded to match the resolution. Both have a morphological closing operator applied, are blurred and adaptively thresholded. This connects inter-print shapes and results in approximate outlines. Image contours that don't intersect with a small central region of the image are discarded. This step discards neighbouring prints whilst ensuring the entire shape in the center is retained. The remaining contours are sorted into an outline contour and filled, this new image estimates the outline geometry of the entire print. ECC is used to estimate an affine matrix between the two images. The ECC coefficient is used to find matches that had high confidence and those that failed.

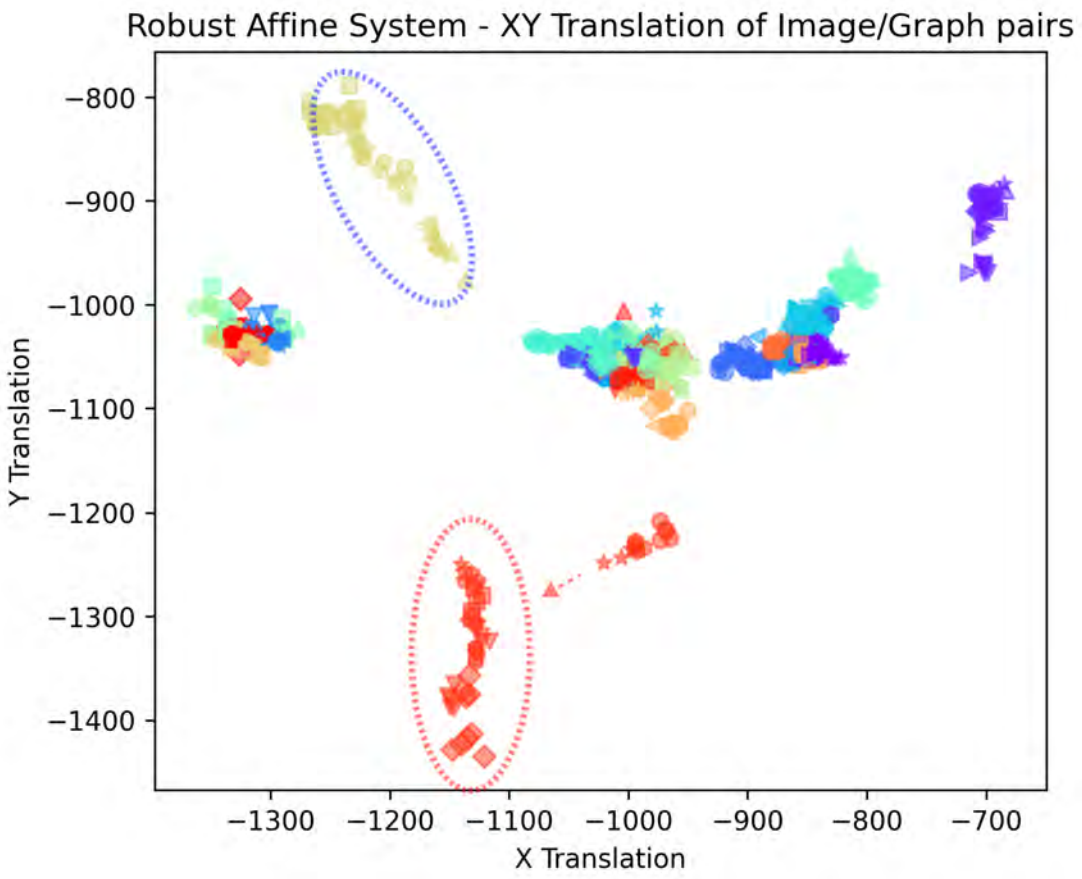

Figure 3.5: **Robust Affine Translation Estimation Scatter Plot.** This represents the affine translation estimations over the dataset, experiments of 55 samples are colour-coded, whereas the five neighbouring samples of the same geometry share marker type. Indicated in dashed ellipses are severe belt jamming/slippage incidents, which this method can successfully handle e.g. the blue ellipse indicates a belt jamming incident. The red to the left, the blue to the right. Note: the small red arrow denotes a major slippage event. The data points within the red highlighted region show results from the experiment with a belt jam failure displayed in Figure 3.3.

incidences of belt jamming we see the system track the translation in the direction of the belt jam. See Figure 3.6 for examples of pre-processed images, and Figure 3.7 for aligned image examples with the graph overlaid.

Once images were correctly registered they were threshold masked at 20% to eliminate background texture and left-over residue. Finally, the image was resized to the desired resolution, and normalised to -1 to 1. Augmentation was then applied to the images and their corresponding graph coordinates. The augmentations were restricted to Euclidean transformations only. This was to ensure that underlying information was not distorted e.g. scaling an image would destroy the relationship between flow rate and line width in the image. Random rotations of up to 360 degrees, and shifts up to 25% of the image dimensions were applied. These parameters are more extreme than typically seen in a natural image deep learning pipeline. However, since a printer can start anywhere and in any orientation, these values are appropriate. See Figure 3.8 for some examples of pre-processed and augmented data, samples were chosen to match the same data points in Figure 3.7 .

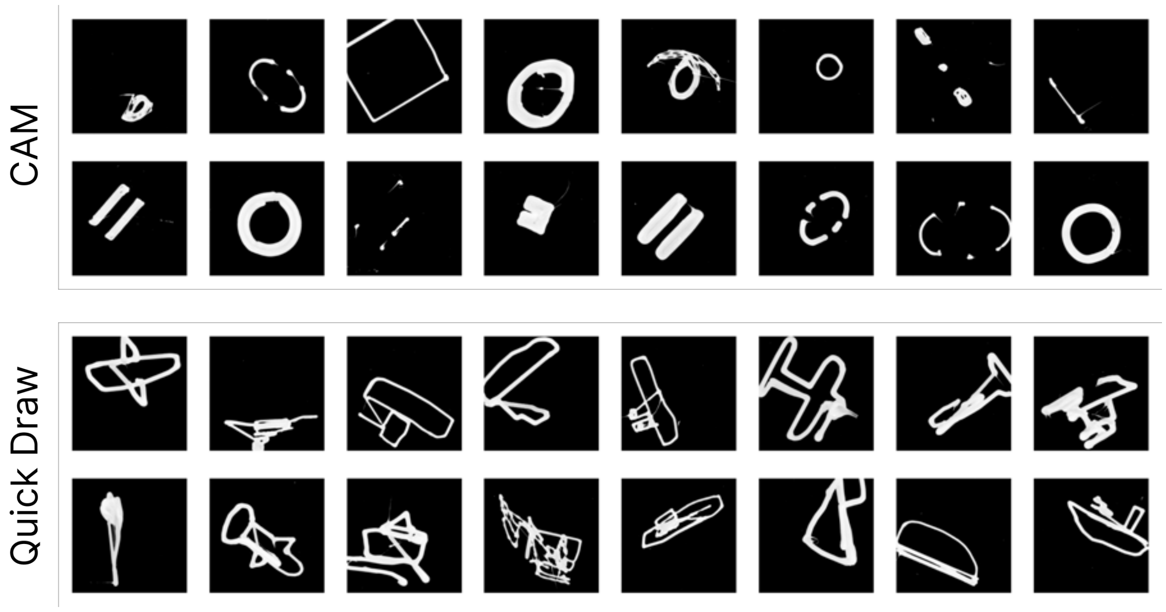

Figure 3.6: **Pre-processed Samples for Each Dataset.** Images show captured images from the dataset. Top rows show data from graphs provided by CAM. Bottom rows show data from graphs derived from Quick draw dataset (plane and ship class subsets).

After the image and graph pairs were pre-processed they were stored in custom HeteroData objects from *PyTorch Geometric* (Fey and Lenssen [2019]). This heterogenous representation allows for flexible numbers of node and edge types. This automatically handled variable length graph representations efficiently. This is done by stacking the graph adjacency matrices into one large adjacency matrix and stores them in a sparse matrix (which induces less computational and memory overhead). Isolated sub-graphs cannot pass messages between them via GNN operators, therefore it is safe to group graphs in this manner. At run-time, this data representation allowed easy access to each graph and corresponding image, but importantly enabled the entire stacked graph to run efficiently through GNN models.

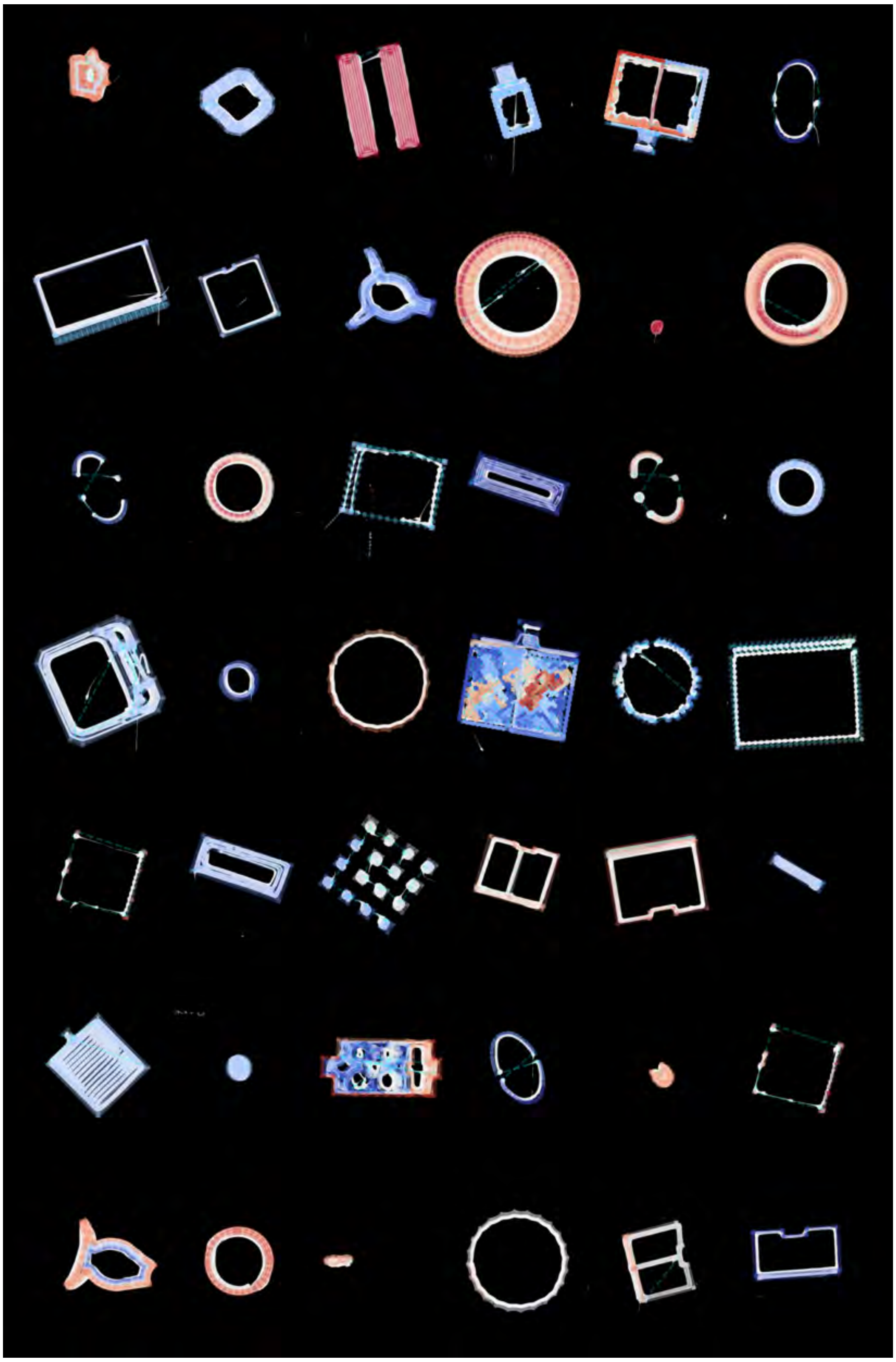

Figure 3.7: **Uncurated Samples of Image-to-Graph Alignment**: Graphs with color-coded flow rates are overlaid on the aligned layer images. While the system is robust, it does not achieve perfect alignment. Without this system, multiple layers would appear in a single graph, leading to incorrect model predictions. Dashed lines show inactive edges.

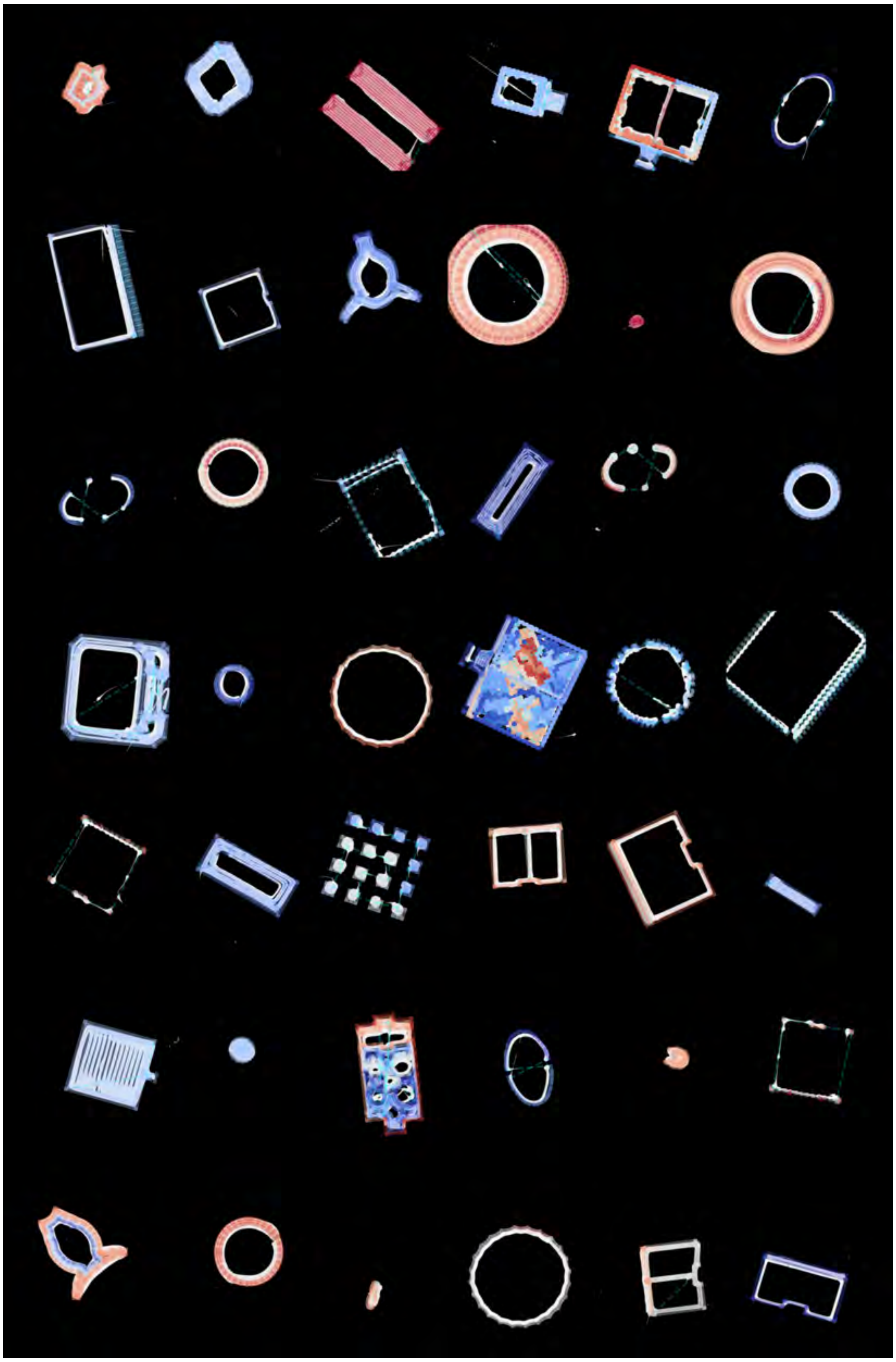

Figure 3.8: **Augmented Images.** This figure presents the same samples from Figure 3.7, but with random rotation and translation applied to the graph and the image.

3.1.3 Synthetic Data Generation

The collection of real-world data is time-consuming and expensive. Additionally, the most successful generative diffusion models have access to millions of images as opposed to a few thousand. Transfer learning is a common technique in deep learning where a model is trained on a related task, then a second training stage fine-tunes the model parameters for a more specialised downstream task. It was hypothesised that training the graph-to-image model on synthetic data would act as a valuable pre-conditioning stage, the results of which can be found in Section 4.2.

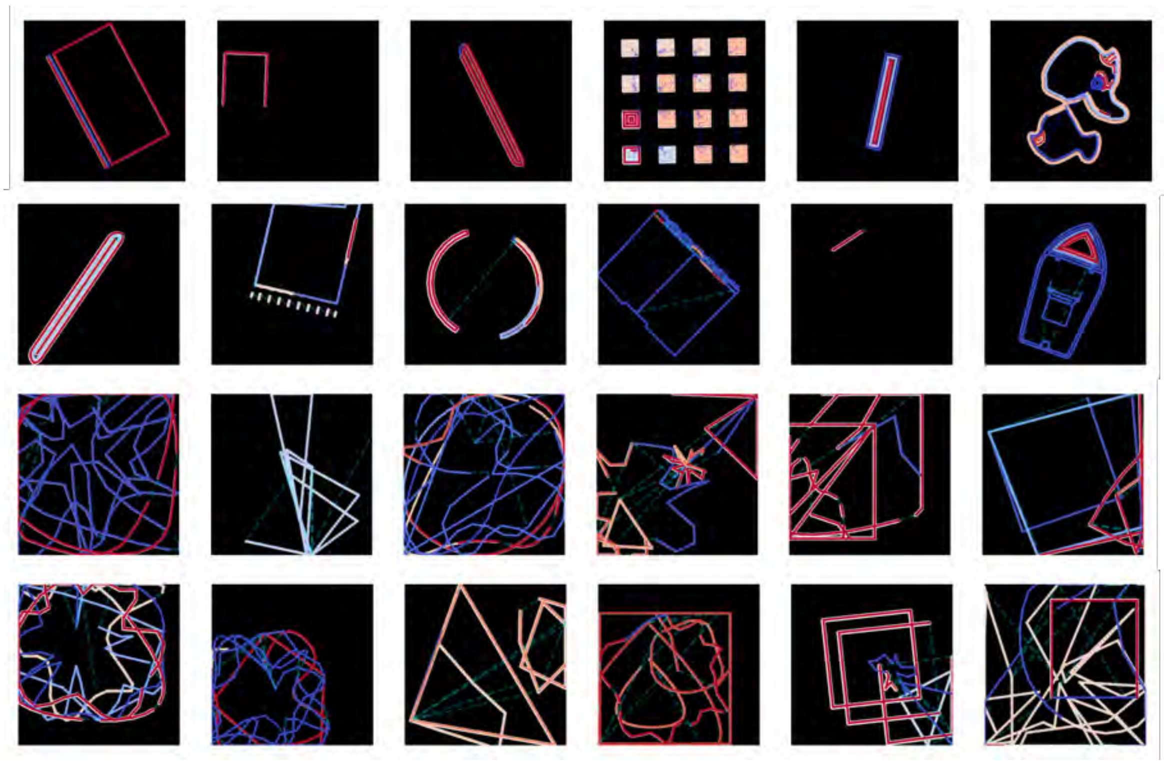

Figure 3.9: **Synthetic Data.** The top half displays rendered images from the provided graph data set. The bottom half are the random graphs. The visualisation plots a colour coded graph over the top of the render. N.B. the render perfectly aligns with the graph but has randomly varying widths. Dashed lines indicate inactive edges.

To generate synthetic data, the existing space-time graphs were gathered alongside randomly generated graphs. Randomly generated graphs were created in several different ways. For each random graph, a number of sub-graphs were randomly sampled (up to 10). Each sub-graph were composed of either simple geometries (rectangles, triangles, etc) or were generated from sampled points along random Bezier curves. Furthermore, each node in the graph had a random probability to be active or inactive, and finally a random line-width and line-width variability were generated. On average line-width would vary five times within a single graph. To generate the corresponding image, these graphs were rendered using matplotlib (Hunter [2007]). These renders do not exhibit errors that are prevalent in the real-data, however, the synthetic data still provides a surrogate graph-to-image task, albeit deterministic and classically solvable.

The training pipeline expects to receive graph and image pairs, however, pre-computing images for each graph is expensive and restrictive. A better approach would generate renders in real time. This could be achieved at run-time by passing a graph to a Matplotlib function that exports the canvas as an array. Matplotlib has been highly optimised for such plotting tasks. Furthermore, by wrapping this operation in a Pytorch dataset (Paszke et al. [2019]) this allowed parallelisation of the task. Since renders were generated on-the-fly more extreme data augmentation such as scaling, width variation and shearing could be also be used. See Figure 3.9 for examples of this synthetic data.

3.2 Conditioning Images with Graphs

This section discusses possible strategies to condition a diffusion model with a graph. For notational clarity in some sections graphs are highlighted with magenta, images with blue. A common strategy in text-to-image models is to encode a text prompt into text embeddings using an existing language model, then to give the diffusion model a summarised vector representation of this prompt. This may be appropriate for global conditioning of an image but one can imagine that such a summarised embedding would prove to be an information bottleneck in fine-grained spatial tasks such as is the focus here. For example, it would be difficult to effectively represent a graph containing 10,000 nodes and a graph containing a single node in the same fixed length vector representation. Even if the target representation is 2D (e.g. a latent image), the model must handle variable length inputs and explicit invariances such as rotation, translation, and pair-wise permutations. Furthermore, once we extend to multiple layers it must also handle the task of projecting multiple layers in 3D to the 2D image and the complicated physical interactions this entails. Super-resolution models provide a better task comparison: by conditioning a diffusion model on a low-resolution image, we effectively restrict possible generations to adhere to the underlying spatial information. In this work we do not have a low-resolution image to provide, however, we can provide similar conditioning signals such as a line mask of the ideal render. Essentially, this formulates the task as the residual problem - the model only needs to learn the difference (e.g. the errors). However, a simple line mask ignores the graph and process parameter information e.g. the flow rate. Therefore, several more considered approaches were explored, two of which are described in detail. The first and simplest of the two, is referred to as a *learnable render using line lookup*. The second is referred to as the *dual-graph GNN*.

3.2.1 Learnable Render using Line Lookup

As discussed, the simplest approach to graph-to-image would be to provide the model a line mask. Line masks can be rendered using the graph's coordinates in a similar way to how synthetic data is generated. However, one problem with this procedure is that it doesn't use the full suite of available information, for example we know there is a relationship between flow rate and resulting line which is unknown at training time. In an approach inspired by classical rendering and the Neural Texture work of Thies et al. [2019], we can lookup a single line mask that connects two nodes, and fill them in with an appropriate value. This value or vector of values can be directly learned from the local nodes process parameters. In essence, for a graph with N nodes this approach would produce $N-1$ latent line mask images. These line mask images can be combined via summation to produce a single D dimensional latent image representation of the graph and its process parameters. For an illustrative overview of this approach see Figure 3.10. Formally, this learnable render module can be described with the following equation:

$$R = \sum_{i=1}^{N-1} (\phi(\mathbf{n}_i) \odot M_i) \tag{3.1}$$

Where output render R is equal to the sum over all latent line images. Where ϕ is a learnable transformation of node n_i multiplied element-wise ($\odot$) by line mask M_i. Where line mask M_i is 1 for all pixels that intersect with the linear path between node n_i and n_{i+1}, and is 0 everywhere else.

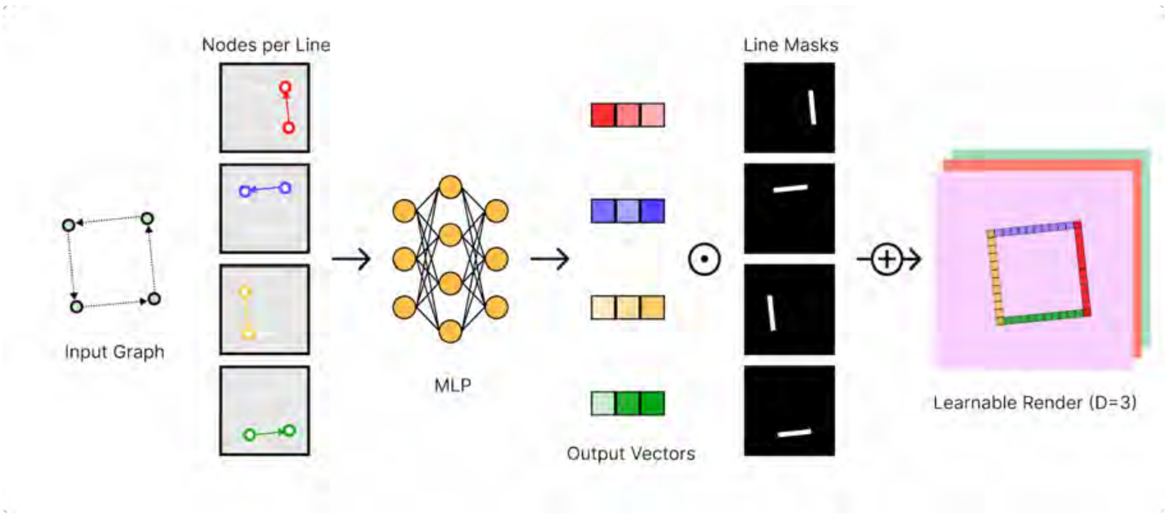

Figure 3.10: **Learnable Render Overview** From left to right. Given a space-time graph, the N-1 lines are extracted as pairs of sequential nodes, and line masks are generated for each pair. Each node is passed through a MLP which outputs a vector representing the print parameters for the line, this is multiplied by its corresponding line mask. A 2D latent image is calculated by summing the $N-1$ line masks into a single D dimensional latent image. In this diagram $D = 3$.

Algorithm 2 Learnable Render Module for Space-Time Graph

```
Require: Heterogeneous Graph Data: het_graph
 1: print_nodes ← get_node(het_graph, print_nodes)
 2: print_textures ← MLP(print_nodes)
 3: output_render ← zeros(size)
 4: for $i \in \{1, \ldots, N-1\}$ do
 5:     start_node ← print_nodes[i].xy
 6:     end_node ← print_nodes[i+1].xy
 7:     mask_idx ← draw_line(start_node, end_node)
 8:     output_render.index_add(mask_idx, print_textures[i])
 9: end for
10: return output_render
```

Calculating line masks between two points is a highly optimised procedure, and this did not prove to be a bottleneck during inference, even for large graphs. This unit was implemented into the model at various resolutions, which gives the model to generate effects beyond the the fine-grained line at full-resolution. Anti-aliased lines were also investigated. The results are presented in section 4. For an algorithmic overview see algorithm 2, note in practice that masks can be pre-computed with vectorised lookup procedures for entire graph batches (avoiding loops), this allows highly efficient and learnable rendering.

3.2.2 Dual-Graph GNN

Although the learnable render described in the previous section solves basic spatial conditioning, it is likely insufficient for accurate prediction of complex physical defects. This is because many complex defects require knowledge of multiple nodes and the relationships between them, and is indeed why inputs are formulated as graphs as opposed to sequences. One option to incorporate the graph would be to replace the MLP in the learned render with a GNN, and then extract node embeddings. However, this approach suffers from the fact that that every value along a latent line image is the same i.e. it makes the assumption that the relationship between pixels and points along the line stays uniform. Although it is possible for the diffusion model to sample from the underlying distribution, a more flexible approach may allow the model to learn complex patterns of error. To support a more flexible conditioning format the concept of a *dual-graph* was introduced.

A dual-graph representation considers the graph-to-image task as an operation that exists on two graphs. The first graph is the familiar space-time graph that consists of *print nodes* and is connected with *temporal print edges*. The second graph represents the target image or its latent representation. This graph consists of *image nodes*, where each node can represent a single pixel or a patch of pixels. These two graphs can be inter-connected in various ways, for example an obvious choice is to connect every image node that falls along a print line to the previous node, essentially filling the same role of the learnable render but with more expressivity. Refer to Figure 3.11 for an illustrative breakdown of the dual-graph concept and different edge types.

The benefits of operating on a dual-graph representation are numerous and more dynamic than they may first appear. Some of the benefits include: natural handling of variable length graphs, exploiting invariances, equivariances and inductive biases, and an efficient way to switch between image and graph representations via node updating. Firstly, this representation efficiently handles variable length graphs without padding. Secondly, the process parameters of the print graph are invariant to rotation and translation, in that the flow rate of a rotated print graph remains unchanged. Yet, the connection of the print graph to the image graph is equivariant (ignoring discretisation

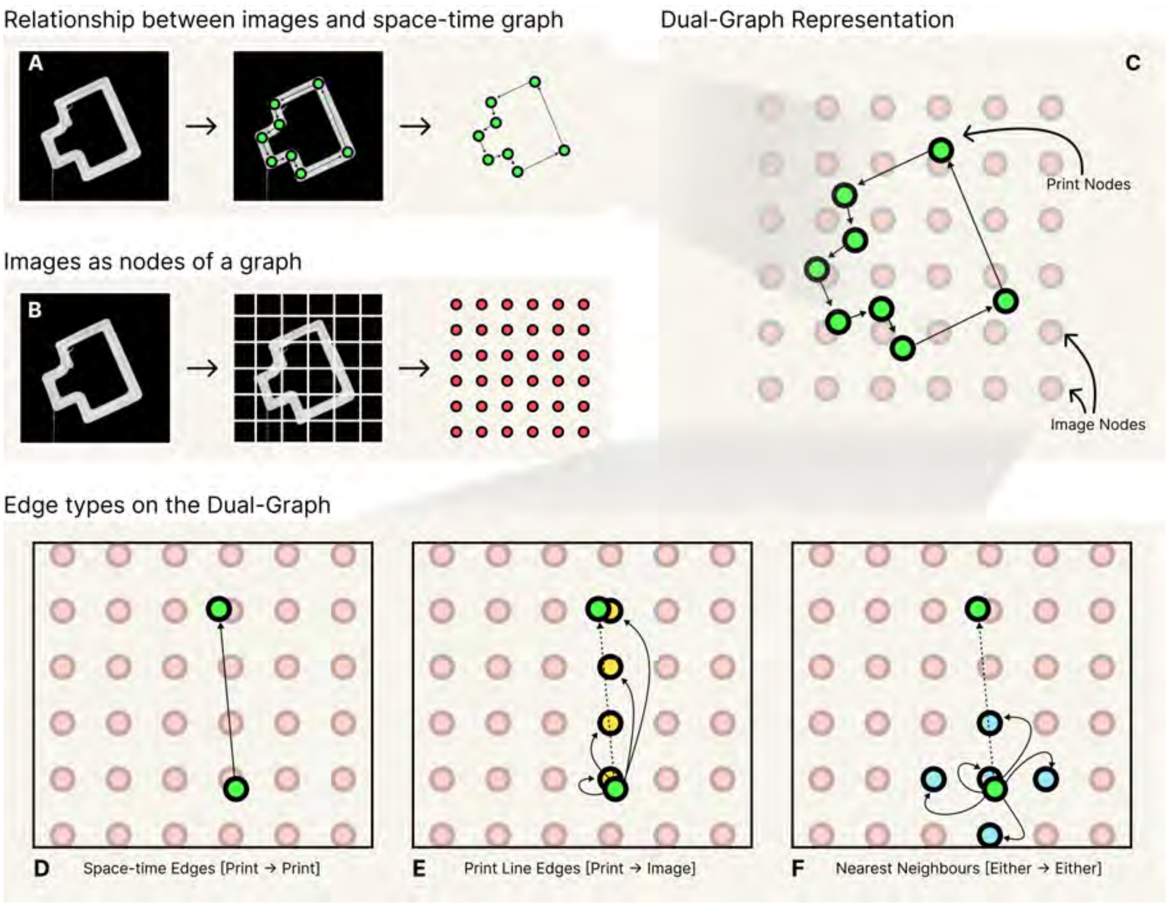
Relationship between images and space-time graph
A
Images as nodes of a graph
B
Dual-Graph Representation
C
Print Nodes
Image Nodes
Edge types on the Dual-Graph
D Space-time Edges [Print → Print]
E Print Line Edges [Print → Image]
F Nearest Neighbours [Either → Either]

Figure 3.11: **Dual Graph Representation. A)** For every layer image there exists the corresponding space-time graph that was used to print it. **B)** Every layer image is made up of pixels, each pixel can be thought of as a node in a grid graph. Likewise in CNN or ViT architectures, once images are fed into a model one pixel can also represent a local patch of pixels. **C)** The dual-graph representation imagines overlaying the print graph and image graph on top of one another. The image lives in a discretised coordinate space, whereas the space-time graph lives in a continuous space. **D)** The print graph can be connected to itself with temporal print edges. **E)** The print graph can connect to the image graph by connecting to nodes along the print line. Edges can also contain additional features (e.g. distances). **F)** Other possible edge types include nearest neighbours which can be between either print nodes, image nodes or both.

error). This has significant effects on the output. Consider an example graph that exhibited adhesion defects, by rotating or translating the graph we can expect the output to display equivariance, but potential defects can be processed invariantly on the print graph. Finally, this representation allows for an efficient way to switch between variable length graphs and fixed resolution images, in turn this opens the door to unique processing by switching between the representations. The dual graph can take as input a set of 'current image features' i.e. the current latent representation of the image passed from the U-Net model. These can be then used to update the image nodes on the graph before the the dual-graph is passed through a heterogeneous GNN. Furthermore, it flexibly allows image nodes on the graph to be converted back into latent image patches, passed through any transformation (such as a ResNet block) and then converted back into nodes on the graph for further processing.

Formally we can present operations on a dual-graph GNN with the following equations. The first operation allows current feature maps to replace image nodes in the dual-graph, for incoming image features $\mathbf{x}$ of shape $[B, C, H, W]$:

$$\mathbf{x}^{patch} = P(\mathbf{x}, [B \cdot H \cdot W, C]) \tag{3.2}$$

$$\mathbf{x}^{node} = U(\mathbf{x}^{patch}) \tag{3.3}$$

Where U is a vertex updating function, and function P permutes and reshapes image features into the N image nodes. The vertex update function U can be chosen as the identity, a learned projection or a more complex transformation.

Once image nodes are updated, we can perform any variety of heterogeneous message passing using the dual graph GNN:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v, e_{uv}, t_u, t_v)\right) \tag{3.4}$$

Where the message passing transformations depend on both the edge features e_{uv} and the types of nodes t_u, t_v. Readers may refer to appendix A.2. for specific materialisations of Eq 3.4. Furthermore, we can easily switch back and forth between image and dual-graph representations to apply intermediate image transformations on the image nodes, for example a convolutional layer. For all image nodes $\{\mathbf{h}_u^{\text{image}}\}$:

$$\{\hat{\mathbf{h}}_u^{\text{image}}\} = U\left(P\left[\psi(P^{-1}(\hat{U}(\{\mathbf{h}^{\text{image}}{}_u\})))\right)\right] \tag{3.5}$$

Where P^{-1} transforms the set of nodes nodes back into images, i.e. the inverse operation of P in eq 3.3. $\hat{U}$ is again a node wise transformation such as a projection that facilitates transforming nodes back into image patches. Where ψ can be any image-wise learnable transformation, such as a convolutional layer or block. It is trivial to switch back and forth between the image and dual-graph representation which opens the door to new types of models. Moreover, common deep learning techniques such as skip connections are easy to integrate. For clarity we present a simple example formulation of the dual graph block, using a simple convolutional image layer and simple graph convolutional layer with mean aggregation. For input image features $\mathbf{x}$ and conditioning nodes $\mathbf{c}$:

$$\hat{\mathbf{x}}_i = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \left[\mathbb{1}_{\{j \in \text{Image}\}} \cdot (e_{ji} \cdot U(P(\mathbf{x}_j))) + \mathbb{1}_{\{j \in \text{Conditioning}\}} \cdot (e_{ji} \cdot \hat{\mathbf{c}}_j) \right] \tag{3.6}$$

$$\tilde{\mathbf{x}}_i = \sum_{j \in \mathcal{R}(i)} w_{ji} \cdot \left(P^{-1}(\hat{U}(\hat{\mathbf{x}}_j)) + b_i \right) \tag{3.7}$$

Equation 3.6 applies the GNN to the local neighborhood $\mathcal{N}$, incorporating both image and conditioning nodes. In 3.7, the image nodes are transformed back into an image feature map, and an image convolution is applied to the local receptive field $\mathcal{R}$. We can interpret this formulation not only as a GNN that operates on a graph with two types of nodes, but also as a framework that operates on two low-dimensional structures -specifically, how $\mathcal{N}$ operates on the GNN and $\mathcal{R}$ operates on the image. Activation functions are omitted for brevity. We also present a more general algorithmic overview in 3 below.

The metadata for dual graphs can be computed on-the-fly from augmented data e.g. print line edge connections. Additionally, new features can be attributed to print graphs such as distances between other nodes or positional encoding information. See Figure 3.12 for an example dual-graph with real data.

Ultimately this novel dual-graph representation allows for highly expressive spatial conditioning, and allows for unique operations as it switches back and forth between image and dual-graph. Alongside its flexibility, this form of dual-graph conditioning is parameter-efficient due to weight sharing, message-passing and aggregation, allowing highly specific spatial conditioning with only minimal parameter increases. See Figure 3.13 for an overview of the dual-graph GNN block. Note, this block is flexible in its design and part of the project was to explore best practices.

Prior work has proposed image processing using graphs (Han et al. [2022]), processing images to generate graphs (Xu et al. [2017], Yang et al. [2018]), and even

Algorithm 3 Dual-Graph Block for Image and Heterogeneous Graph

```
Require: Image and Heterogeneous Graph Data: image, het_graph
 1: for $i \in \{1, \ldots, L\}$ do
 2:     image_nodes ← reshape_to_patches(image)
 3:     image_nodes ← projection_layer(image_nodes)
 4:     het_graph ← update_nodes(het_graph, image_nodes)
 5:     for $g \in \{1, \ldots, G\}$ do
 6:         hidden ← GNN(het_graph)
 7:     end for
 8:     hidden_image_nodes ← extract_image_nodes(hidden)
 9:     image ← reshape_to_image(hidden_image_nodes)
10:     for $c \in \{1, \ldots, C\}$ do
11:         image ← activation(conv(norm(image))
12:     end for
13: end for
14: return image
```

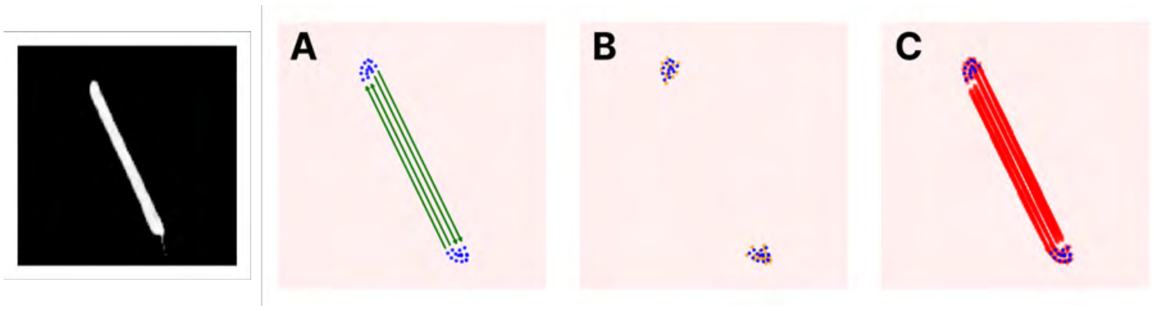

Figure 3.12: **Example Dual Graph.** Layer image and corresponding dual-graph representation, print nodes shown in blue, dense image nodes in pink. **A)** *Temporal print edges* in green. **B)** *Nearest Neighbour edges* in orange. **C)** *Print line edges* in red

diffusion models conditioned on abstract scene-graphs (Mishra and Subramanyam [2024], Farshad et al. [2023], Yang et al. [2022]). However, to my knowledge no work has presented a representation similar to a dual-graph GNN as it introduces a dynamic back-and-forth approach between graph and image, and allows fine-grained spatial conditioning. We postulate this to be pivotal for the proposed task because the image contains important information for use in updating the graph representation.

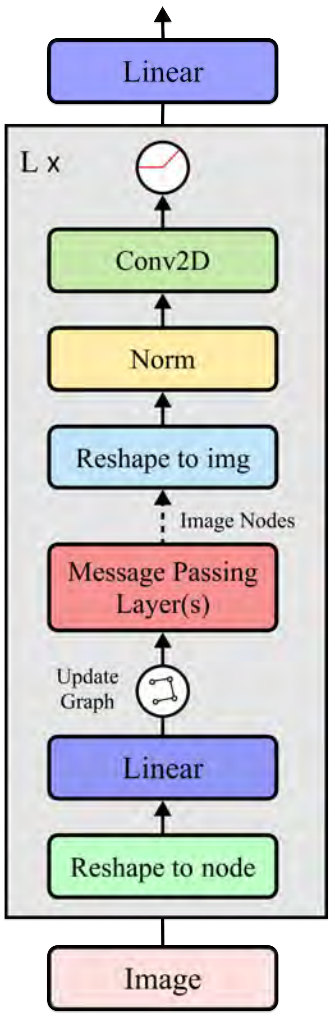

Figure 3.13: **Dual Graph Architecture.** The dual-graph block expects an input image per heterogeneous graph datapoint. The image is reshaped into N image nodes, and passed through a learned projection. The image nodes in the graph are replaced with these features. The dual-graph is then passed through a GNN which returns a new set of node embeddings. Image nodes are extracted from these new embeddings, reshaped back to images and passed through a convolutional block. This process repeats for L blocks before being returned. Options such as residuals, complex convolutional or transformer blocks are all feasible.

3.3 Graph-to-Image Diffusion Model

This section presents a unified system using the graph-to-image dataset and the proposed conditioning strategies. The Elucidated Diffusion framework (Karras et al. [2022], as outlined in section 2.2) was employed alongside the U-Net backbone from Saharia et al. [2022]. Elucidated diffusion models introduce a substantial volume of notation, to avoid distracting from the main substance of the report, this notation and accompanied treatment is presented in appendix section A.1. The proposed graph-to-image conditioning is integrated directly into the U-net encoder. This provides early signal to the model, yet allows the model to adjust features in the decoder if required.

For sample generation (predicting a layer image from a given graph), random Gaussian noise is drawn and passed to the model D_θ conditioned on graph $\mathbf{g}$, producing samples from $p(\mathbf{x}|D_\theta(\mathbf{g}))$. We adopt the 2nd order stochastic sampler from Karras et al. [2022] and stick to their default linear schedule and ImageNet hyper-parameters for sampling and training. We modify their algorithm 2 with our conditioning and present it for reader convenience:

Algorithm 4 Stochastic Sampler with Conditioning Graph
(Adapted from Karras et al. [2022] algorithm 2)

```
Require: Trained model and conditioning hetero graph: $D_\theta$, $\mathbf{g}$
 1: Sample $\mathbf{x_0} \sim \mathcal{N}(0, t_0^2\mathbf{I})$
 2: for $i \in \{0, \dots, N-1\}$ do
 3:     Sample $\epsilon_i \sim \mathcal{N}(0, S_{\text{noise}}^2\mathbf{I})$
 4:     $\hat{t}_i \leftarrow t_i + \gamma_i t_i$        ▷ $\gamma_i = \begin{cases} \min\left(\frac{S_{\text{churn}}}{N}, \sqrt{2}-1\right) & \text{if t}_i \in [S_{\min}, S_{\max}] \\ 0 & \text{otherwise} \end{cases}$
 5:     $\hat{x}_i \leftarrow x_i + \sqrt{\hat{t}_i^{\,2} - t_i^2}\epsilon_i$
 6:     $d_i \leftarrow \left(\hat{x}_i - D_\theta(\hat{x}_i|\mathbf{g}; \hat{t}_i)\right)/\hat{t}_i$
 7:     $x_{i+1} \leftarrow \hat{x}_i + (t_{i+1} - t_i)d_i$
 8:     if $t_{i+1} \neq 0$ then
 9:         $d'_i \leftarrow (x_{i+1} - D_\theta(x_{i+1}|\mathbf{g}; t_{i+1}))/t_{i+1}$
10:         $x_{i+1} \leftarrow \hat{x}_i + (t_{i+1} - \hat{t}_i)\left(\frac{1}{2}d_i + \frac{1}{2}d'_i\right)$
11:     end if
12: end for
13: return $x_N$
```

We also follow the training procedure from Karras et al. [2022], including pre-conditioning and loss re-weighting. Following their advice and best practice, we train a separate model F_θ in which D_θ can be derived which allows input and output magnitudes to remain stable. Ultimately, this means we predict both noise and signal, adapting Karras et al. [2022] Eq. 7 with conditioning graph $\mathbf{g}$ this yields:

$$D_\theta(x;\sigma) = c_{\text{skip}}(\sigma)x + c_{\text{out}}(\sigma)F_\theta(c_{\text{in}}(\sigma)x|\mathbf{g}; c_{\text{noise}}(\sigma)) \tag{3.8}$$

We also adapt Karras et al. [2022] Eq. 8:

$$\mathbb{E}_{\sigma,y,n}\left[\lambda(\sigma)c_{\text{out}}(\sigma)^2\left\|F_\theta\left(c_{\text{in}}(\sigma)\cdot(y+n)|g;c_{\text{noise}}(\sigma)\right)-\frac{1}{c_{\text{out}}(\sigma)}\left(y-c_{\text{skip}}(\sigma)\cdot(y+n)\right)\right\|_2^2\right] \tag{3.9}$$

Below we present the training algorithm for our system given a random training point:

Algorithm 5 Elucidated Diffusion Training with Conditioning Graph

```
Require: Model, image and conditioning hetero graph: $F_\theta$, images, g
1: Sample $\sigma \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}})$ from noise distribution
2: Sample $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
3: noised_images ← images $+\sigma\cdot\epsilon$
4: denoised_images ← preconditioned_model($F_\theta$, noised_images, g, $\sigma$)    ▷ see 3.6
5: loss ← $\frac{1}{H\cdot W}\sum_{i=1}^{H\cdot W}$(denoised_images − images)$^2$
6: weighted_loss ← loss $\cdot(\sigma^2+\sigma_{\texttt{data}}^2)\cdot(\sigma\cdot\sigma_{\texttt{data}})^{-2}$
7: return weighted_loss
```

For readers who are unfamiliar with the work of Karras et al. [2022] and Song et al. [2021], the notation and procedure here may seem opaque. To succinctly summarise, we train a model on noisy images at a given noise level σ, but we also allow the model to process the corresponding space-time graph. We ultimately get the model to predict the denoised image. Once trained on many noisy images, we can predict new samples. We do this by starting with pure noise and a corresponding graph and iteratively denoise it using the last estimate as the prediction for the next time step. We follow Algorithm 4 to do this; despite not being a general purpose SDE solver (Karras et al. [2022]) it has strong empirical results and is faster than other methods.

Finally, to conclude the method section we present an overview of the entire model with integrated conditioning in Figure 3.14 at the end of the chapter.

3.4 Validation Procedures

The validation routes for this project are not straightforward. The most commonly used validation metric: *Fréchet Inception Distance* (FID) is not well-suited for reconstruction tasks. This is because FID relies on pre-trained network statistics, which are out of distribution for our FDM data. Moreover, FID measures visual quality which does not align with our specific goal of reconstruction. For similar reconstruction tasks like super-resolution, metrics such as Peak Signal-to-Noise Ratio (PSNR) or Structural Similarity Index (SSIM) are typically used, though they often do not correlate well with human preferences Saharia et al. [2021]. However, the goal of the task is to faithfully reconstruct the output layer to predict possible errors. Therefore, simple MSE measured against the reconstruction of test images (given input graphs) makes the most suitable quantitative validation metric. Typically, PSNR (a normalised and reweighted MSE) measured in dB is preferred as it normalises the metric with respect to pixel values offering easier comparisons. Note: higher PSNR denotes a higher quality reconstruction, typical lossy image compression ranges from 30-50 dB.

$$\text{MSE} = \frac{1}{n^2} \sum_{i=0}^{n} \sum_{j=0}^{n} \left(y_{(i,j)} - \hat{y}_{(i,j)} \right)^2 \tag{3.10}$$

$$\text{PSNR} = 10 \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right) \tag{3.11}$$

This metric is not ideal as whilst the model may predict the correct type of error (e.g. stringing or adhesion failure), the pixel-precise prediction of where this failure occurs may not be accurate. In practice these types of predictions are still useful for operators despite being penalised by this metric, therefore we resort to careful qualitative analysis of model predictions alongside numericaol analysis. Human evaluation metrics are crucial as they provide insights that purely numerical results may overlook, particularly in assessing the practical utility of model predictions. The downside to numerical analysis is compounded by the fact that diffusion is inherently an stochastic process, and so results will slightly different between samplings runs. Finally, to validate the practical effectiveness of our model we use it to predict optimal process parameters for a given geometry on unseen test samples, this is presented in 4.5.2. The available data was split into training and validation splits (90/10), results are presented on the validation split.

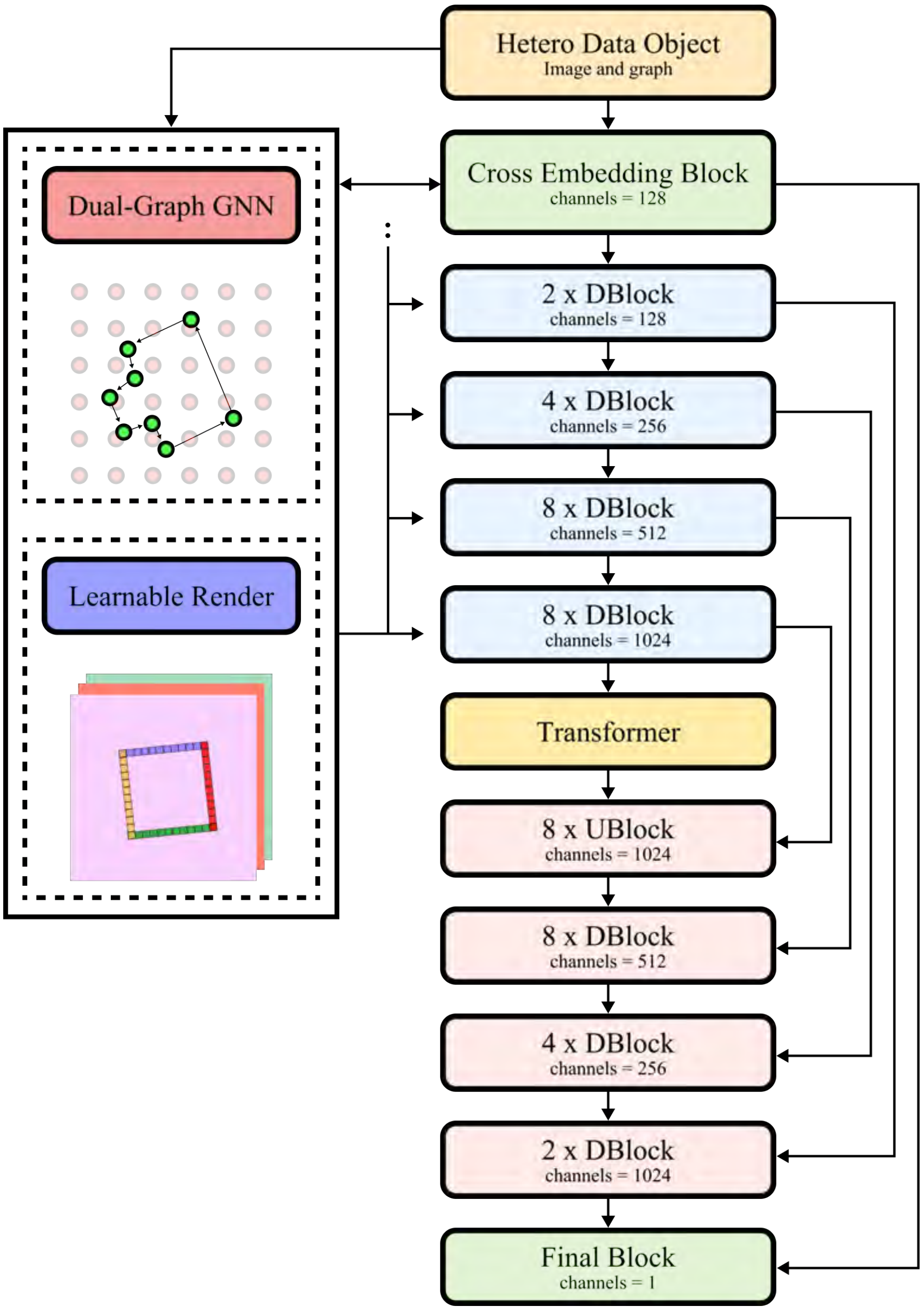

Figure 3.14: **Graph-to-Image Diffusion Model $F_\theta(\mathbf{g})$.** The baseline U-Net model augmented with graph-to-image conditional units. Note the dual-graph GNN uses the current image features and heterogeneous graph data, rather than the learnable render which only uses graph information to map to latent image space. Both units can be plugged into any resolution via feature interpolation. Cross embedding layers consist of multiple convolutional kernel sizes. Down and up blocks consist of ResNet blocks with optional attention and diffusion time conditioning.

Chapter 4

Results

This section covers model training, experimental setup, key ablations and results. It then details training on synthetic and real data, followed by model improvements and further ablations. Finally, reconstructions from the best model are qualitatively examined, along with an real-world case study into optimising process parameters for a given target render.

4.1 Experimental Setup

The experimental setup focused on training a customised U-Net on a graph-to-image diffusion task at 128x128 resolution. Hyper-parameters were directly derived from the *Efficient U-Net* 256x256 super resolution model from Saharia et al. [2022] - see appendix A.3 for details. This establishes a competitive baseline to evaluate the addition of the new computational units. The Adam optimiser (Kingma and Ba [2017]) with a fixed learning rate of 0.0001 was used. The baseline model has 693.6 million parameters.

We incorporate two additional conditioning units: a learnable render unit and a dual-graph GNN. The learnable render unit was developed from scratch to efficiently handle graph-batched data. The learnable render is designed to match the current resolution dimensions; for example, at the lowest resolution, it produces a 1024-dimensional 16x16 render. Additionally, an initial render with 8 dimensions at full resolution was included. The baseline learnable renders add 1.49 million parameters. The dual-graph GNN interface was written in PyTorch. We evaluated existing models from PyTorch Geometric for the GNN layers and selected the Heterogeneous Graph Transformer (HGT) from Hu et al. [2020] as the baseline model; different architectures are explored in a later section. This unit includes custom node updating functionality, intermediate image transformations, representation switching, and additional conditioning (e.g., diffusion time). By default, we set the number of Dual-Graph GNN layers to 2, with 2 blocks, resulting in a total of 4 GNN layers and 2 intermediate image transformations. We use all three edge types described in Section 3.3, including explicit reverse-type edges (i.e., undirected heterogeneous). The baseline dual-graph GNN adds 1.32 million parameters. Sample generation involves 32 diffusion steps.

4.2 Synthetic Data

This section presents model training using synthetic data, which is abundant and classically solvable so high performance is expected. The dataset contains 35,000 base graphs, but is rendered at runtime with varying widths and augmentations. This approach allowed us to evaluate proposed conditioning strategies before real data had been collected. Firstly, a baseline was established using only the learnable applied at full resolution. This was subsequently improved by applying it at multi-resolution and with anti-aliased line masks. The dual-graph GNN was then added, and subsequently enhanced with intermediate convolutional layers. As hypothesised, using both conditional units provided the best results, as shown in Table 4.1.

Model Name	**Steps**	**PSNR (dB)**	**Improvement (%)**
Learnable Render (LR)	5k	23.68	-
LR Multi-Res	5k	24.56	3.71
LR Multi-Res Anti-Aliased	5k	25.08	5.91
Dual-Graph GNN without Conv	5k	24.62	3.97
Dual-Graph GNN	5k	25.17	6.32
Dual-Graph GNN and LR	5k	26.75	**13.00**
LR Multi-Resolution Anti-Aliased	50k	25.24 $\pm$0.24	6.6
Dual-Graph GNN	50k	29.36 $\pm$0.28	24.00
Dual-Graph GNN and LR	50k	30.21 $\pm$0.27	**27.61**

Table 4.1: **Validation PSNR for Trained Models**. Note percentages are calculated in PSNR which is a logarithmic measure. Values rounded to 2.d.p and standard deviations of the last 5% of training steps are included.

To validate the effects of conditioning, we monitored the loss and reconstruction curves over 50,000 update steps (24 hours of training on a single GPU using the Cambridge HPC), as shown in Figure 4.1 and Figure 4.2. Interestingly, the learnable render reached a performance plateau relatively early, while the GNN continues to improve, suggesting it provides a more expressive representation. There is a larger improvement to reconstruction compared to training loss, indicating the GNN results in more accurately conditioned generations. The best model achieves a PSNR of 30.21, which is comparable to lossy image compression, demonstrating very strong performance on synthetic data. While the models have not yet converged, they generate high-quality samples. Figure 4.3 shows the evolution of validation samples over training, and Figure 4.4 shows validation samples from the best model. This approach successfully conditions images with different graph properties (e.g. line width) to a high degree of accuracy.

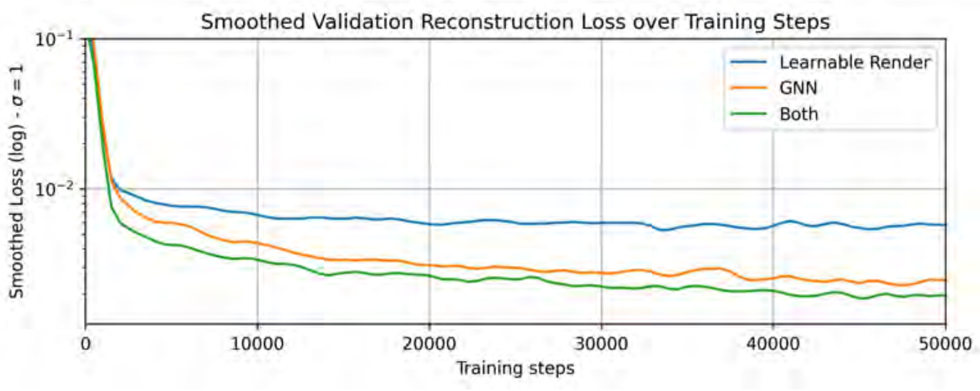

Figure 4.1: **Training on Synthetic Data - Validation Reconstruction.** Validation reconstruction MSE over 50,000 steps. Three runs are shown: one using the GNN, one the learnable render, and one combining both approaches. Even on synthetic data - which lacks complex physical dynamics - the GNN emerges as the most effective conditioning method.

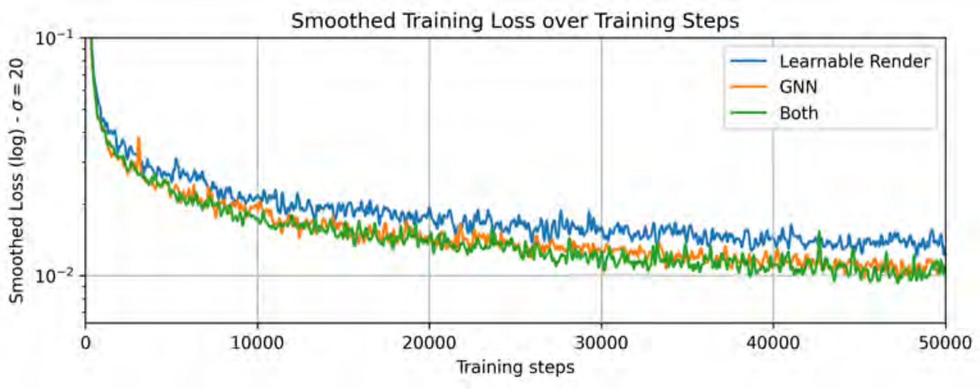

Figure 4.2: **Training on Synthetic Data - Training loss.** Training loss over 50,000 steps, matching Figure 4.1. The loss is very noisy, and is therefore highly smoothed.

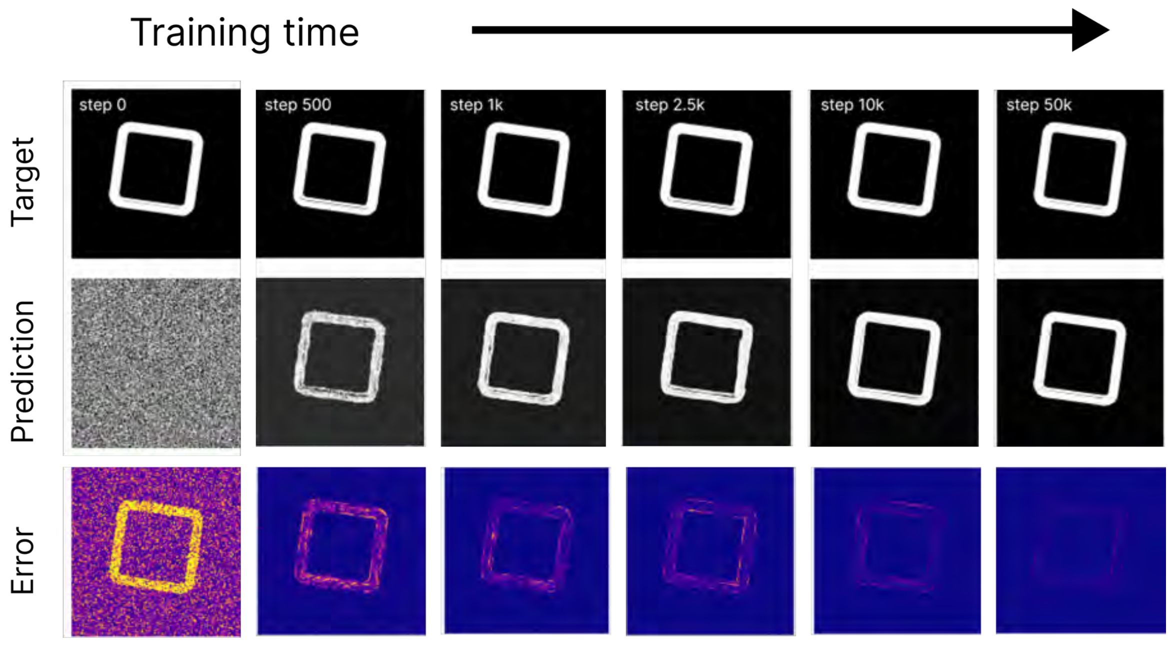

Figure 4.3: **Evolution of Validation Reconstruction over training steps.** These images show samples from same conditioning graph over model training with steps: $0, 500, 1000, 2500, 10000, 50000$. The top row is same target image, middle row is the prediction and the bottom row is the difference between the two e.g. the error.

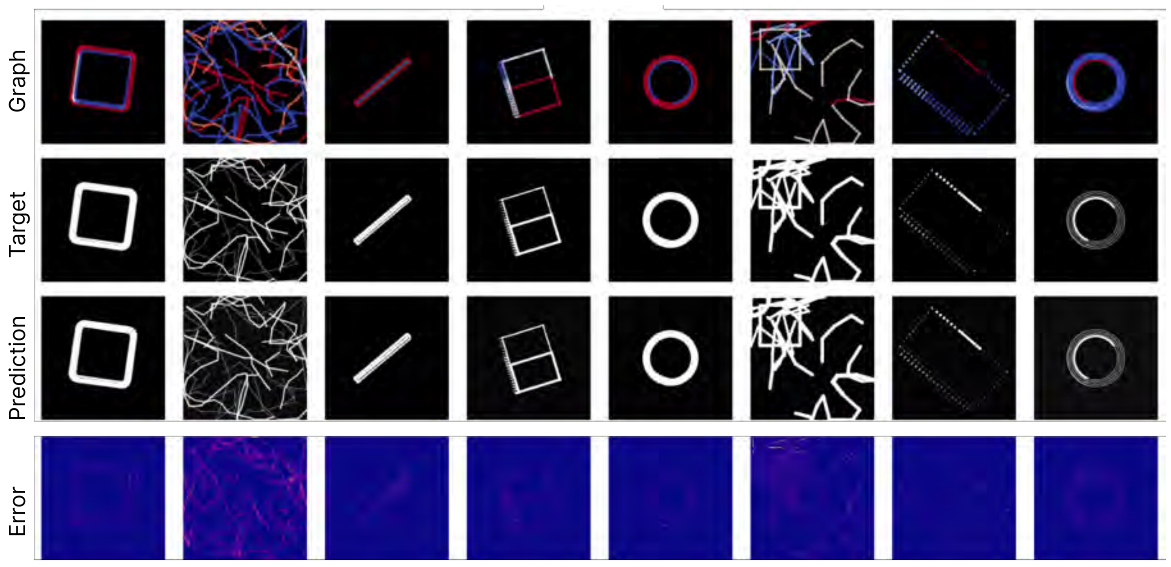

Figure 4.4: **Validation Reconstruction Samples.** Samples generated at 50k steps from the best model. The top row displays the graph color coded with flow rate. The middle rows show the target image and the model reconstruction. The bottom row demonstrates the error between the two.

4.3 Evaluating Real-World Data: Initial Findings

This section explores preliminary results on real data, beginning with the initial dataset of 1,500 samples, with a subsequent section utilising the full 2,500 samples (due to data collection timeline). The primary objective is to assess whether insights gained from synthetic data experiments extend to real data—a significantly more complex challenge.

Model Name	**Steps**	**PSNR (dB)**	**Improvement (%)**
Learnable Render (LR)	50k	16.53 ±0.41	-
Dual-Graph GNN	50k	17.61 ±0.42	6.51
Dual-Graph GNN and LR	50k	17.51 ±0.32	5.95
- *no pre-training*	50k	17.58 ±0.11	6.34

Table 4.2: Validation Reconstruction (PSNR) for Models trained for 50K steps on real data. To mitigate noise PSNR values are calculated by taking the mean and standard deviation of the last 5% of training steps. Values are reported to 2.d.p.

To this end, four experiments were trained for 50,000 update steps using the best-performing settings from the synthetic data experiments, such as multi-resolution and anti-aliasing techniques (see table 4.1). The loss curves are presented in Figure 4.5 and Figure 4.6; surprisingly, pre-trained models did not exhibit performance compared to those trained from scratch. Yet, it did initially speed up convergence. This is to be expected as pre-trained models have sensibly calibrated outputs. This suggests that the characteristics of the real data do not provide additional information that would be advantageous for the real task. This phenomenon could be attributed to the strong inductive biases of the conditional units, as they encode the majority of information provided from the synthetic data (i.e. line placement).

The learnable render showed little to no benefit when combined with the GNN, which is in keeping with the findings from synthetic data. Interestingly, the training loss of GNN models exhibit only a modest decrease of $\sim 10\%$, yet, more substantive improvements of up to 30% in reconstruction MSE (N.B. PSNR is log scale). This suggests that the GNN enhances accurate conditioning rather than merely improving visual quality. The PSNR values were calculated with respect to validation reconstruction and are displayed in the table 4.2. To mitigate measurement noise the average values over the last 5% of training steps is reported. Measurements on real data will be noisier, particularly as the validation set is small.

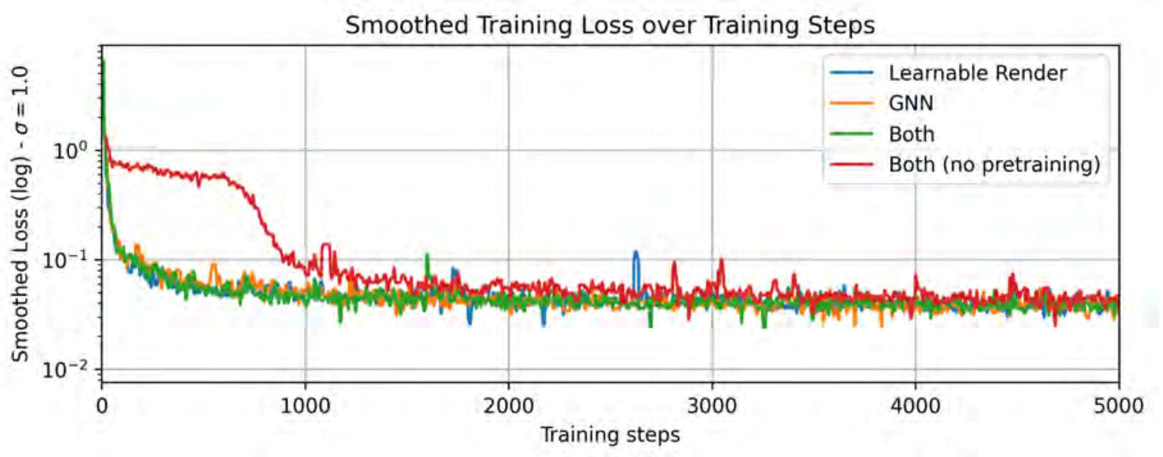

Figure 4.5: **Training Loss Curves on Real Data.** The graph presents smoothed validation reconstruction MSE and smoothed training loss over 50k update steps for the four initial training runs. Results smoothed with a Gaussian kernel $\sigma = 1$.

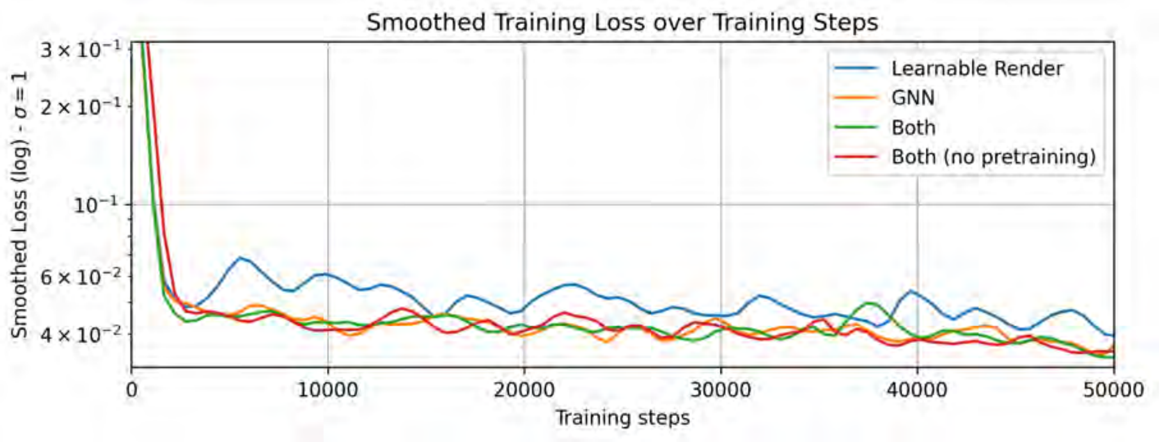

Figure 4.6: **Validation Reconstruction Curves on Real Data.** The graph presents smoothed validation reconstruction MSE over 50k update steps for four initial training runs. Results smoothed with a Gaussian kernel $\sigma = 1$.

We anticipated the model's performance to degrade when applied to real data, due to factors such as non-determinism, images misalignment, and significantly smaller dataset size. Additionally, one could expect the model to take longer to converge on this more complex task. Nonetheless, it is encouraging that the insights gained from synthetic data were transferable to the real domain. Given the limitations in dataset size and the presence of measurement noise, longer training runs should validate these findings. The GNN model consistently demonstrated its advantages over the learnable renderer. However, due to computational constraints, we are unable to fully confirm this advantage through extended validation.

The next sections focus on improving the model before presenting a final version trained for longer and on a larger dataset. To avoid repeating qualitative analysis, we reserve this for section 4.5.

4.4 Further Improvements & Ablation Studies

This section explores possible optimisations of the dual-graph GNN, as it has been established as the most effective conditioning unit. This includes: testing different GNN architectures, a study into the effect of GNN layer depth, conditioning the dual-graph GNN on diffusion time, additional learned spatial embeddings and finally an ablation into the effect of image/graph alignment. Each of these experiments is discussed in the following subsections.

Additional Spatial & Temporal Conditioning

We explore the use of additional spatial and temporal conditioning, through the use of learned sinusoidal embeddings (similar to ViT positional embeddings Dosovitskiy et al. [2021]). The intuition here is that print nodes contain sub-pixel positional information which may be useful for the model, and allow the model to learn more complex spatial relationships. For all graph nodes we calculate spatial and temporal embeddings (e.g. pixel position, or print position and sequence ordering) with the following formulation:

$$\mathcal{F}(\mathbf{x}) = \begin{bmatrix} \mathrm{x}_1 \cdots x_d \\ \sin(\mathrm{x}_1 \cdot w_1 \cdot 2\pi) \cdots sin(x_d \cdot w_d \cdot 2\pi) \\ \cos(\mathrm{x}_1 \cdot w_1 \cdot 2\pi) \cdots x_d \cdot cos(w_d \cdot 2\pi) \end{bmatrix} \tag{4.1}$$

Where **w** is a learned vector. We then pass these additional features alongside original node features into an MLP to get updated embeddings per node. We apply this before the graph is passed to the GNN.

Additionally, the baseline U-Net conditions features with diffusion time. It does this using learned scaling and shift parameters. We apply a similar approach to the nodes of the dual-graph GNN, learning scale and shifts parameters per node type based upon diffusion time.

Algorithm 6 Time Conditioning for Heterogeneous Graph

```
Require: Hetero Graph and Diffusion Time: g, time_embedding
1: node_scales, node_shifts ← MLP(time_embedding)
2: for node_type in g do
3:     node[node_type] *= node_scales[node_type] + 1
4:     node[node_type] += node_shifts[node_type]
5: end for
6: return g
```

In practice, these additional conditioning strategies made little to no improvement, but involve almost no additional overhead. We hypothesise that improvements were only minor due to the strong spatial inductive bias of the dual-graph GNN and the U-Net. Moreover the U-Net is already heavily conditioned on diffusion time, which means that further conditioning only has a minor overall impact.

Alternative Dual-Graph GNN Architectures

To identify a better architecture, we first conduct a small ablation on GNN depth by evaluating the HGT network with different number of layers. Readers should also be warned that memory usage for large or dense GNNs are typically much higher than more localised models such as the U-Net. This is because one node may connect to hundreds of neighbouring node which leads to large activation maps. This is particularly relevant in our case, as print-line or spatial edges on the dual-graph can be dense, connecting to hundreds of pixels at once. In our experiments we didn't find substantial improvements by utilising deeper networks, but found decreasing them would degrade performance.

Rather than trying to theoretically justify the best possible GNN architecture, we based our approach on empirical performance. We explored several alternative GNN architectures beyond the HGT, including: Graph Convolution (Morris et al. [2021]), Graph Attention Networks (GAT Veličković et al. [2018]), GATv2 (Brody et al. [2022]), SAGE Convolution (Hamilton et al. [2018]) and Heterogeneous Attention Networks (HAN Wang et al. [2021]), view A.2 for a theoretical overview. We find very comparable performance between these architectures despite different computational overheads, see Figure 4.7 for an overview. However, these were small experiments run for only 10,000 steps due to computational constraints. When evaluating architectures, one should consider factors such as parameter count and runtime complexity; for instance, attention-based methods have quadratic complexity. It is known that sparse

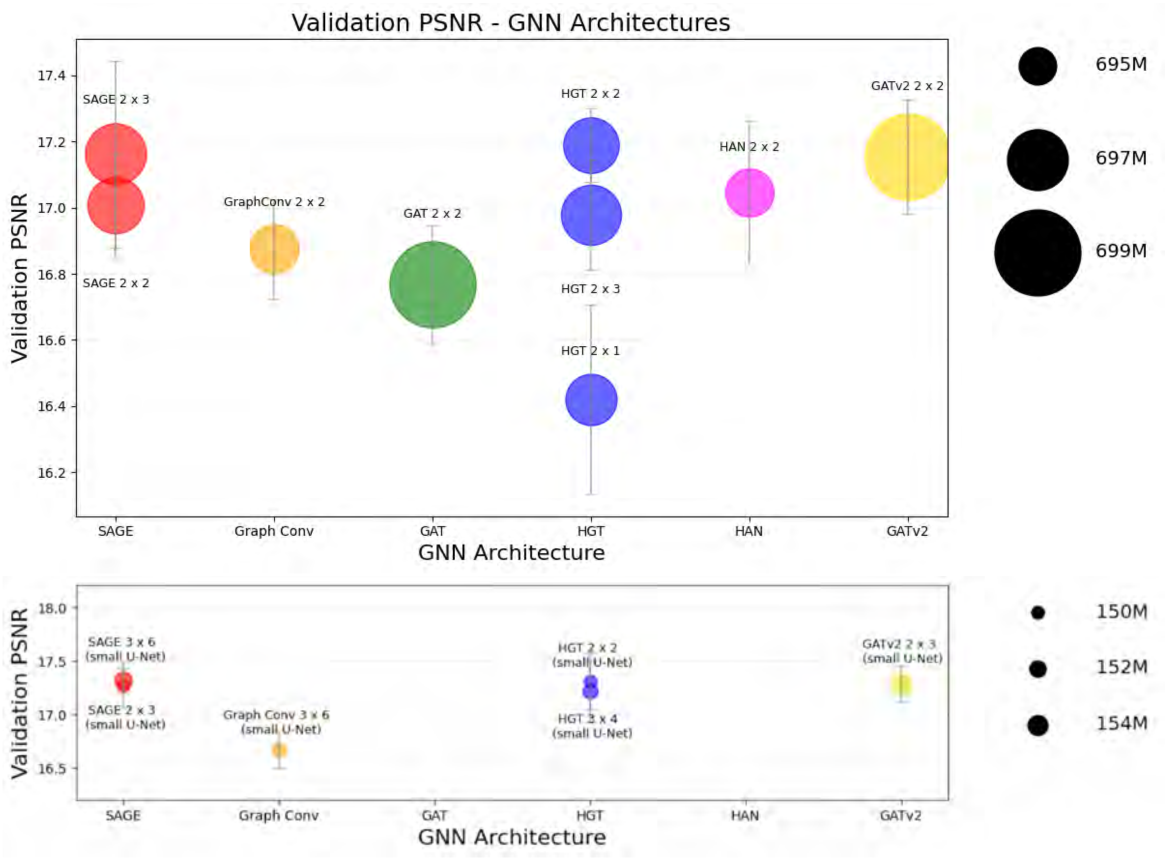

Figure 4.7: **PSNR for Different GNN Architectures.** Bubble area indicates number of parameters plotted against PSNR for models run for 10k steps. Models have very comparable performance to one another. Bottom indicates smaller U-net models with large GNNs, also notice very comparable performance.

representations such as GNNs have worse computational trade-offs compared with dense models with respect to parameter count. GAT, GATv2, HAN and HGT all have the most similar complexity profile due to self-attention, we found that GATv2 and HGT performed the best. Interestingly SAGE provided a highly competitive and more computationally efficient alternative. Graph Convolution appeared to be the most computationally efficient approach but it did not perform as well in practice.

We also investigate shifting the balance of parameters from the U-Net to the GNN. We must balance the load between image generation (from the U-Net) and useful conditioning information (from the GNN). In the baseline approach the vast majority of parameters are reserved for the generative model and only a few reserved for the conditioning model. To explore the idea of changing the balance, we switch to a smaller U-Net by reducing dimensional multipliers and the number of ResNet blocks, reducing the number of U-Net parameters by around 75%. We then increase the number of dual-graph GNN blocks and their depth, following advice from Li et al. [2020] on training deeper graphs. Interestingly we find almost identical reconstruction performance in these approaches (see 4.7), at the cost of significantly fewer parameters. However, memory overheads at training time stay roughly equivalent due the larger activation maps of the dual-graph GNN (e.g. it has several dense areas of connectivity at high resolution). To provide a fair assessment we also train small U-Net models with baseline GNN depths and see very little difference between the two. Perhaps a sensible conclusion to draw from this is to note that given our small dataset size the model is simply highly over-parameterised. These results may indicate we have reached a natural plateau through this technique. However, we observed small quantitative differences were noticed between the smaller and larger models and we are wary of the fact that the numerical validation procedures are noisy given the dataset size and these models were not trained to convergence. Furthermore, we had to be mindful of available computing constraints. We decide to present a final model only in the large variant using HGT 2x2.

Node Sampling

One observation of the dual-graph representation is that certain edge types are dense. Namely, print line edges that run along the line of the nozzle head connect to every pixel along the line. These are by far the most numerous type of connection on the graph, and there should be a high level of redundancy in the information they carry. Furthermore, in other work on diffusion generation, they show improved generation by augmenting low-resolution images with low levels of noise (Saharia et al. [2022]). Additionally, it is common practice to sample a neighbourhood of nodes in GNN frameworks (Hu et al. [2020]). In a similar vein, we implement node sub-sampling at the start of each model pass for this type of node. We randomly sample a subset of spatial and print line edges using a Bernoulli distribution where $p = 0.2$.

$$\text{Sample}(\mathcal{E}_t) = \{e_i \in \mathcal{E} \mid \mathbb{I}(\text{Bernoulli}(p) = 1)\} \tag{4.2}$$

Remarkably, we see no significant degradation in validation reconstruction or training loss using this method and receive substantial memory savings. We leave it to future work to explore this phenomenon further and to find best p values.

Image Alignment Ablation

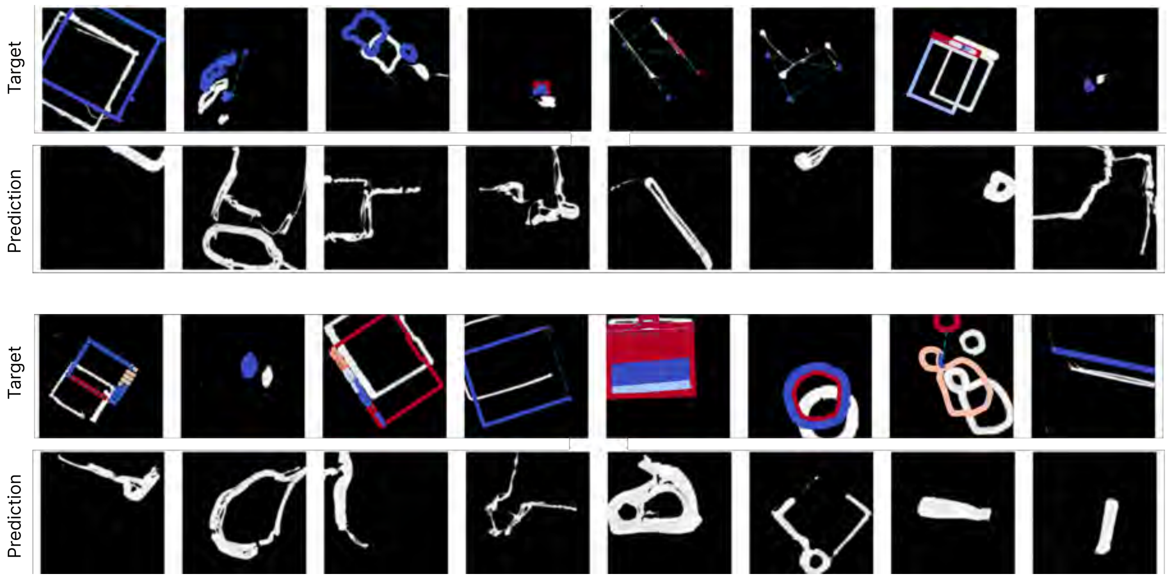

Figure 4.8: **Samples from Model without Alignment.** Top rows show graph overlaid on target image without alignment i.e. notice how the graph and image do not match. Bottom rows are the prediction given the un-aligned graph. This model was trained for 10,000 steps.

To study the effect of image alignment on the system, random translations were applied to the image only. As can be seen in Figure 4.8 the samples from this model are very poorly conditioned and the visual quality has also suffered. This demonstrates the important of data pre-processing and the image alignment processed outlined in section 3. The model here was trained for 10,000 steps, so it may have possibly improved later in training. However, by giving the model aligned images we allow the model to focus on the important part of the generation.

4.5 Final Model

The model with best parameters was trained for longer and utilised additional data. Half of these extra data points consisted of prints generated using graphs supplied by CAM, while the other half were random samples from the Quick Draw dataset (Ha and Eck [2017]). It was trained for 60+ hours on the HPC, using model check-pointing to side-step job time limits, achieving a validation PSNR of 18.26. The final model demonstrates a solid understanding of the PSP relationship. Samples are generated with 32 sampling steps unless otherwise specified.

4.5.1 Generated Samples

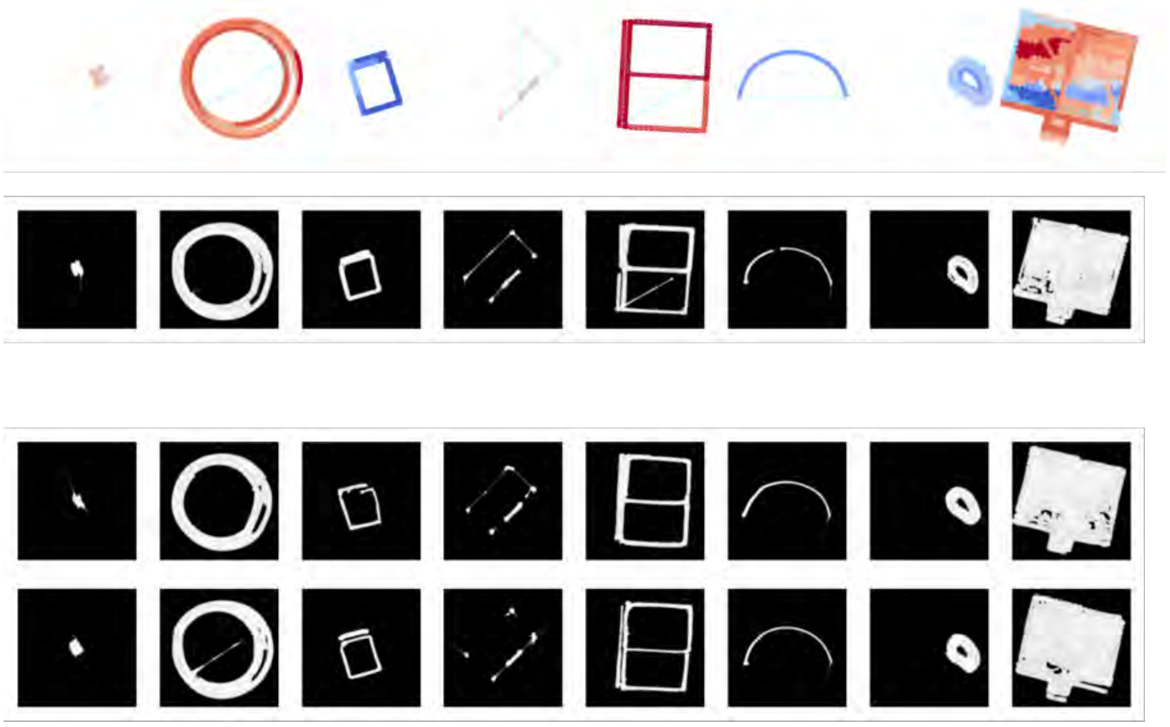
Figure 4.9: **Validation Sample Generation.** The top two rows show validation samples and their corresponding graphs. The bottom rows are generated samples conditioned on the same graphs.

We conduct a qualitative analysis of samples to further analyse the results. Figure 4.9 and Figure 4.10 display generated samples from the model. In Figure 4.10, geometrically identical graphs are generated with different flow rate parameters. When the flow rate is too low, the layer quality deteriorates and exhibits line artefacts. The model has clearly learned this relationship, as the reconstructions reflect similar types of errors. In contrast, high flow rate layers appear to suffer fewer defects and display greater consistency. The model appears to generate highly plausible samples in this case, producing better results for images with higher flow rates. This is unsurprising, as under-extruded samples exhibit higher levels of stochasticity, making them more difficult to predict accurately. The model correctly learns when and where there is a higher likelihood of stringing artefacts. Stringing commonly occurs between parts of the print but may also appear as the printer moves on to a new print. It seems that the model learns this distribution, and although the predictions are not accurate in terms of numerical metrics (e.g. PSNR/MSE), it appears to generate the same type of errors.

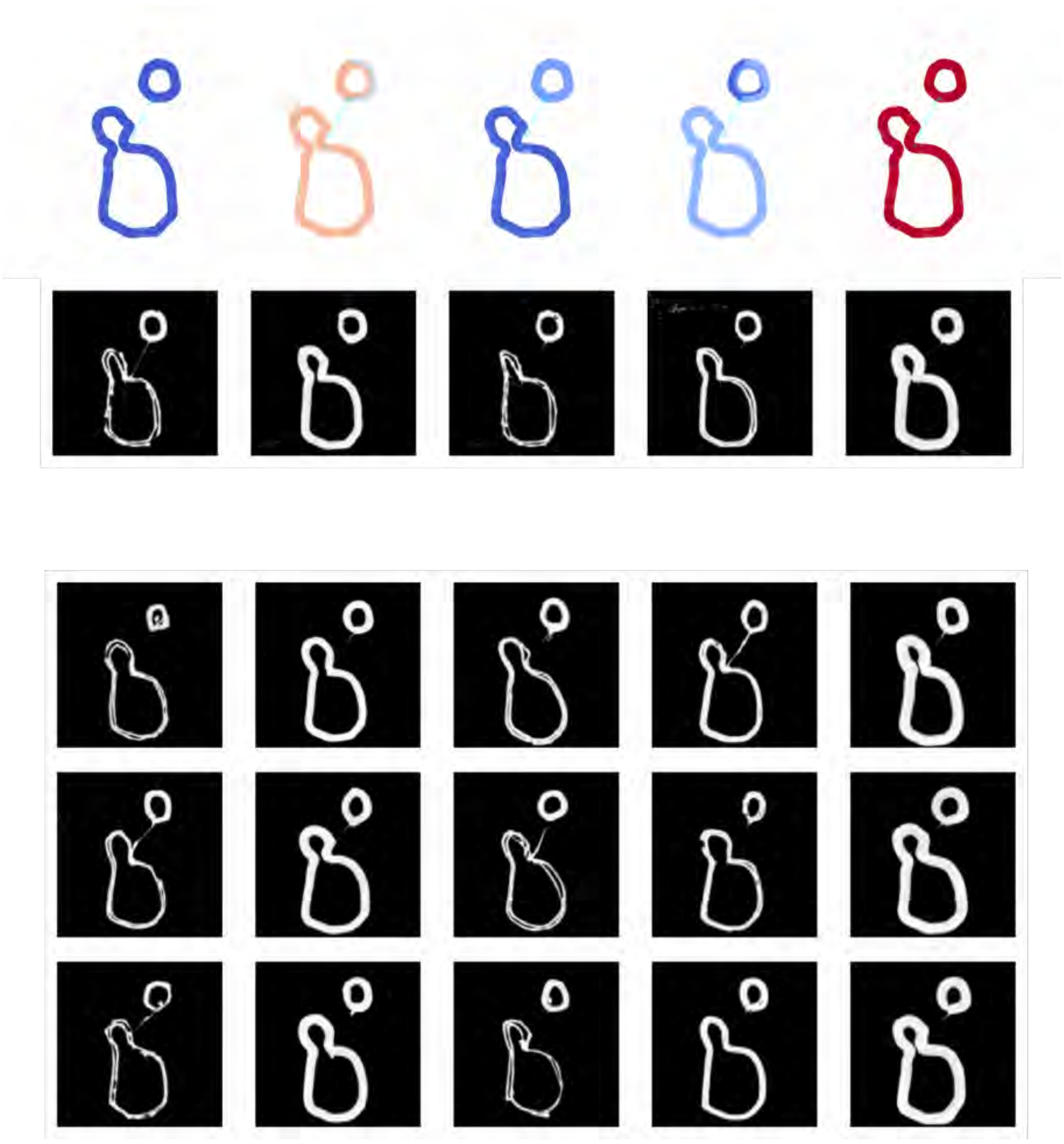
Figure 4.10: **Varying Flow Rate Sample Generation.** Matching Figure 4.9, this figure displays validation samples over varying flow rate graphs.

We explore the difference between the SDE and ODE samplers over different numbers of sampling steps. They both produce plausible samples until the number of steps is reduced to below 6. Figure 4.11 shows that the ODE sampler produces near identical outputs. Similarly to Karras et al. [2022] we see higher performance using the stochastic sampler, we display validation PSNR over sampling steps in 4.12.

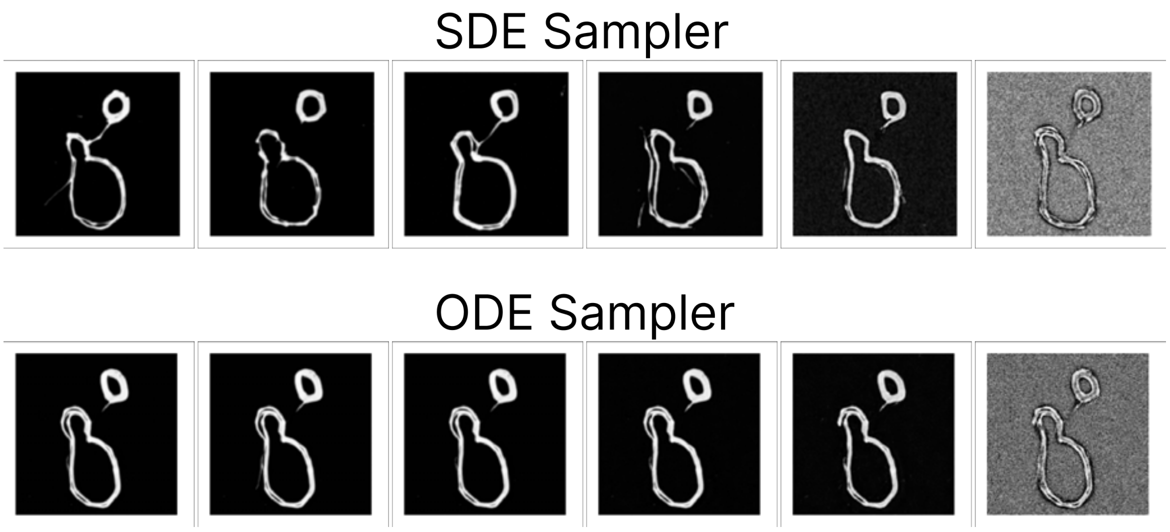

Figure 4.11: **Effects of Different Samplers.** generated samples using different sampling schemes. The only source of randomness in the ODE is the initial noise.

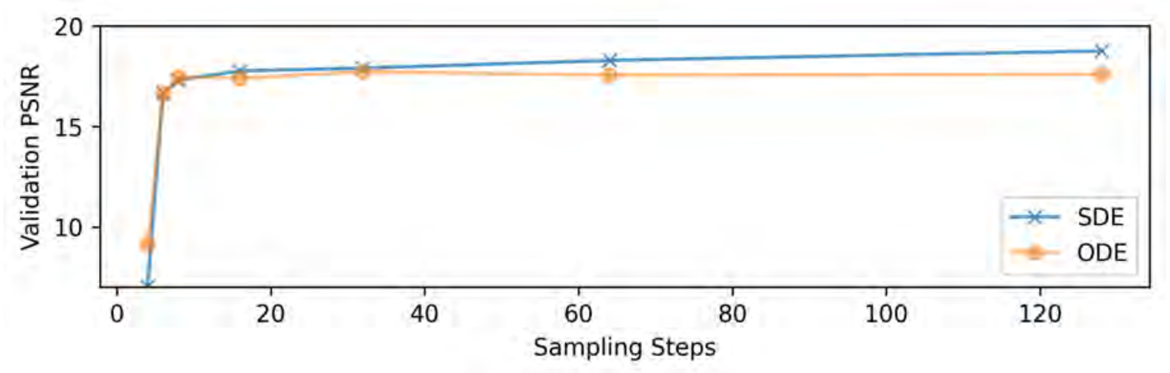

Figure 4.12: **PSNR by sample steps.** Shows validation PSNR over sampling steps for both types of samplers.

We present additional investigations into the trained model and the conditional units in appendix section A.4.

Out-of-Distribution Generation

To demonstrate model performance on out-of-distribution graphs, we display samples were generated from text graphs, a task the model was not trained on. For this test, we also sweep flow rate values from 1 to 400%. The flow rate values at training time were set between 10 and 300% of the default, so this allows us to examine both out-of-distribution geometries and parameters. Figure 4.13 highlights that the model behaves as expected: low flow rates yield adhesion errors and extrusion errors, and high flow rates lead to over-extrusion. Again we see very plausible stringing artefacts that follow inactivate print line connections. This indicates that the model has learned representations that generalise to both unseen parameters and unseen geometries.

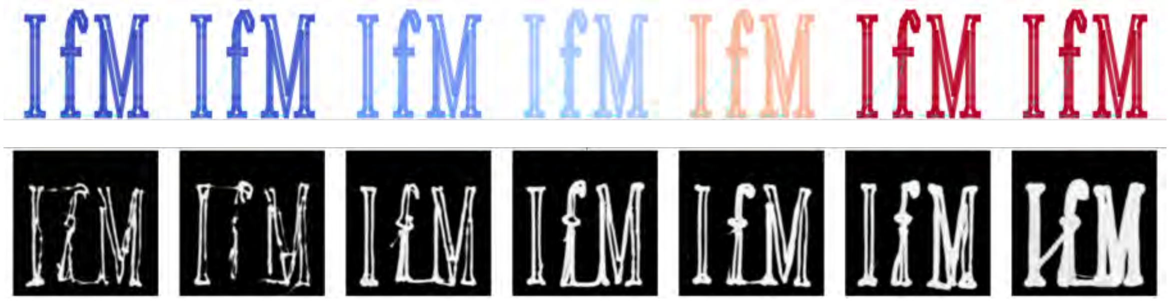

Figure 4.13: **Out-of-Distribution Generation.** Generated samples using text graphs and flow-rates that extend beyond the training distribution. From left to right flow rate is set to [1, 10, 50, 100, 200, 300, 400].

4.5.2 Case Study: Process Parameter Optimisation

To showcase our model's capabilities and validate its performance, we present a test-case focused on process parameter optimisation. Specifically, given a target render, we search for optimal flow rate parameters that minimise the difference between the render and the generated samples. In Figure 4.14 we observe a minimum error at a specific flow rate value. By comparing the optimal sample, the predicted flow rate value, and the target render, we observe a clear similarity between the images. However, this is an artificial example, so it is unclear whether these findings would hold true for real world examples. To explore this idea further, an additional dataset comprising of 55 prints using the same geometry with different flow rates was collected.

The collected dataset comprised of the Clare College crest, and was printed with linearly-spaced flow rates from 10 to 300 (see Figure 4.15), and randomly varying flow rates. We find MSE between the image and the sample to be a noisy metric. This is amplfied by the non-determinism in both the generated samples and the real data. To mitigate this issue, we explore two perceptual loss methods. We first used a perceptual loss by comparing high-level features from a pre-trained model (VGG16 Simonyan and Zisserman [2015]) to match images based on perceptual qualities rather than pixel

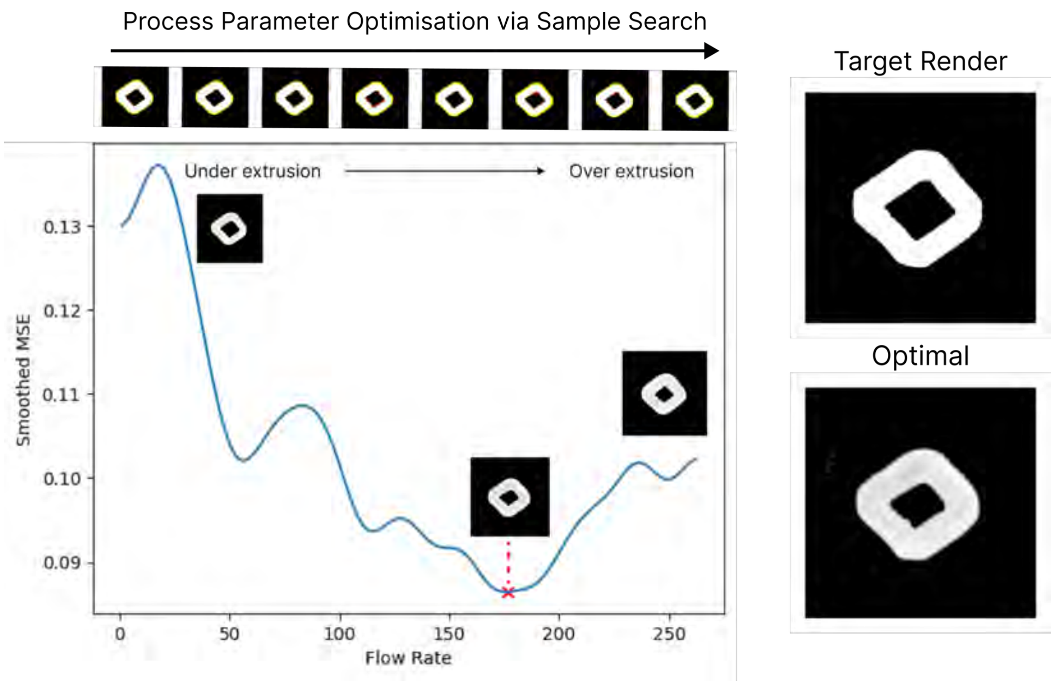

Figure 4.14: **Optimal Parameter Search.** By sampling a target render from the model with various process parameters, we can perform a simple MSE optimisation to find the parameter most likely to result in the target render, and therefore to avoid over or under-extrusion errors. In this example, we use a target render with a constant flow rate parameter. Yet even this task is highly dependent on input geometry. We generate one sample per flow-rate value and the output is noisy. To alleviate this, we smooth predictions with a Gaussian kernel $\sigma = 1$. An alternative here would be to perform Bayesian inference in light of these noisy samples and a flow-rate prior.

differences. Subsequently we used the initial convolutional layer of our trained diffusion model whilst skipping graph conditioning. We found the latter to produce better results. The error profile across different flow rates is presented in Figure 4.16. This reinforces our understanding that the model produces high-quality samples, because both real and generated samples that share the same parameters are neighbours in latent space. Interestingly, we find the difference in latent space to be small. A more considered approach would select the features with highest correlation to flow-rate, and pass outputs through a softmax to amplify likely predictions.

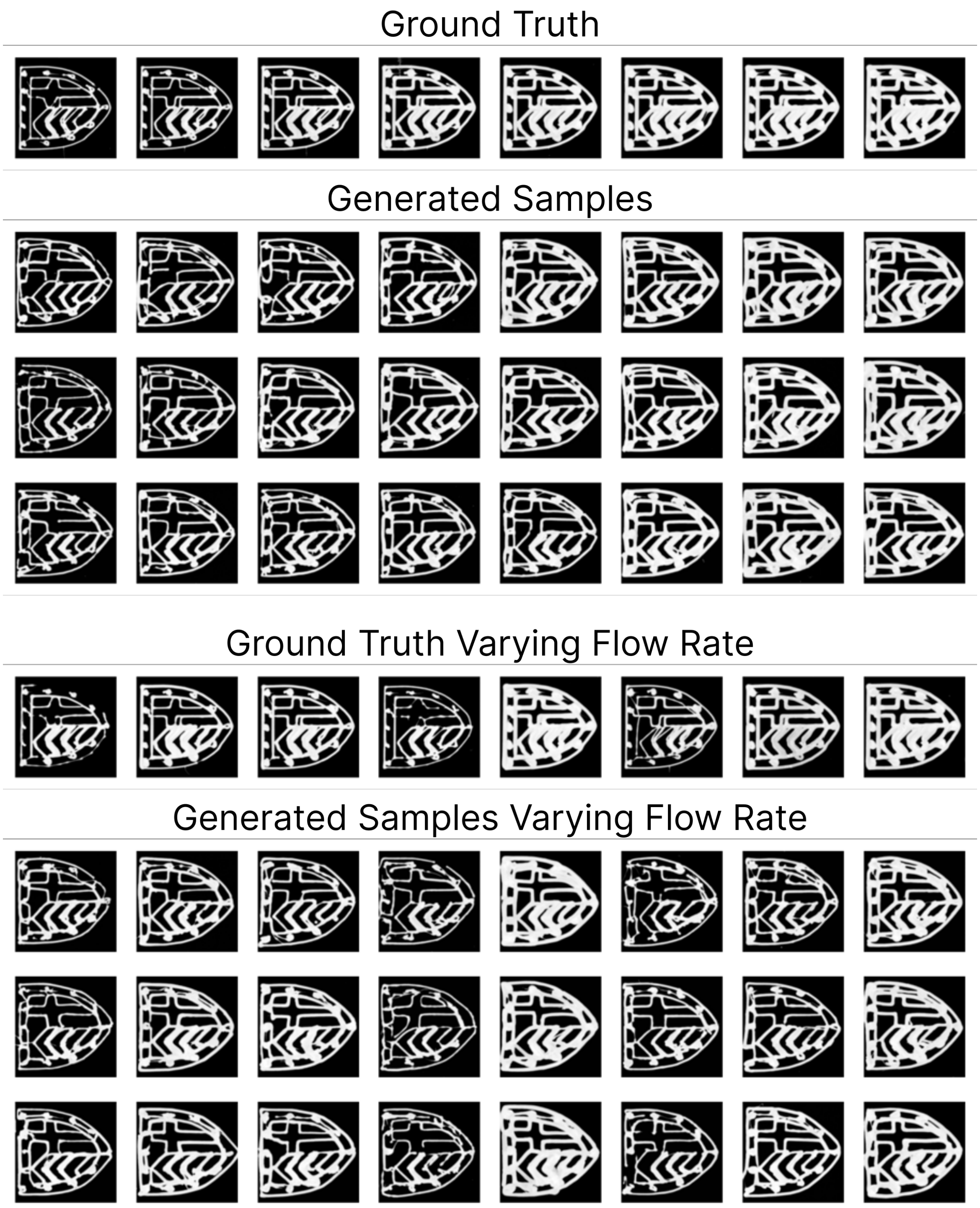
Ground Truth
Generated Samples
Ground Truth Varying Flow Rate
Generated Samples Varying Flow Rate

Figure 4.15: **Clare College Crest Data and Generated Samples.** This figures presents printed layer images of the Clare College crest, and generated samples from the same input graph underneath. The top half use linearly spaced but constant flow rate. The bottom half have randomly varying flow rates.

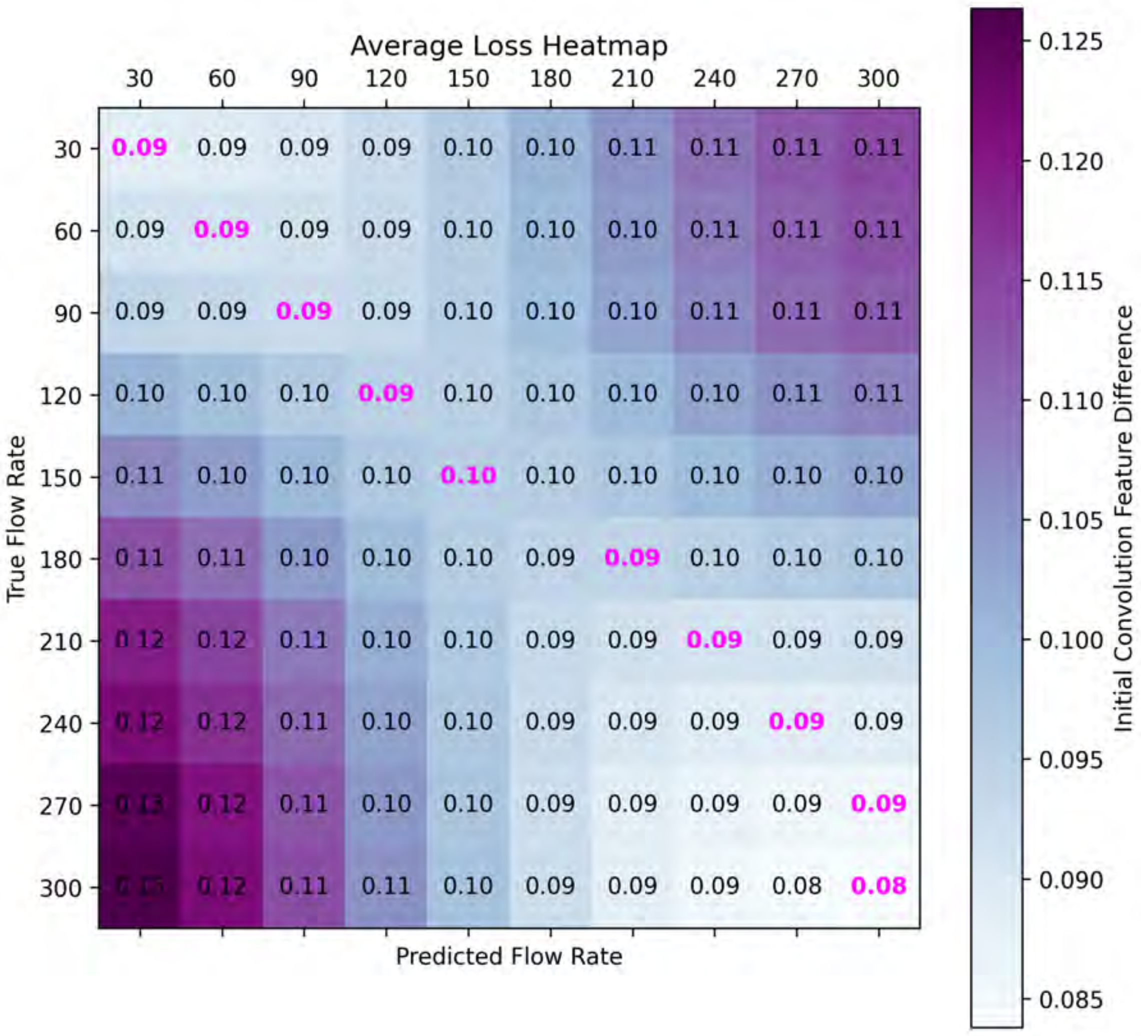

Figure 4.16: **Normalised Average Feature Difference between Samples and Ground Truth.** Ground truth layer images of the Clare College crest and generated samples were compared by calculating their difference in features using the trained model. These errors are averaged over 30 samples per flow rate. Rows are normalised to sum to 1. Maximum value per row highlighted with magenta.

Chapter 5

Conclusions

This final section summarises the project's findings and the exciting research offshoots which are currently being explored. It begins with a high-level discussion of the project by highlighting its achievements, followed by a critical evaluation. The report concludes by presenting ongoing and future work, namely the integration of additional sensors e.g. a nozzle-head camera. This includes a discussion concerning the applicability of this work in other domains and the potential benefits of doing so.

5.1 Discussion

This project successfully managed to condition a diffusion model with spatial graph inputs. Despite a limited amount of real data, the model was able to predict plausible FDM layer images which shared characteristics with the ground truth data. Furthermore, the novel dual-graph method was proven to be an effective and flexible way to condition images. The system showed better reconstruction accuracy, demonstrating that the conditioning strategy was more effective than other methods. Overall, this work is an innovative and successful approach towards mapping the PSP relationship in FDM. We also believe these contributions should extend beyond the AM domain (this is discussed in the next section). Furthermore, we were able to gather a first-of-its-kind dataset of over 2,500 FDM layer images. Alongside this, we presented a new robust affine matching system and pre-processing pipeline.

Despite its successes, this project had a number of limitations and shortcomings. For example, the dataset consisted of single-layer images - this does not match real-world applications which utilise hundreds of layers. Furthermore, it is challenging to analyse how the size of the dataset held back the model's predictive capability. This is especially pertinent as model performance typically grows with a power-law with respect to dataset size and data quality (Li et al. [2024]). The dataset was not only small, but had issues such as image misalignment. This was compounded by other mechanical failures which are difficult to track and represent in the input graphs. This ultimately resulted in a dataset with higher inherent stochasticity, which could have been avoided by using a different printer design (e.g. the moving belt caused many of the issues). Critiquing this point further, the project did not incorporate the importance of the physical printer set-up e.g. print bed height adjustment, type of filament feed-stock and temperature control. It is known that environmental factors (e.g. humidity) play an important role in FDM (Baechle-Clayton et al. [2022]), meaning that the trained model may not be applicable in environmental settings different from those of the IfM lab.

Validating the model's performance proved difficult, particularly in numerically assessing the reconstruction accuracy. This was because image alignment was not perfect meaning the model may (correctly) predict shifted outputs which are heavily penalised by simple metrics such as PSNR. Although, this was somewhat alleviated by validating optimal parameters with a real-world test case. Perhaps a more sophisticated user study would have been a better predictor of model success. Allowing a human operator to use the model in their workflow would provide better insight into its effectiveness in reducing trial-and-error. Finally, a methodological weakness of the project is that the baseline approach includes one of the proposed conditional units. This was done as this was the simplest way to provide the model all available information, however, for evaluation purposes using a simple line mask baseline could provide a more insightful comparison to evaulate the proposed units.

5.2 Future Work

5.2.1 Future Work in FDM

There is currently ongoing work to integrate video data from a nozzle-head camera into the dual-graph system. The dual-graph is highly flexible and allows many types of conditioning to be integrated easily. We also want to adapt the system to the multi-layer setting. We hypothesise that a dual-graph representation would have significant benefits over a learnable render in this scenario, as it allows far more complex interactions of nodes projecting down onto a surface. We are gathering a larger dataset of around 10,000 data samples to facilitate this improved system.

A promising avenue for future research could include reversing the sampling procedure to recover the conditioning graph. Although we showed it was possible to find optimal parameters through a forward optimisation approach, there may be better methods to go from an ideal render to optimal process parameters. Finally, a deeper analysis of the inner workings of the model and the dual graph may reveal a representation of the underlying physical process. This could be useful for researchers who aim to understand the PSP relation. Finally, we could gain deeper insight into the AM process by calculating 'sample standard deviation' for a given input graph, we could verify whether this is a useful metric to validate potential instabilities in layer images.

Another interesting direction would allow the GNN edges to be dynamically predicted at run time. For example, when the model is being trained with images that have little noise, this could help dynamically adjust instances with poor image alignment or catastrophic errors. There is some prior work in these types of dynamic graphs e.g. Spatial Graph Convolutional Networks learn a spatial neighbourhood of edge connections (Danel et al. [2020]). On a different point, in 4.4 we showed highly promising results in sub-sampling graphs nodes. One promising future avenue would be randomly draw the value of p and condition the model on this value, in a method similar to Saharia et al. [2022] i.e. we should view sub-sampling as a type of data augmentation.

5.2.2 Applications in Other Domains

Many applications can benefit from high-accuracy spatial conditioning. Conditional image generation tasks such as inpainting or text-to-image can benefit from higher degrees of spatial conditioning. For example, human face generation may be improved by utilising the dual-graph conditioning system and using facial features instead of the space-time graph. Common challenges with current approaches include the generation of text within images, which often fails to accurately adhere to the given text prompt, and the generation of human hands, which frequently results in errors. By utilising a dual-graph system, models may be able to overcome these challenges. In addition, a dual-graph may naturally lend itself as a user-friendly interface for humans to interact with such systems to enforce strong but flexible spatial conditioning. To test out this hypothesis, we want to apply this method to an existing segmentation dataset. We postulate this will improve generation coherence by connecting related parts of the image via a dual-graph using class-conditional nodes. Another potential application could be satellite image prediction. Climate modelling often makes use of sparse ‘off-the-grid’ data points which can be difficult to integrate into existing systems. The benefit of utilising a dual-graph beyond the performance gains is the flexibility it allows. Any number of observations or nodes can be added to the graph and integrated into the system to provide conditioning information such as weather beacons being spatially connected to a satellite image for weather forecasting applications.

References

Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J. Ballard, Joshua Bambrick, Sebastian W. Bodenstein, David A. Evans, Chia-Chun Hung, Michael O'Neill, David Reiman, Kathryn Tunyasuvunakool, Zachary Wu, Akvilė Žemgulytė, Eirini Arvaniti, Charles Beattie, Ottavia Bertolli, Alex Bridgland, Alexey Cherepanov, Miles Congreve, Alexander I. Cowen-Rivers, Andrew Cowie, Michael Figurnov, Fabian B. Fuchs, Hannah Gladman, Rishub Jain, Yousuf A. Khan, Caroline M. R. Low, Kuba Perlin, Anna Potapenko, Pascal Savy, Sukhdeep Singh, Adrian Stecula, Ashok Thillaisundaram, Catherine Tong, Sergei Yakneen, Ellen D. Zhong, Michal Zielinski, Augustin Žídek, Victor Bapst, Pushmeet Kohli, Max Jaderberg, Demis Hassabis, and John M. Jumper. Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature*, pages 1–3, May 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-07487-w. URL https://www.nature.com/articles/s41586-024-07487-w. Publisher: Nature Publishing Group.

Waad Almasri, Florence Danglade, Dimitri Bettebghor, Faouzi Adjed, and Fakhreddine Ababsa. Deep Learning for Additive Manufacturing-driven Topology Optimization. *Procedia CIRP*, 109:49–54, January 2022. ISSN 2212-8271. doi: 10.1016/j.procir.2022.05.317. URL https://www.sciencedirect.com/science/article/pii/S2212827122007673.

Maggie Baechle-Clayton, Elizabeth Loos, Mohammad Taheri, and Hossein Taheri. Failures and Flaws in Fused Deposition Modeling (FDM) Additively Manufactured Polymers and Composites. *Journal of Composites Science*, 6(7):202, July 2022. ISSN 2504-477X. doi: 10.3390/jcs6070202. URL https://www.mdpi.com/2504-477X/6/7/202. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Qinsheng Zhang, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, Tero Karras, and Ming-Yu Liu. eDiff-I: Text-to-Image Diffusion Models with an Ensemble of Expert Denoisers, March 2023. URL http://arxiv.org/abs/2211.01324. arXiv:2211.01324 [cs].

Douglas A. J. Brion and Sebastian W. Pattinson. Generalisable 3D printing error detection and correction via multi-head neural networks. *Nature Communications*, 13(1):4654, August 2022a. ISSN 2041-1723. doi: 10.1038/s41467-022-31985-y. URL https://www.nature.com/articles/s41467-022-31985-y. Publisher: Nature Publishing Group.

Douglas A. J. Brion and Sebastian W. Pattinson. Quantitative and Real-Time Control of 3D Printing Material Flow Through Deep Learning. *Advanced Intelligent Systems*, 4(11):2200153, 2022b. ISSN 2640-4567. doi: 10.1002/aisy.202200153. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202200153. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202200153.

Douglas A. J. Brion, Matthew Shen, and Sebastian W. Pattinson. Automated recognition and correction of warp deformation in extrusion additive manufacturing. *Additive Manufacturing*, 56:102838, August 2022. ISSN 2214-8604. doi: 10.1016/j.addma.2022.102838. URL https://www.sciencedirect.com/science/article/pii/S2214860422002378.

Shaked Brody, Uri Alon, and Eran Yahav. How Attentive are Graph Attention Networks?, January 2022. URL http://arxiv.org/abs/2105.14491. arXiv:2105.14491 [cs].

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. URL http://arxiv.org/abs/2104.13478. arXiv:2104.13478 [cs, stat].

Filipa G. Cunha, Telmo G. Santos, and José Xavier. In Situ Monitoring of Additive Manufacturing Using Digital Image Correlation: A Review. *Materials*, 14(6):1511, January 2021. ISSN 1996-1944. doi: 10.3390/ma14061511. URL https://www.mdpi.com/1996-1944/14/6/1511. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

Tomasz Danel, Przemysław Spurek, Jacek Tabor, Marek Śmieja, Łukasz Struski, Agnieszka Słowik, and Łukasz Maziarka. Spatial Graph Convolutional Networks, July 2020. URL http://arxiv.org/abs/1909.05310. arXiv:1909.05310 [cs, stat].

Prafulla Dhariwal and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis, June 2021. URL http://arxiv.org/abs/2105.05233. arXiv:2105.05233 [cs, stat].

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021. URL http://arxiv.org/abs/2010.11929. arXiv:2010.11929 [cs].

G.D. Evangelidis and E.Z. Psarakis. Parametric Image Alignment Using Enhanced Correlation Coefficient Maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(10):1858–1865, October 2008. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.113. URL http://ieeexplore.ieee.org/document/4515873/.

Azade Farshad, Yousef Yeganeh, Yu Chi, Chengzhi Shen, Björn Ommer, and Nassir Navab. SceneGenie: Scene Graph Guided Diffusion Models for Image Synthesis, April 2023. URL http://arxiv.org/abs/2304.14573. arXiv:2304.14573 [cs].

Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, May 2019. URL https://github.com/pyg-team/pytorch_geometric.

Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In Martin A. Fischler and Oscar Firschein, editors, *Readings in Computer Vision*, pages 726–740. Morgan Kaufmann, San Francisco (CA), January 1987. ISBN 978-0-08-051581-6. doi: 10.1016/B978-0-08-051581-6.50070-2. URL https://www.sciencedirect.com/science/article/pii/B9780080515816500702.

John M. Gardner, Kevin A. Hunt, Austin B. Ebel, Evan S. Rose, Sean C. Zylich, Benjamin D. Jensen, Kristopher E. Wise, Emilie J. Siochi, and Godfrey Sauti. Machines as Craftsmen: Localized Parameter Setting Optimization for Fused Filament Fabrication 3D Printing. *Advanced Materials Technologies*, 4(3):1800653, 2019. ISSN 2365-709X. doi: 10.1002/admt.201800653. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/admt.201800653. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/admt.201800653.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry, June 2017. URL http://arxiv.org/abs/1704.01212. arXiv:1704.01212 [cs].

Yilin Guo, Wen Feng Lu, and Jerry Ying Hsi Fuh. Semi-supervised deep learning based framework for assessing manufacturability of cellular structures in direct metal laser sintering process. *Journal of Intelligent Manufacturing*, 32(2):347–359, February 2021. ISSN 0956-5515. doi: 10.1007/s10845-020-01575-0. URL https://doi.org/10.1007/s10845-020-01575-0.

David Ha and Douglas Eck. A Neural Representation of Sketch Drawings, May 2017. URL http://arxiv.org/abs/1704.03477. arXiv:1704.03477 [cs, stat].

Ghazaleh Haghiashtiani, Kaiyan Qiu, Jorge D. Zhingre Sanchez, Zachary J. Fuenning, Priya Nair, Sarah E. Ahlberg, Paul A. Iaizzo, and Michael C. McAlpine. 3D printed patient-specific aortic root models with internal sensors for minimally invasive applications. *Science Advances*, 6(35):eabb4641, August 2020. doi: 10.1126/sciadv.abb4641. URL https://www.science.org/doi/10.1126/sciadv.abb4641. Publisher: American Association for the Advancement of Science.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs, September 2018. URL http://arxiv.org/abs/1706.02216. arXiv:1706.02216 [cs, stat].

Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. Vision GNN: An Image is Worth Graph of Nodes, November 2022. URL http://arxiv.org/abs/2206.00272. arXiv:2206.00272 [cs].

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv:1512.03385 [cs].

Nathan Hertlein, Philip R. Buskohl, Andrew Gillman, Kumar Vemaganti, and Sam Anand. Generative adversarial network for early-stage design flexibility in topology optimization for additive manufacturing. *Journal of Manufacturing Systems*, 59:

675–685, April 2021. ISSN 0278-6125. doi: 10.1016/j.jmsy.2021.04.007. URL https://www.sciencedirect.com/science/article/pii/S027861252100087X.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, December 2020. URL http://arxiv.org/abs/2006.11239. arXiv:2006.11239 [cs, stat].

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous Graph Transformer, March 2020. URL http://arxiv.org/abs/2003.01332. arXiv:2003.01332 [cs, stat].

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Aapo Hyvärinen. Estimation of Non-Normalized Statistical Models by Score Matching. *J. Mach. Learn. Res.*, 6:695–709, December 2005. ISSN 1532-4435.

Yayati Jadhav, Joseph Berthel, Chunshan Hu, Rahul Panat, Jack Beuth, and Amir Barati Farimani. StressD: 2D Stress estimation using denoising diffusion model. *Computer Methods in Applied Mechanics and Engineering*, 416:116343, November 2023. ISSN 0045-7825. doi: 10.1016/j.cma.2023.116343. URL https://www.sciencedirect.com/science/article/pii/S004578252300467X.

Ayush Jain, Ehsan Haghighat, and Sai Nelaturi. LatticeGraphNet: A two-scale graph neural operator for simulating lattice structures, February 2024. URL http://arxiv.org/abs/2402.01045. arXiv:2402.01045 [cs].

Zeqing Jin, Zhizhou Zhang, and Grace X. Gu. Automated Real-Time Detection and Prediction of Interlayer Imperfections in Additive Manufacturing Processes Using Artificial Intelligence. *Advanced Intelligent Systems*, 2(1):1900130, 2020. ISSN 2640-4567. doi: 10.1002/aisy.201900130. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.201900130. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.201900130.

Marshall V. Johnson, Kevin Garanger, James O. Hardin, J. Daniel Berrigan, Eric Feron, and Surya R. Kalidindi. A generalizable artificial intelligence tool for identification and correction of self-supporting structures in additive manufacturing processes. *Additive Manufacturing*, 46:102191, October 2021. ISSN 2214-8604. doi: 10.1016/j.addma.2021.102191. URL https://www.sciencedirect.com/science/article/pii/S2214860421003535.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the Design Space of Diffusion-Based Generative Models, October 2022. URL http://arxiv.org/abs/2206.00364. arXiv:2206.00364 [cs, stat].

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs].

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, February 2017. URL http://arxiv.org/abs/1609.02907. arXiv:1609.02907 [cs, stat].

Sebastian Larsen and Paul A. Hooper. In-situ Anomaly Detection in Additive Manufacturing with Graph Neural Networks, May 2023. URL http://arxiv.org/abs/2305.02695. arXiv:2305.02695 [cs, eess].

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. DeeperGCN: All You Need to Train Deeper GCNs, June 2020. URL http://arxiv.org/abs/2006.07739. arXiv:2006.07739 [cs, stat].

Hao Li, Yang Zou, Ying Wang, Orchid Majumder, Yusheng Xie, R. Manmatha, Ashwin Swaminathan, Zhuowen Tu, Stefano Ermon, and Stefano Soatto. On the Scalability of Diffusion-based Text-to-Image Generation, April 2024. URL http://arxiv.org/abs/2404.02883. arXiv:2404.02883 [cs].

Yuxuan Li, Zhangyue Shi, Chenang Liu, Wenmeng Tian, Zhenyu Kong, and Christopher B. Williams. Augmented Time Regularized Generative Adversarial Network (ATR-GAN) for Data Augmentation in Online Process Anomaly Detection. *IEEE Transactions on Automation Science and Engineering*, 19(4):3338–3355, October 2022. ISSN 1558-3783. doi: 10.1109/TASE.2021.3118635. URL https://ieeexplore.ieee.org/document/9592834. Conference Name: IEEE Transactions on Automation Science and Engineering.

Christos Margadji, Douglas A. J. Brion, and Sebastian W. Pattinson. Iterative learning for efficient additive mass production. *Additive Manufacturing*, 89:104271, June 2024. ISSN 2214-8604. doi: 10.1016/j.addma.2024.104271. URL https://www.sciencedirect.com/science/article/pii/S2214860424003178.

Rameshwar Mishra and A. V. Subramanyam. Image Synthesis with Graph Conditioning: CLIP-Guided Diffusion Models for Scene Graphs, January 2024. URL http://arxiv.org/abs/2401.14111. arXiv:2401.14111 [cs].

Omar A. Mohamed, Syed H. Masood, and Jahar L. Bhowmik. Optimization of fused deposition modeling process parameters: a review of current research and future prospects. *Advances in Manufacturing*, 3(1):42–53, March 2015. ISSN 2195-3597. doi: 10.1007/s40436-014-0097-7. URL https://doi.org/10.1007/s40436-014-0097-7.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks, November 2021. URL http://arxiv.org/abs/1810.02244. arXiv:1810.02244 [cs, stat].

Joel C. Najmon, Sajjad Raeisi, and Andres Tovar. 2 - Review of additive manufacturing technologies and applications in the aerospace industry. In Francis Froes and Rodney Boyer, editors, *Additive Manufacturing for the Aerospace Industry*, pages 7–31. Elsevier, January 2019. ISBN 978-0-12-814062-8. doi: 10.1016/B978-0-12-814062-8.00002-9. URL https://www.sciencedirect.com/science/article/pii/B9780128140628000029.

Tuan D. Ngo, Alireza Kashani, Gabriele Imbalzano, Kate T. Q. Nguyen, and David Hui. Additive manufacturing (3D printing): A review of materials, methods, applications and challenges. *Composites Part B: Engineering*, 143:172–196, June 2018. ISSN

1359-8368. doi: 10.1016/j.compositesb.2018.02.012. URL https://www.sciencedirect.com/science/article/pii/S1359836817342944.

Francis Ogoke, Quanliang Liu, Olabode Ajenifujah, Alexander Myers, Guadalupe Quirarte, Jack Beuth, Jonathan Malen, and Amir Barati Farimani. Inexpensive High Fidelity Melt Pool Models in Additive Manufacturing Using Generative Deep Diffusion, November 2023. URL http://arxiv.org/abs/2311.16168. arXiv:2311.16168 [cond-mat].

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, December 2019. URL http://arxiv.org/abs/1912.01703. arXiv:1912.01703 [cs, stat].

William Peebles and Saining Xie. Scalable Diffusion Models with Transformers, March 2023. URL http://arxiv.org/abs/2212.09748. arXiv:2212.09748 [cs].

Ryan Po, Wang Yifan, Vladislav Golyanik, Kfir Aberman, Jonathan T. Barron, Amit H. Bermano, Eric Ryan Chan, Tali Dekel, Aleksander Holynski, Angjoo Kanazawa, C. Karen Liu, Lingjie Liu, Ben Mildenhall, Matthias Nießner, Björn Ommer, Christian Theobalt, Peter Wonka, and Gordon Wetzstein. State of the Art on Diffusion Models for Visual Computing, October 2023. URL http://arxiv.org/abs/2310.07204. arXiv:2310.07204 [cs].

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis, July 2023. URL http://arxiv.org/abs/2307.01952. arXiv:2307.01952 [cs].

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents, April 2022. URL http://arxiv.org/abs/2204.06125. arXiv:2204.06125 [cs].

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015. URL http://arxiv.org/abs/1505.04597. arXiv:1505.04597 [cs].

Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Image Super-Resolution via Iterative Refinement, June 2021. URL http://arxiv.org/abs/2104.07636. arXiv:2104.07636 [cs, eess].

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, May 2022. URL http://arxiv.org/abs/2205.11487. arXiv:2205.11487 [cs].

Amirul Islam Saimon, Emmanuel Yangue, Xiaowei Yue, Zhenyu James Kong, and Chenang Liu. Advancing Additive Manufacturing through Deep Learning: A Comprehensive Review of Current Progress and Future Challenges, March 2024. URL http://arxiv.org/abs/2403.00669. arXiv:2403.00669 [cs] version: 1.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015. URL http://arxiv.org/abs/1409.1556. arXiv:1409.1556 [cs].

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics, November 2015. URL http://arxiv.org/abs/1503.03585. arXiv:1503.03585 [cond-mat, q-bio, stat].

Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution, October 2020. URL http://arxiv.org/abs/1907.05600. arXiv:1907.05600 [cs, stat].

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations, February 2021. URL http://arxiv.org/abs/2011.13456. arXiv:2011.13456 [cs, stat].

Zheren Song, Xinming Wang, Yuanyuan Gao, Junbo Son, and Jianguo Wu. A Hybrid Deep Generative Network for Pore Morphology Prediction in Metal Additive Manufacturing. *Journal of Manufacturing Science and Engineering*, 145(071005), March 2023. ISSN 1087-1357. doi: 10.1115/1.4057012. URL https://doi.org/10.1115/1.4057012.

Jeremy Straub. Initial Work on the Characterization of Additive Manufacturing (3D Printing) Using Software Image Analysis. *Machines*, 3(2):55–71, June 2015. ISSN 2075-1702. doi: 10.3390/machines3020055. URL https://www.mdpi.com/2075-1702/3/2/55. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred Neural Rendering: Image Synthesis using Neural Textures, April 2019. URL http://arxiv.org/abs/1904.12356. arXiv:1904.12356 [cs].

Richard E. Turner, Cristiana-Diana Diaconu, Stratis Markou, Aliaksandra Shysheya, Andrew Y. K. Foong, and Bruno Mlodozeniec. Denoising Diffusion Probabilistic Models in Six Simple Steps, February 2024. URL http://arxiv.org/abs/2402.04384. arXiv:2402.04384 [cs, stat].

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks, February 2018. URL http://arxiv.org/abs/1710.10903. arXiv:1710.10903 [cs, stat].

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Peng Cui, P. Yu, and Yanfang Ye. Heterogeneous Graph Attention Network, January 2021. URL http://arxiv.org/abs/1903.07293. arXiv:1903.07293 [cs].

Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. Scene Graph Generation by Iterative Message Passing, April 2017. URL http://arxiv.org/abs/1701.02426. arXiv:1701.02426 [cs].

Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph R-CNN for Scene Graph Generation, August 2018. URL http://arxiv.org/abs/1808.00191. arXiv:1808.00191 [cs].

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, November 2022. URL http://arxiv.org/abs/2211.11138. arXiv:2211.11138 [cs].

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding Conditional Control to Text-to-Image Diffusion Models, November 2023. URL http://arxiv.org/abs/2302.05543. arXiv:2302.05543 [cs].

Qi Zhu, Kang Yu, Hanqiao Li, Qingqing Zhang, and Dawei Tu. Rapid residual stress prediction and feedback control during fused deposition modeling of PLA. *The International Journal of Advanced Manufacturing Technology*, 118(9):3229–3240, February 2022. ISSN 1433-3015. doi: 10.1007/s00170-021-08158-0. URL https://doi.org/10.1007/s00170-021-08158-0.

Appendix

To avoid cluttering the main body of text and risk obfuscating the focus of the work, we present more detailed mathematical derivations here in the appendix.

A.1 Elucidated Diffusion

Here we present a more rigorous treatment of the elucidated framework. N.B. this is still a summarisation and interpretation of Karras et al. [2022] for reader convenience.

ODE vs SDE Formulation

The foundational differential equation which governs deterministic sampling is:

$$dx = -\dot{\sigma}(t)\sigma(t)\nabla_x \log p(x;\sigma(t))dt \qquad \text{(Eq. 1 Karras et al. [2022])}$$

Where dot denotes the time derivative and $\nabla_x \log p(x;\sigma(t))$ the score function. A forward step in this ODE pushes the sample away from the data. Conversely, a backward step pulls the sample closer to the data distribution. Due to special properties of the score function it can be evaluated through a simple expected L2 denoising error. This leads them to Eq. 2 and 3 in their paper:

$$\mathbb{E}_{y\sim p_{\text{data}}}\left[\mathbb{E}_{n\sim\mathcal{N}(0,\sigma^2 I)}\|D(y+n;\sigma)-y\|_2^2\right] \qquad \text{(Eq. 2 Karras et al. [2022])}$$

$$\nabla_x \log p(x;\sigma) = \frac{D(x;\sigma)-x}{\sigma^2} \qquad \text{(Eq. 3 Karras et al. [2022])}$$

Ultimately, given an trained denoiser $D(x; \sigma)$ the solution to Eq. 1 can now be found via traditional ODE solution methods. For example a simple numerical integration scheme such as Euler. Skipping ahead to section 4 of the paper they introduce their SDE variant, which adds two additional terms:

$$
\begin{aligned}
dx_{\pm} = & -\dot{\sigma}(t)\sigma(t)\nabla_x \log p(x; \sigma(t))\, dt \quad \underbrace{\text{probability flow ODE (Eq. 1)}} \\
& \pm \beta(t)\sigma(t)^2 \nabla_x \log p(x; \sigma(t))\, dt \quad \underbrace{\text{deterministic noise decay}} \\
& + \sqrt{2\beta(t)}\sigma(t)\, d\omega_t, \quad \underbrace{\text{noise injection}}
\end{aligned}
$$

(eq. 6 Karras et al. [2022])

They show that the SDE variant includes the ODE from Eq. 1 alongside two separate SDEs. Allowing to separate the terms in this manner allows for much greater understanding as we can examine each part in term. The ODE shapes sample trajectories such that they pass through the desired distribution p_t at time t. The additional terms comprise of a deterministic score-based denoising term and a stochastic noise injection term, these randomly explore the distribution p_t. These terms push samples towards the time-dependent marginal distribution, which is equivalent to correcting errors made in earlier time steps. Importantly, this formulation allows the authors to provide explicit hyper-parameters that control each part of the sampling process. In our work we implicitly use Eq. 6 via our algorithm 4.

Sampling Parameters

To facilitate best practices for Eq. 4 the authors introduce several new parameters: $S_{\text{noise}}, S_{\text{t max}}, S_{\text{t min}}, S_{\text{churn}}$, we evaluate each in turn. S_{noise} is simply a multiplier for the current noise level. Whereas $S_{\text{max}}, S_{\text{min}}$ determines noise 'bounds' in which stochasticity can occur i.e. noise levels where $t_i \in [S_{\text{t min}}, S_{\text{t max}}]$. For these noise levels, they introduce 'churning' which they define as adding and removing noise. They control this via the S_{churn} parameter. It further modulated by the current sampling step so that this source of randomness reduces over the sampling procedure. Finally, they clamp this parameter when selecting a new noise level by $\min\left(\frac{S_{\text{churn}}}{N}, \sqrt{2} - 1\right)$. Furthermore by setting S_{churn} to 0 the entire formulation returns back to the ODE formulation in Eq 1. In our implementation we use their default ImageNet parameters: $S_{\text{noise}} = 1.003, S_{\text{t max}} = 50, S_{\text{t min}} = 0.05, S_{\text{churn}} = 80$.

Remarks

In this work the authors unify ODE and SDE variants of diffusion. Using Eq. 6 they can provide a deeper analysis of reverse and forward processes. In turn they empirically verify and explore this design space resulting in the hyper-parameters presented in the previous section. In practice we found this implementation provided high fidelty samples with few sampling steps.

The shortcomings of this work lie in its heavy reliance on empirical verification. For example, we did not have the computing requirements to explore all the potential hyper-parameters, despite their recommendation this should be done in a case-by-case manner. Furthermore they use FID as their metric of choice for optimisation, it is unclear whether this would generalise to other metrics e.g. PSNR in the case of this project.

A.2 GNN Architecture Details

Here we list several of the most pertinent mathematical formalities of the implemented GNNs. Readers may refer to the cited papers or Pytorch Geometric Documentation for further detail. Where applicable we list them down in homogeneous form, they are convert to heterogeneous form by repeating the operator for every edge type (a.k.a meta-path).

Graph Convolution

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j \tag{1}$$

$e_{j,i}$ denotes edge weight. From Morris et al. [2021].

SAGE Convolution

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \cdot \mathrm{mean}_{j \in \mathcal{N}(i)} \mathbf{x}_j \tag{2}$$

From Hamilton et al. [2018].

Graph Attention Network (GAT)

$$e_{ij} = a(\mathbf{W}\mathbf{h}_i, \mathbf{W}\mathbf{h}_j) \tag{3}$$

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})} \tag{4}$$

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{a}^T[\mathbf{W}\mathbf{h}_i\|\mathbf{W}\mathbf{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\mathrm{LeakyReLU}\left(\vec{a}^T[\mathbf{W}\mathbf{h}_i\|\mathbf{W}\mathbf{h}_k]\right)\right)} \tag{5}$$

$$\mathbf{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\mathbf{h}_j\right) \tag{6}$$

Where $\|$ represents concatenation. From Veličković et al. [2018].

Heterogeneous Transformer

$$\mathrm{Attention}_{\mathrm{HGT}}(s,e,t) = \mathrm{Softmax}_{\forall s\in\mathcal{N}(t)}\left(\Big\|_{i\in[1,h]}\text{ATT-head}^i(s,e,t)\right) \tag{7}$$

$$\text{ATT-head}^i(s,e,t) = \left(K^i(s)\cdot W^{\mathrm{ATT}}_{\phi(e)}\cdot Q^i(t)^\top\right)\cdot\frac{\mu(\tau(s),\phi(e),\tau(t))}{\sqrt{d}} \tag{8}$$

$$K^i(s) = \text{K-Linear}^i_{\tau(s)}\left(H^{(l-1)}[s]\right) \qquad Q^i(t) = \text{Q-Linear}^i_{\tau(t)}\left(H^{(l-1)}[t]\right) \tag{9}$$

$$H^l[t] \leftarrow \mathrm{Aggregate}_{\forall s\in\mathcal{N}(t),\forall e\in E(s,t)}\left(\mathrm{Attention}(s,t)\cdot\mathrm{Message}(s)\right) \tag{10}$$

From Hu et al. [2020].

A.3 Further Implementation details

Baseline U-Net

Baseline U-Net hyper-parameters:

- Dimension multipliers of (1, 2, 4, 8)
- Number of ResNet blocks (2, 4, 8, 8)
- Attention layers on the final block, 8 attention heads

- Feed-forward multiplier of 2.0
- Hidden size of 128

Small U-Net

Small U-Net hyper-parameters:

- Dimension multipliers of (1, 2, 4, 4)
- Number of ResNet blocks (2, 2, 4, 4)
- Attention layers on the final block, 8 attention heads
- Feed-forward multiplier of 2.0
- Hidden size of 128

A.4 Model Probing

Here we present experimental probes on the conditional units of the trained model.

Learnable Render

We examine the learnable render to understand what properties it has learned. In appendix Figure 1 we compare the learnable render before and after model training. We can see that even prior to training the random initialisation encodes a representation of the flow rate in output render. However, we see this is amplified as the model learns this to be an important feature. In appendix Figure 2 we visualise the learned deep texture for samples with variable flow rate, we see a similar picture that is learns to encode flow rate, but also line activity, positional information, and indications of node counting (e,g, overlapping lines in the first column).

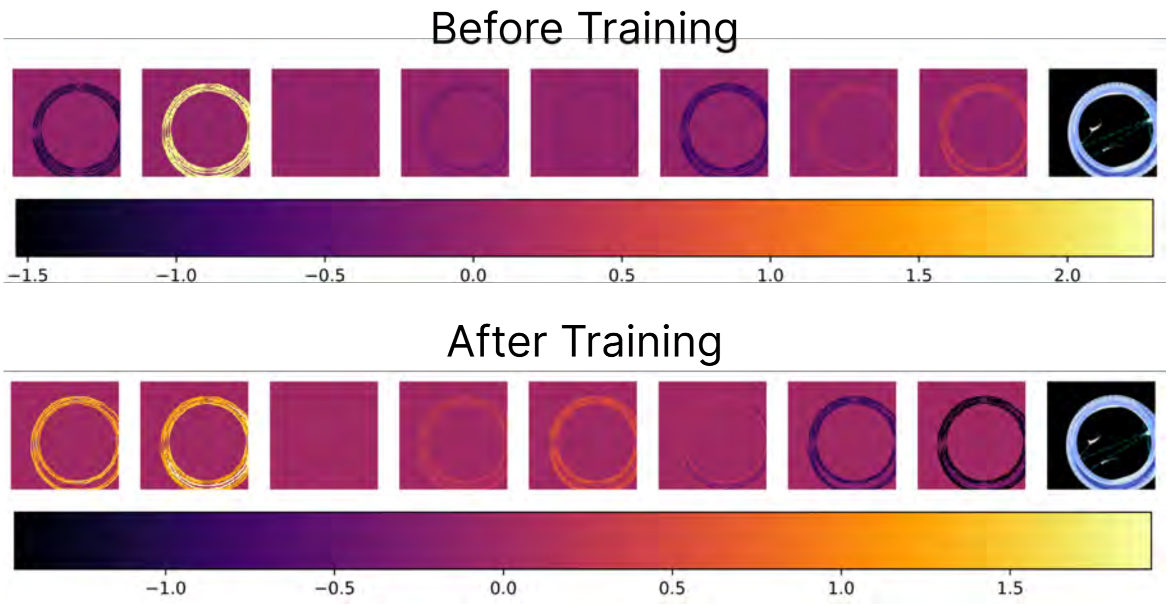

Figure 1: **Learnable Render Before and After Training.** The initial learnable render is set to D=8 channels, this is the render visualised on a per-channel for a random validation sample. The figure has been normalised by the maximum and minimum of all channels. The last column images is the target graph with colour coded flow-rate.

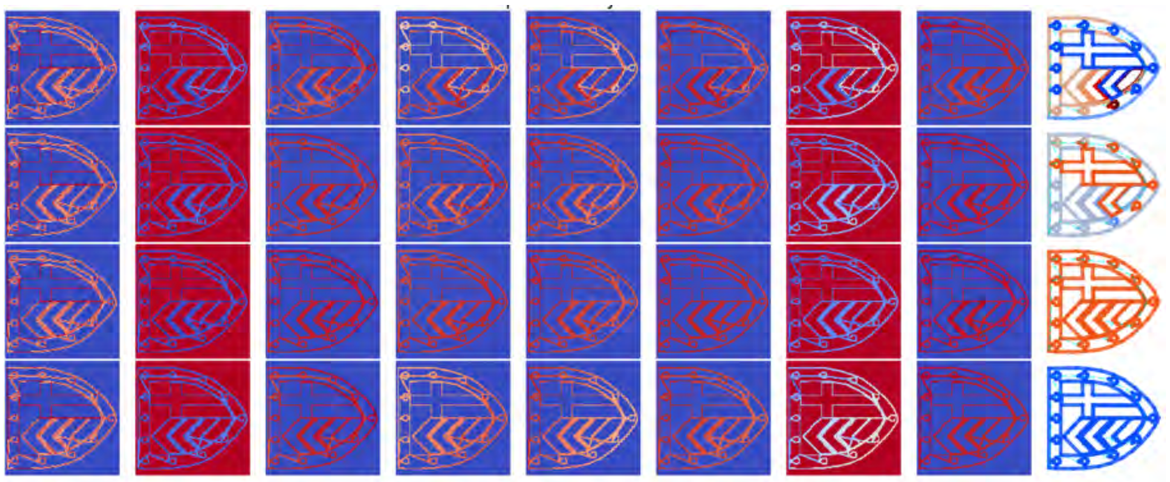

Figure 2: **Learnable Render Visualisation.** The initial learnable render is set to D=8 channels, this is the render visualised and normalised on a per-channel basis for 5 test-case samples. The last column are the target graphs with colour coded flow-rate.

Dual Graph

We examine the dual graph unit and find that the unit has a significant impact on input features. We present insights in appendix Figure 3.

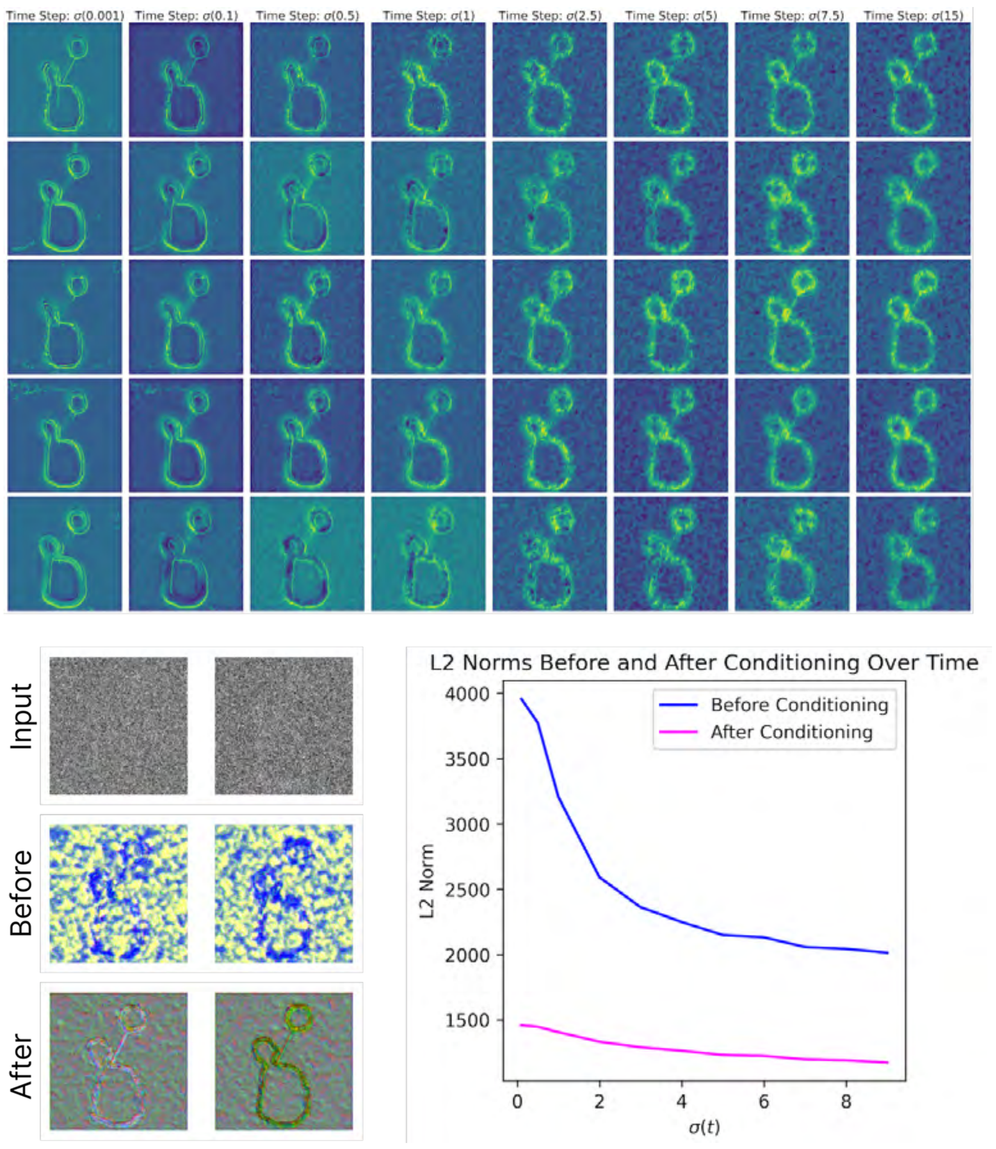

Figure 3: **Dual Graph Visualisation.** The top section visualises feature differences before and after the dual graph. Bottom left visualises specific features before and after the unit. We also plot the L2 norm of the features over diffusion time. This indicates that the GNN unit has a significant impact in feature space.

Unit Ablation

We produce samples by skipping each of the individual conditional units in appendix Figure 4. We see the samples generated without the GNN are more degraded than those with the GNN, indicating the model has higher reliance on this unit. We also conducted this visualisation on models trained only on each specific unit, however, from a qualitative point of view there wasn't major differences.

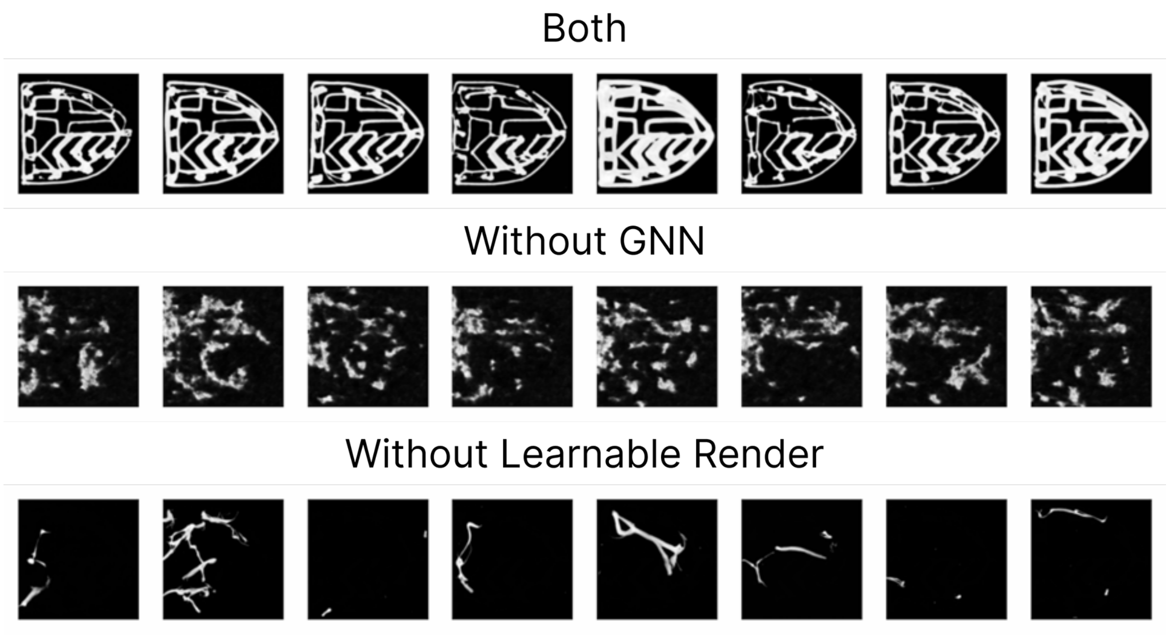

Figure 4: **Unit ablation.** Samples generated by skipping conditional units.

Examining Output Invariances

We show samples from the same input graph using the ODE sampler to examine output invariances and equivariances. We see that we get similar outputs despite the rotation, yet, they are not truly invariant to the rotation operation. This is understandable as the random seed noise isn't rotated and so we would expect different image materialisations despite the same graph.

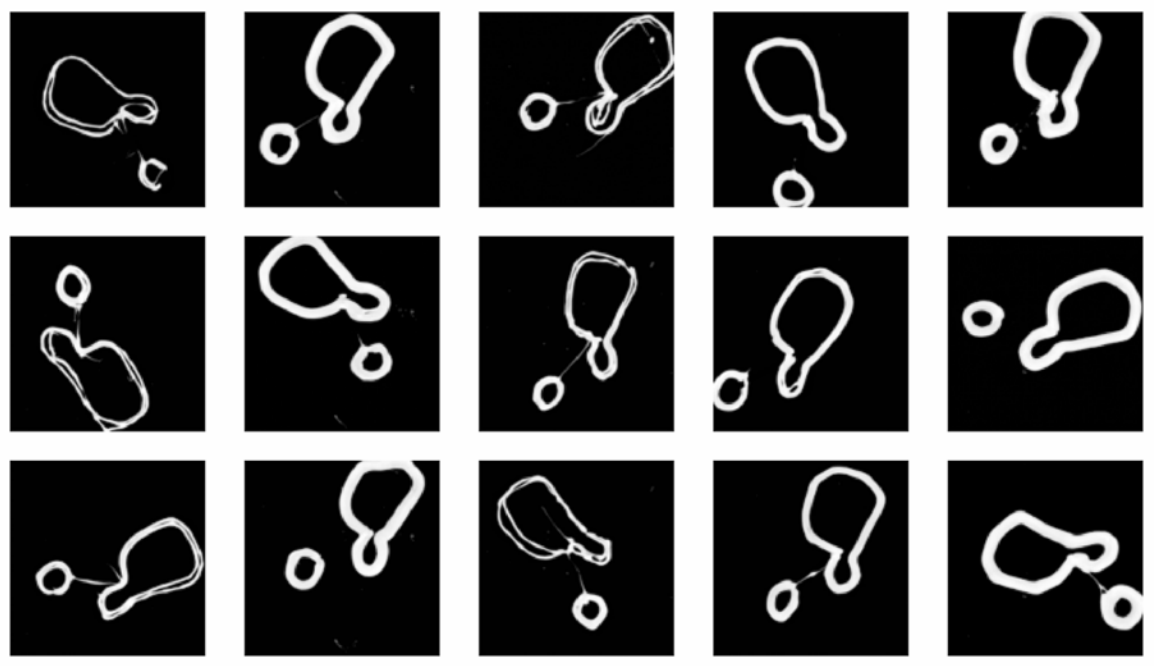

Figure 5: **Invariance Analysis.** Generated samples using with 32 sampling steps. Each row uses the same conditioning graphs from the validation split, but each row is randomly rotated.