



UNIVERSITY OF
CAMBRIDGE

Department of Engineering

Enhancing the Normalizing Flow on the Function Space

Author Name: Shuohan Tao

Supervisor: Prof. Mark Girolami

Date: 25th May, 2024

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

Signed Shuohan Tao date 25th May, 2024

Technical Milestone Report

Scale-invariant Normalizing Flow for PDE Solution Generation

Shuohan Tao, Selwyn College

Abstract

This report demonstrates the main outcomes and challenges in the project so far and outlines strategies to finish the project with the goal of designing a scale-invariant normalizing flow. Three existing approaches, including the one proposed by the project, for scale-invariant image generation were implemented, and a general pipeline was constructed to test their performance in terms of image quality and inference speed. Future work will mainly focus on improving the 2D-NeRF approach and more extensive testing on broader datasets.

1 Introduction

A project was proposed to design a scale-invariant normalizing flow for PDE solution generation. The goal is to generate coherent PDE solution, can be considered as an image, at different resolutions. Normalizing flow, a kind of image generative model, is unique in its ability to provide likelihood of generated samples, serving as uncertainty quantification in fields like fluid simulation. Due to the similarity between image and 2D-PDE solutions, using normalizing flow as PDE solvers would be reasonable. Two methods were suggested. The first approach borrowed the idea of super-resolution from the field of image generation and tries to generate and super-resolute images, where model infers and fills in finer details to upsample a low-resolution image, to get higher resolution images. The second approach learns the distribution of a discrete representation of the continuous functions by trying to generate the parameters of 2D-NeRFs [4], a kind of theoretically omni-function-approximator. Normalizing flow has, however, hindered the progress of the second approach due to the its constraint on using complex models. The first choice has been shown to be promising. In the meantime, a unified pipeline has been constructed to allow for training, testing, and comparing different models. The benchmark results for three models will be demonstrated here, including two other mdoels: generative adversarial neural operator (GANO) [5] and spectral neural operator (SNO) [3] incorporated with residual flow [1]. All models are trained and tested on MNIST dataset, image of digits that are more complex than PDE solutions, for comparing with popular image generation models. The three proposed architectures, coupling autoregressive flow, spectral neural operator residual flow, and 2D-NeRF flow have their structures plotted in figure 7.

2 Proposed Approaches

2.1 Coupling Autoregressive Flow (CAF)

Borrowing the idea of super-resolution, our proposed method first generates a fixed resolution image with normalizing flow and uses a second model to autoregressively, where all previous outputs add up to become the new input, upsample the image. This autoregressive nature allows it to generate images of arbitrary resolution above the base resolution, differing it from a previous study [6] that used dedicated models for each different resolution and thus only allows a fixed set of resolutions. CAF first models a fixed resolution image autoregressively generates higher frequency components

conditioned on lower resolution images. A significant improvement was made since last presentation to scale each of the higher resolution components by an exponentially decaying value and have achieved good result, which will be discussed in the ablation study section. Both the base model and the autoregressive model are structured with coupling flow, a kind of normalizing flow that’s fast but is constrained in the transformations allowed to be used.

2.2 Spectral Neural Operator Residual Flow (SNORF)

Spectral neural operator maps between functions by transforming the function’s mapping on a finite number of orthonormal basis, a finite sequence. Two choices of basis functions are Chebyshev polynomials and Fourier basis functions. Demonstrated in the original paper [3], Fourier basis functions generally outperform Chebyshev polynomials and thus are adopted here. It was suggested by an advisor to incorporate this module with residual flow [1], which allows relatively free-form expression on normalizing flows.

2.3 2D-NeRF Flow

Inspired by [2], learning the distribution of models that can generate continuous functions would be a good idea. Each 2D-NeRF outputs pixel values of a unique image given coordinate of those pixels, and thus learning the distribution of the 2D-NeRFs would allow for the generation of discretization-invariant images, in addition to scale-invariance. However, the paper used adversarial training, which is significantly more powerful than normalizing flows, thus the poor performance on the generated images in figure 2 can be expected. But this approach still worths studying since adversarial training doesn’t provide likelihood. Another study in the field of NeRF, originally designed to reconstruct 3D scenes, called Plenoxel [7] has revealed that the neural network in NeRF can actually be substitute with simpler spherical harmonics maintaining the performance. Adapting the idea to 2D images might make 2D-NeRF flow possible due to the decrease in dimension of density to be modelled.

3 Benchmark and Ablation Study

A pipeline was developed for benchmarking the models in terms of inference speed and image quality. All models are trained on 28 by 28 images and are sampled for images at scales 28, 32, 36, and 40, 4 images at each scale, shown in table 2. Their inference speed at different scales are tested in terms of number of function evaluations (NFE), shown in figure 4. The total number of parameters are also shown in table 1. A kernel-based metric called maximum mean discrepancy (MMD, the lower the better) is used to measure the distance between the learned distribution of the model and the distribution of the test dataset to test image quality. All test dataset are downsampled from highest test resolution to maintain higher frequency information. In figure 6, MMD scores of the three models are plotted against resolutions. CAF has a relatively constant and low MMD score across all resolution levels, the best of all three models. GANO has an oscillating and increasing MMD score as resolution increases and generates blurry image for higher resolutions. SNORF has a constantly increasing MMD score as resolution increases. Inference speed are also tested in terms of NFE (smaller the better) and time. Time taken for inference is non-deterministic and so is only for reference.

Table 1: Parameter counts

	CAF	GANO	SNORF
Total Number of Parameters	40015716	4904578	41524864

Shown in figure 1a, figure generated by model without scaling frequency components is noisy. When training, the model learns how to infer higher frequency components given lower frequency information of the training data. The issue is that even though the model can infer detailed parts (higher frequency components) of an image, it sometimes creates these details poorly. This leads to errors when it tries to add more details to an already flawed low-resolution image. Ignoring these lower quality details (suppressing previously generated components) could help the model focus on improving the base 16x16 image, which is reliably created by a different model. It can be argued that suppressing high frequency components is like adding zeros in the frequency domain of an image (sinc upsample). However, unlike shown in figure 1b, where a 28x28 image is sinc upsampled with visible ripple artefact around the edges, called Gibbs phenomenon, high fidelity is retained in generated images. For very high resolutions, like 64x64 shown in figure 1d, images display artefact similar to zero padding. This occurs because high frequency components scales to almost zero, similar to direct zero padding. An additional study, shown in figure 1c, used e^{-f} for padding, where f is the frequency. Artefact persisted, demonstrating the super-resolution model’s effectiveness.

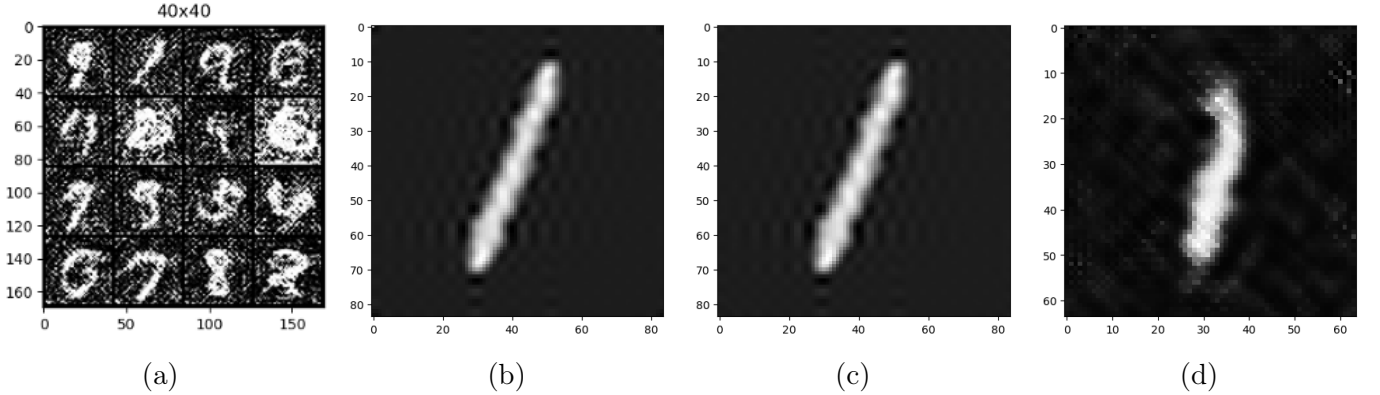


Figure 1: (a) Noisy image; (b) Zero-padded; (c) Exponentially padded; (d) Artefact;

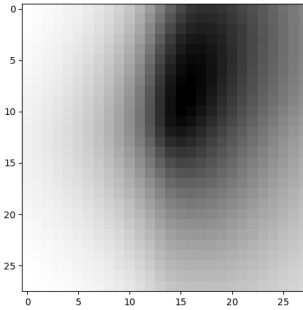


Figure 2: 2D-NeRF flow generated image

4 Advantages and Potential Improvements

The proposed CAF is scalable and fast. Since the model learns upsampling mapping and fixed-resolution image generation at the same time, its performance can be improved by changing the part for the second task alone. In addition, the NFE of the model scales linearly with the desired resolution starting from 1 corresponding to a 16 by 16 image, making it faster than residual flow, which relies on numerical simulations.

However, the existing artefact in very high resolution still needs to be solved. Currently the exponentially decaying function takes the form $(1 - \epsilon)a^{-\lambda(f-18)} + \epsilon$, where f is the resolution and ϵ , λ , and a are manually tuned. The set of parameters used for generating all images demonstrated are $\epsilon = 0.01$, $\lambda = 0.5$, and $a = 2$. Two possible improvements could be studied. The first one is using a neural network to define the scaling function, and the second one is using MMD score to optimize the three proposed parameters.

5 Summary and Further Studies

In summary, though the two derivative studies on SNORF and 2D-NeRF flow have encountered difficulty, the main study on CAF has achieved good result and is promising for generating scale-invariant PDE solutions, surpassed the performance of an existing study [6] by allowing generation of arbitrary resolution image, a pipeline has also been constructed that can accelerate future studies. In addition to CAF, the other two proposed models can be further studied. For SNORF, it is worth studying more complicated transformation than point-wise operations on higher frequency information. For 2D-NeRF flow, incorporating continuous normalizing flow, a more powerful and free-form structure, with Plenoxel could potentially solve the current issue and thus worths further study. After these studies, a further benchmark could be done on PDE dataset to test these model's performance on generating PDE solutions. A detailed Gantt chart for plans of the project is provided in figure 3.

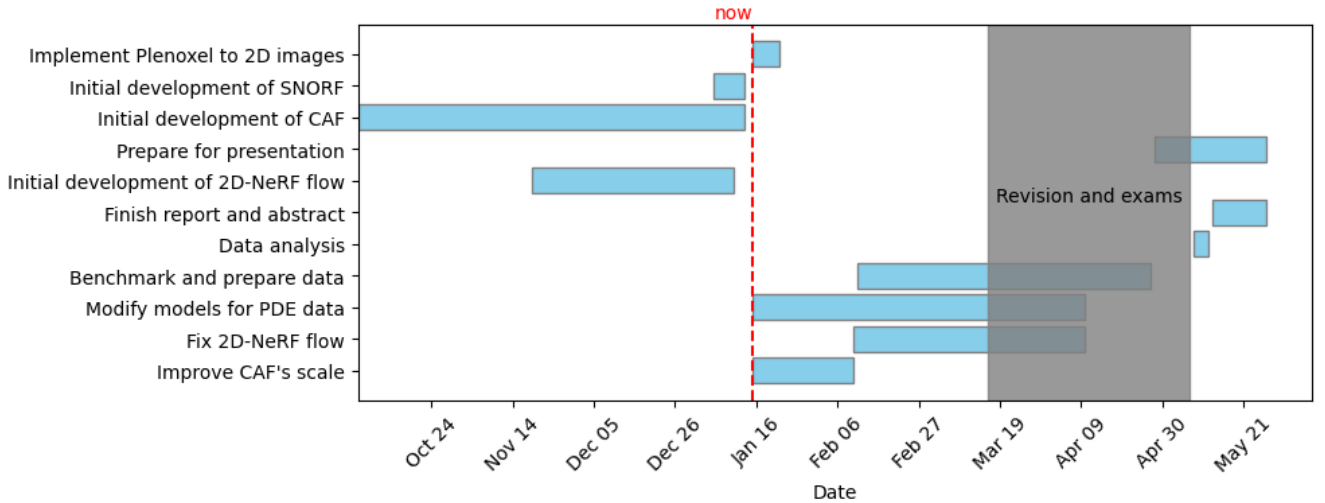


Figure 3: Gantt chart for future plan

Appendix

Table 2: Comparison of samples from three models

CAF		GANO		SNORF	
28x28		32x32		28x28	
32x32		32x32		32x32	
36x36		36x36		36x36	
40x40		40x40		40x40	

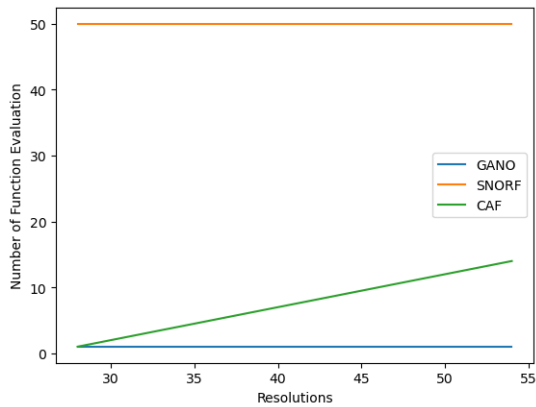


Figure 4: Number of function evaluations

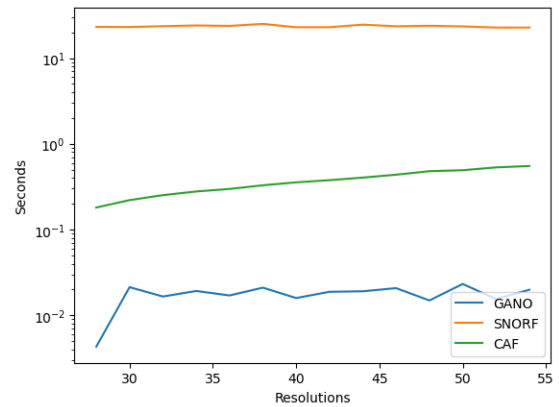


Figure 5: Time taken for sampling 4 images

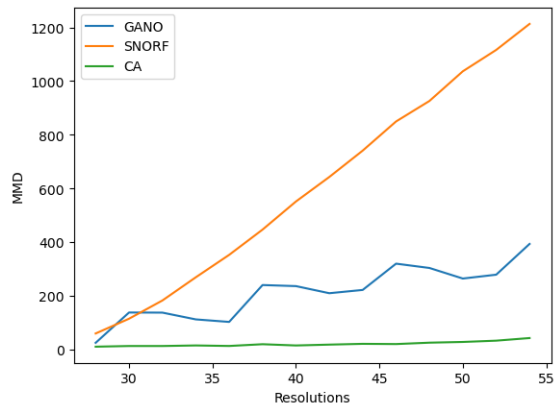


Figure 6: MMD score at different resolutions

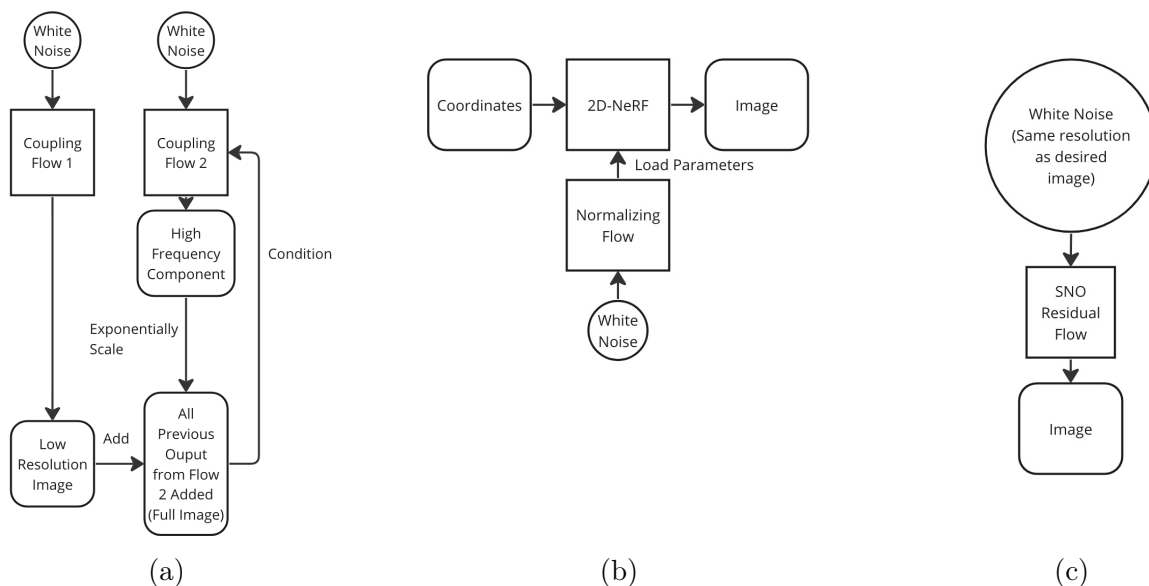


Figure 7: Architectures: (a) CAF; (b) 2D-NeRF flow; (c) SNORF;

References

- [1] R. T. Q. Chen, J. Behrmann, D. Duvenaud, and J.-H. Jacobsen. Residual flows for invertible generative modeling, 2020.
- [2] E. Dupont, Y. W. Teh, and A. Doucet. Generative models as distributions of functions, 2022.
- [3] V. Fanaskov and I. Oseledets. Spectral neural operators, 2022.
- [4] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.
- [5] M. A. Rahman, M. A. Florez, A. Anandkumar, Z. E. Ross, and K. Azizzadenesheli. Generative adversarial neural operators, 2022.
- [6] V. Voleti, C. Finlay, A. Oberman, and C. Pal. Multi-resolution continuous normalizing flows, 2021.
- [7] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks, 2021.

Technical Abstract

Enhancing the Normalizing Flow on the Function Space

Supervised by Prof. Mark Girolami

Shuohan Tao, Selwyn College

May 2024

1 Overview

This report focuses on developing novel methods to enhance normalizing flow in function space to generate continuous, arbitrary-resolution image-like data that can serve as PDE solutions. The aim is to enable normalizing flow models to generate data suitable for partial differential equation (PDE) solutions. This enhancement allows for accurate uncertainty quantification due to the inherent likelihood estimation provided by normalizing flows.

2 Problem Statement

The primary challenge addressed in this project is the generation of continuous data across arbitrary resolutions using normalizing flows. This involves overcoming the limitations of traditional normalizing flow methods which are often restricted to fixed-resolution outputs. By generating data that is continuous and scalable, the enhanced normalizing flow model can be effectively used for generating solutions to PDEs, which are critical in various scientific and engineering applications.

3 Proposed Methods

In total 4 methods are proposed to solve the problem stated.

3.1 Pixel Patch Transformer (PPT)

This model is based on an encoder-decoder transformer. Images are first segmented into fixed patches, then pixels in each patch and its neighboring patches are concatenated to serve as context information which is input to the transformer encoder. The high

resolution grid coordinates is used as query value to input to the transformer decoder, and the transformer will output the corresponding higher resolution image.

3.2 Spectral Neural Operator Residual Flow (SNORF)

This method incorporates FNO, which is a kind of discretization-invariant mapping, and residual flow to directly generate images at arbitrary resolution. The base distribution is modified to Gaussian random field to match the rationale of mapping between functions.

3.3 Coupling Autoregressive Flow (CAF)

This approach first generates a fixed-resolution image with a coupling flow and then autoregressively generates higher resolution images conditioned on lower resolution ones using a separate conditional normalizing flow.

3.4 Implicit Neural Representation Incorporated with Coupling Flow

This approach was inspired by "generative models as distributions of functions". It learns to learn the distribution of neural representations of the images, which are inherently omni-function approximators, and thus theoretically can learn any elements in the function space.

4 Main Result

The main outcomes of the study are that several architectures of scale-invariant normalizing flow are proposed, and conclusions are drawn on why some of them works while others don't. Combining these conclusions, the best performing model, PPT, is designed and has shown expressive super-resolution ability, capable of upsampling 16 by 16 images of digits to resolutions of 2048 by 2048. In addition, this model is mesh-free, meaning it can generate PDE solution at arbitrary coordinate, making it a powerful PDE solution generator.

Enhancing the Normalizing Flow on the Function Space

Supervised by Prof. Mark Girolami
Shuohan Tao, Selwyn College

May 2024

Contents

1	Introduction	1
2	Background and Literature Review	2
2.1	Fourier Neural Operator	3
2.2	Normalizing Flow	5
2.3	Transformers	7
2.4	Pixel Transformer	8
3	Proposed Methods	9
3.1	Pixel Patch Transformer (PPT)	9
3.2	Coupling Autoregressive Flow (CAF)	11
3.3	Spectral Neural Operator Residual Flow (SNORF)	12
3.4	Implicit Neural Representation Flow	12
4	Experiment Results	14
4.1	Metrics	14
4.2	Datasets	16
4.3	Methods	16
4.4	MNIST Result	16
4.5	Ablation Study	17
4.6	Darcy Flow Result	18
5	Analysis	20
5.1	PPT	20
5.2	CAF	22
5.3	SNORF	23
6	Future Studies	24
7	Conclusion	25
	References	26
	Appendix A: Pixel VAE	29
	Appendix B: Pixel Fourier Transformer Flow (PFTF)	30
	Appendix C: Patch Transformer	33
	Appendix D: Previous Results	34

List of Figures

1	Architecture of FNO. Figure adapted from [1].	3
2	(a): Coupling transformation. (b): Checkerboard pattern, white and black parts are transformed in alternating iterations. Figures adapted from [2].	5
3	(a): Architecture of PPT. (b): Details of "Organizing Patches" Block. .	10
4	Architecture of CAF.	11
5	Architecture of Implicit Neural Representation Flow	13
6	PPT's generated samples across different resolutions.	17
7	(a): 32 by 32 image of PPT trained with neighboring patches. (b): 64 by 64 image of PPT trained with neighboring patches. (c): 32 by 32 image of PPT trained without neighboring patches. (d): 64 by 64 image of PPT trained without neighboring patches.	18
8	(a): MMD score against resolutions. (b): PPT curve from figure (a) zoomed in.	19
9	Mean squared error of PPT on resolutions of 16, 20, 24, 28, 32, 48, 64, 128, 256, and 512.	19
10	Illustration that some of the queried points require extrapolation tasks.	21
11	Poor quality generated samples from CAF.	22
12	Generated sample of SNORF trained on Darcy flow dataset.	23
13	Generated images of Pixel VAE using 64 randomly sampled pixels as context information.	29
14	Sample image from PFTF trained on 32 by 32 images.	30
15	Histogram of frequency bins of an FFT transformed image.	31
16	(a): PFTF's predicted spectrum. (b): Ground-truth spectrum.	32
17	Mutual information between frequency components from 16 to 32 and the frequency component at 16.	32
18	Generated images of patch transformer trained on digits of zero.	33
19	High frequency artefact in raw outputs of CAF.	34
20	MMD scores of CAF, SNORF, and GANO on MNIST dataset.	35
21	Generated samples of CAF trained on MNIST.	35
22	Generated samples of SNORF trained on MNIST.	35

1 Introduction

Solving PDEs have always been a popular and difficult research topic. Traditional ways of solving PDEs based on numerical methods are slow, though they usually come with bounded errors. These methods usually start from an initial condition, simulating the field’s evolution in time to get the PDE solution at a particular interested time or spatial location. These methods scale poorly as dimensionality increases, with computational cost scales exponentially as dimensionality increases [3]. More efficient ways of solving PDEs are thus of particular interest.

Deep generative models, as a topic in machine learning that has recently achieved great progress, learning data distributions and synthesizing new data. State of the art models, like DDPM (Denoising Diffusion Probabilistic Model) [4], has shown extremely impressive ability to model very complex and high dimensional distributions like images of celebrity faces with high quality. Generally, the data modelled by this kind of models can range from graph to sequences, among which the most well-studied ones are images and natural language. Several families of generative models exist to model the distribution of images, which has great similarity to PDE solutions. This is because images can be considered as pixel intensity fields sampled at fixed grid positions defined by the images’ resolutions, just as PDE solutions whose values are presented on coordinates of interest. Therefore, using image generative models to model PDE solutions would be a good idea, and there are several advantages of doing so.

First, some families of generative models can generates samples very fast. Due to the GPU’s capability of running multiple calculations in parallel, cecertain types of generative models like GANs (Generative Adversarial Networks) [5] and VAEs (Variational Autoencoder) [6] can generate batched samples in a single forward pass through the neural network without any numerical methods involved. Second, most deep generative models allows for conditional generation, meaning that the initial conditions and time steps can both be considered when generating the solutions. Most of the image generative models use unit Gaussian distributed random samples as latent variable and map them to image samples. A separate model can be trained to map from the initial conditions and queried timesteps to the latent variable deterministically, which is then mapped to the PDE solutions corresponding to those initial conditions and timesteps.

There also exists several challenges, however. The first one is that deep models, especially generative models, require large amount of data to achieve good performance. With PDEs, however, data isn’t a problem. Synthetic data can be solved with traditional PDE solvers, providing almost limitless number of training samples.

The second challenge is that, as an application with use cases that requires high

accuracy, e.g., fluid simulations, most generative models don't provide measure of uncertainty to allow the user to balance the risk of adopting a certain PDE solution. Nevertheless, a family of generative models called normalizing flows [7], which explicitly provides likelihood of data samples. They are trained to maximize the likelihood of data samples, and thus likelihood provided by these models can be leveraged to provide a natural way of calculating uncertainties.

The third challenge is that numerical methods for solving PDEs can produce solutions at arbitrarily fine grid, some might not even need to be on the grid, but almost all of the mainstream generative models work only at one or several fixed resolutions. This poses a serious challenge of using image generative models to solve PDEs.

Another family of deep models aiming at solving PDEs, PINNs (physics informed neural networks) [8], have been proven to be practically useful and mesh-free. This kind of model suffers from the lack of use of spatial local correlations of PDE data as they only map pointwisely from coordinates to PDE values by training a model to directly regressing on the differentiated quantity, whereas in conventional deep learning, spatial information are processed by operations like convolutions to capture the local dependencies. This makes training quite lengthy as the regression task is much harder without considering correlations between inputs. A more recent study proposed a model called FNO (Fourier Neural Operator) [1] that approximately maps between features continuously. It works in the frequency domain to provide discretization invariant mapping.

In this article, several new architectures will be proposed to try to solve each of these shortcomings of existing models. Their own limitations will also be studied to serve as guidance for the design of a final better model, which is capable of generating natural and smooth PDE solutions with likelihood for all resolutions below the training set and generalizing well to reasonably higher resolutions than the training dataset.

2 Background and Literature Review

PDE solutions can be considered as deterministic time-dependent mappings between the initial conditions and solutions. Traditional numerical methods for solving PDEs, including finite difference methods, finite element methods, and spectral methods, have been the standard approach for many decades. These methods rely on simulating the physical process represented by the PDEs directly, and are quite computation heavy. They also scale poorly to the increase in dimensionality as the solution value at each of the coordinates of interests is simulated separately. This leads to the motivation to use generative models to solve PDEs.

A family of generative model that could potentially solve most of the existing disadvantages of traditional numerical methods is normalizing flow [7]. This kind of model

is trained to maximize likelihood of observed data and thus can provide likelihood as uncertainty quantification, it generates data like images or PDE solutions very fast in a single forward pass, and it allows for conditional generation by training a separate model. The only shortcoming of it is that it generates data on a fixed grid due to its use of traditional deep models like CNNs.

Another family of deep models that avoid such a problem is PINN [8]. This type of model solves the PDEs in a way very similar to the traditional numerical methods. It simulates the physical process as well, but only that it uses a deep model to directly regress on a physical quantity of interest. That model is then substituted into the PDE equation to simulate in time and spatial domain. Training dataset includes the ground-truth PDE value at each of these time steps, and thus an error could be calculated at each of these timesteps using mean-squared error as shown in Eq. 1.

$$\frac{1}{N} \sum_{n=1}^N \left(f(x_n, y_n, t_n) - \hat{f}(x_n, y_n, t_n) \right)^2 \quad (1)$$

$\hat{f}$ represents the quantity output by the neural network, N is the total number of samples, and x_n, y_n, t_n and $f(x_n, y_n, t_n)$ are provided by the dataset. The model can then be trained to minimize this error such that it directly simulates the ground truth function f . Since neural networks are essentially continuous functions, they can then generate solution values at unseen coordinates and timesteps. However, since the function learns pointwise mapping, considering any individual data point as training samples, it is very slow to train or sample.

2.1 Fourier Neural Operator

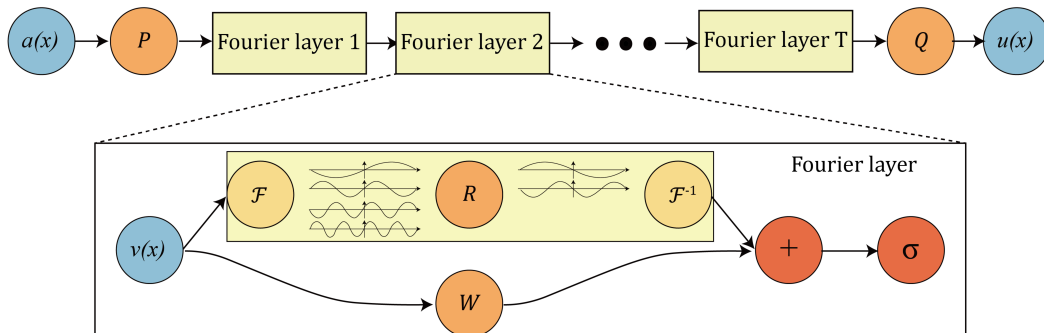


Figure 1: Architecture of FNO. Figure adapted from [1].

A recent study to learn efficient discretization invariant mapping is FNO [1]. Its architecture is shown in figure 1. In each of the fourier layers shown, the input $v(x)$

goes through two transformations in parallel. The first transformation happens in the frequency domain. Input is Fourier transformed, all frequency components higher than a fixed value is zeroed, and lower frequencies are elementwise multiplied by a learnable weight R . The resulting value is inverse Fourier transformed back into the spatial domain. The second transformation happens in the spatial domain. The input is directly multiplied by the same learnable weight elementwisely, which is essentially a 1-by-1 convolution. The output from both transformations are added together and passed through an activation function. The raw input $a(x)$ is first elementwise multiplied by a learnable weight P , which keeps the spatial dimension of the input but lifts the channel dimension of the input to a higher value. The output of the last Fourier layer is elementwise multiplied by a learnable weight Q , which reverses the effect of P to bring the channel dimension back to the same as the original input. According to [9], choosing a proper fixed number of lowest frequencies to transform in the frequency domain is critical. Using too low value when major high frequency information exists would hurt generalization, as they are omitted in the frequency domain transformation and only considered in the spatial domain transformation.

Although with the weaknesses mentioned above, it generally performs well in mapping between continuous values. Therefore, studying its effectiveness in mapping from random functions to data distributions would be desirable for discretization invariant generative models. To work in the function space, the base independent Gaussian distribution used by most generative models needs to be substituted with random functions. One of such kind of well-defined random functions is Gaussian random field. They are essentially multidimensional Gaussian processes with finite trace in infinite dimensional space, which is crucial for discretization invariance as it makes the discretized function well-defined as the grid gets finer and finer. There are relevant work to incorporate almost all of the existing mainstream generative models with FNOs. For GANs [5], there is GANO [10], and for score-based diffusion models, there is multi-level diffusion [11]. GANs are a type of generative models that adopts adversarial training, where a model is trained to generate synthetic samples, and a separate model is trained to distinguish between the synthesized samples and true samples. Score-based diffusion models are a type of generative models that learns the gradient of data distribution and generates samples using Langevin dynamics.

Though discretization-invariant, they have generally failed to model complex spatial data like natural images, which contains much more high frequency components than most PDEs. Given the limitations of existing models, it is desirable to design a model that generates samples fast, provides uncertainty quantifications, and is discretization-invariant. Ideally, it also needs to be mesh-free, meaning that it can generate values on unstructured grids. A successful model having these properties is of great commercial and academical value to relevant areas in fluid dynamics, material science, and various

other scientific fields.

2.2 Normalizing Flow

Normalizing flow [7] is a family of model that maps from a base simple distribution to the data distribution using invertible transformations. These invertible transformations allow the calculation of exact likelihood values of data samples through the following change of variable formula.

$$p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{X}}(\mathbf{g}^{-1}(\mathbf{y})) \left| \det \left(\frac{\partial \mathbf{g}^{-1}}{\partial \mathbf{y}} \right) \right| \quad (2)$$

$p_{\mathbf{Y}}(\mathbf{y})$ is the data distribution, $p_{\mathbf{X}}(\mathbf{x})$ is the base simple distribution usually chosen to be unit Gaussian, $\mathbf{g}$ is an invertible transformation parameterized by neural networks, and the derivative $\frac{\partial \mathbf{g}^{-1}}{\partial \mathbf{y}}$ simply equals the Jacobian of the invertible transformation. In practice, all these values would be taken log such that the formula becomes the following form.

$$\log(p_{\mathbf{Y}}(\mathbf{y})) = \log(p_{\mathbf{X}}(\mathbf{g}^{-1}(\mathbf{y}))) + \log \left| \det \left(\frac{\partial \mathbf{g}^{-1}}{\partial \mathbf{y}} \right) \right| \quad (3)$$

The limitation of using invertible transformations is so that the determinant Jacobian term is non-zero and the volume is thus preserved. Therefore, different to the traditional sense of "likelihood", the likelihood values provided by normalizing flows are valid probabilities as they are normalized, making it possible to compare between different models' performances using the likelihood. The only part involving neural networks is the invertible transformation, and thus the main topic of study is how to design this invertible transformation.

One common way of parameterizing the transformation is through coupling layers, as



Figure 2: (a): Coupling transformation. (b): Checkerboard pattern, white and black parts are transformed in alternating iterations. Figures adapted from [2].

proposed by RealNVP [2]. Coupling layers define invertible transformations through

affine transformations on alternating parts of the input.

$$\mathbf{s}, \mathbf{t} = f(\mathbf{x}_1) \quad (4)$$

$$\mathbf{y}_2 = \mathbf{s} \odot \mathbf{x}_2 + \mathbf{t} \quad (5)$$

f represents the neural network, $\mathbf{x}_1$ and $\mathbf{x}_2$ represent two parts of the input which combines to form the whole input, and arbitrary ways to segment the input into two parts would work but the usual approach would be to segment input using checker-board patterns as shown in figure 2b adapted from the original paper [2]. Since one part of the input that decides the transformation undergone by the other part is unchanged after the transformation, it can easily be inverted by putting the unchanged part through the neural network to get scale and shift values and invert the changed part as follows.

$$\mathbf{x}_2 = (\mathbf{y}_2 - \mathbf{t}) \odot \mathbf{s}^{-1} \quad (6)$$

This can be clearly understood from figure 2a adapted from the original paper [2]. Multiple such layers could be stacked together to form more powerful transformations. Though with universality proof on log-concave distributions [12], it generally lacks expressiveness and has worse performance than other generative models when generating images. It was also recently shown in [13] that the model isn't capable of learning true multi-modal distributions, all it's doing is shifting regions of the base Gaussian distribution around without compressing or inflating the space. In addition, since the segmentation of the input happens in the spatial domain, FNO, which works in the frequency domain, is inherently not suitable for working with coupling layers. To allow the model to use broader families of transformations other than affine transformations, residual flow [14] was proposed. This type of flow relies on Banach fixed-point theorem to define invertible transformations [15]. The transformation is defined as follows,

$$\mathbf{y} = \mathbf{x} + g(\mathbf{x}), \quad (7)$$

where g is simply a neural network. According to Banach fixed-point theorem, the transformation can be easily inverted using numerical methods if the neural network g is contractive, which means its Lipschitz constant is smaller than 1. To invert it, use the following numerical method.

$$\mathbf{x}_{t+1} = \mathbf{y} - g(\mathbf{x}_t) \quad (8)$$

$\mathbf{x}_0$ is initialized to $\mathbf{y}$ and

$$\lim_{t \rightarrow \infty} \mathbf{x}_t = \mathbf{x}. \quad (9)$$

This allows the model to use relatively free-form transformations, any neural network that has Lipschitz constant smaller than 1, makes incorporating FNO with normalizing flows possible.

In addition, the training of normalizing flow has long been considered unstable. Glow [16] proposed to use activation normalization and invertible 1-by-1 convolutions to stabilize the training process. Activation normalization are flow layers that scales and shift the image along each channel using learnable weights. It helps to bring the range of the flow to the desired value. Invertible 1-by-1 convolution is used to help different channels communicate information. Though we’re working with data with a single channel (grayscale), RealNVP proposed to use multiscale architecture, which segment the images into patches and concatenate them along a new dimension, which is considered as a channel dimension. This renders the importance of communication between different channels. Invertible 1-by-1 convolution uses a smart parameterization shown in Eq. 10 to make the transformation invertible and have easily tractable determinant Jacobian.

$$\mathbf{W} = \mathbf{P}\mathbf{L}(\mathbf{U} + \text{diag}(\mathbf{s})) \quad (10)$$

$\mathbf{W}$ is the weight of the convolution kernel, $\mathbf{P}$ is a fixed permutation matrix, $\mathbf{L}$ is a learnable lower triangular matrix with diagonal elements one, $\mathbf{U}$ is a learnable upper triangular matrix with diagonal elements zero, and $\mathbf{s}$ is a learnable vector. This formulation means the determinant of $\mathbf{W}$ equals:

$$\det(\mathbf{W}) = \det(\mathbf{P}) \cdot \det(\mathbf{L}) \cdot \det(\mathbf{U} + \text{diag}(\mathbf{s})) \quad (11)$$

$$= 1 \times 1 \times \prod_i \text{diag}(\mathbf{s})_i \quad (12)$$

$$\log |\det(\mathbf{W})| = \sum_i \log |\mathbf{s}_i| \quad (13)$$

This guarantees invertability as long as element in $\mathbf{s}$ is non-zero. In general, activation normalization is equivalent to an elementwise multiplication along the channel dimension followed by a vector addition, while invertible 1-by-1 convolution is equivalent to a matrix product along the channel dimension followed by a vector addition.

2.3 Transformers

Transformers [17] are a type of model that handles sequence input. It is very powerful due to its attention mechanism, allowing the model to use different transformations on the input based on input values. Different from the usual sequence models like RNNs (recurrent neural networks) [18] and LSTMs (long short term memory) [19], it processes the input in parallel and is permutation equivariant, meaning that you can permute the order of your input tokens in the sequence and the output will

be permuted in the same order. Positional encoding was introduced to make the model aware of the position of the tokens in the sequence by appending a certain representation of the position of the token to the token value itself. PDE solutions, compared to image data, are actually more similar to unordered data. A single PDE solution can be sampled at arbitrarily many coordinates, that don't have to be on the grid, making it more similar to sequences, which can also be of varying input length. If we consider each pixel, or sampled coordinate of the PDE solution, as an element in the sequence, directly attaching the coordinates as positional encodings would make the sequence a unique representation of the PDE solution at the sampled coordinates. This makes it interesting to use transformers to model the PDE solutions. There are many types of transformers, and what's adopted here is called the encoder-decoder transformer as proposed in [17]. It can be considered as two separate transformers, one encodes the context information provided to pass to the decoder. The decoder processes the encoded context information, together with the additional query value to generate the corresponding output to the query value.

2.4 Pixel Transformer

Pixel transformer [20] was proposed to learn to infer image pixel values given part of the ground-truth pixel values of the image. The model processes the image by considering each pixel as an element in the sequence. It randomly sample a fixed number of pixels in an image to serve as context information. The coordinates of the remaining pixels are then queried to make the model output the parameters that define an MoG (Mixture of Gaussian). The expected value of the MoG is then used as the pixel intensity at that coordinate. The model turns out to work pretty well on modelling 2D image, 3D shapes, and video data. There are, however, several main drawbacks of their proposed method. First, their model didn't use the true MoG, instead, only the Gaussian with the largest inferred component prior is considered, and the "expected value" is simply the mean of that Gaussian instead of the whole MoG. In addition, in total 255 Gaussians are used in the paper to be able to capture multi-modal distributions since there are in total 255 discrete different values for the intensity of a pixel. However, since we want to simulate PDE solutions, which is continuous, a finite number of bins wouldn't be suitable. In addition, randomly sampling pixels, using them as context information, and query the pixel values at the remaining coordinates can be quite memory inefficient. To infer a specific pixel, the information provided by pixels far from it is almost non-existent, and having the model calculating the attention score between the two pixels is quite inefficient and unnecessary.

3 Proposed Methods

Several methods are proposed to generate discretization-invariant solutions to PDEs. In general, there are two approaches: directly learning the functional distribution of the data and learning to super-resolute a low resolution solution. Two additional models were also proposed along with the study, which helped with the design of the main proposed models.

3.1 Pixel Patch Transformer (PPT)

Pixel patch transformer is directly inspired by Pixel Transformer. It is also influenced by ViT(Vision Transformer) [21] and MAE(Masked Autoencoder) [22], where images are segmented into patches to serve as input to transformers.

This model works solely for super-resolution task. It segments a lower resolution image into patches and predicts the higher resolution version of each of the patches, as shown in figure 3a. It does so by using the corresponding lower resolution patch of the queried patch position, together with its 4 neighboring patches (upper, lower, left, and right), as context information to form some kind of "context convolution", allowing the model to process local informations. Illustrated in figure 3b, the input image is segmented into a fixed number of patches (16 in the experiments done), whatever the resolution is. A grid will be created at the target high resolution, which is passed through a learnable linear transformation to get the coordinate embedding. The embedding is similarly segmented into 16 patches. At inference time, to generate the high resolution patch corresponding to patch number 6 in figure 3b, its 4 neighboring patches are going to be flattened and concatenated with the target patch to form the context sequence, which is going to be input to the transformer encoder. The corresponding high resolution patch is going to serve as query coordinates to input to the transformer decoder. Different to ViT and MAE, it processes the patches pixel-wise, allowing it to be mesh-free while considering local correlations, therefore, the "patch" in the name is only about how the context information is organized. After pretraining on relatively low-resolution super-resolution tasks, it is trained on higher resolution images by omitting the neighboring patches and randomly sampling the upsampling patch. This makes it memory efficient, allowing it to super-resolute to higher resolutions. This model can be incorporated with a flow which predicts the marginal probability of pixel intensities. The model outputs the mean and variance of the latent representation for the pixel and the flow transforms it into the pixel intensity.

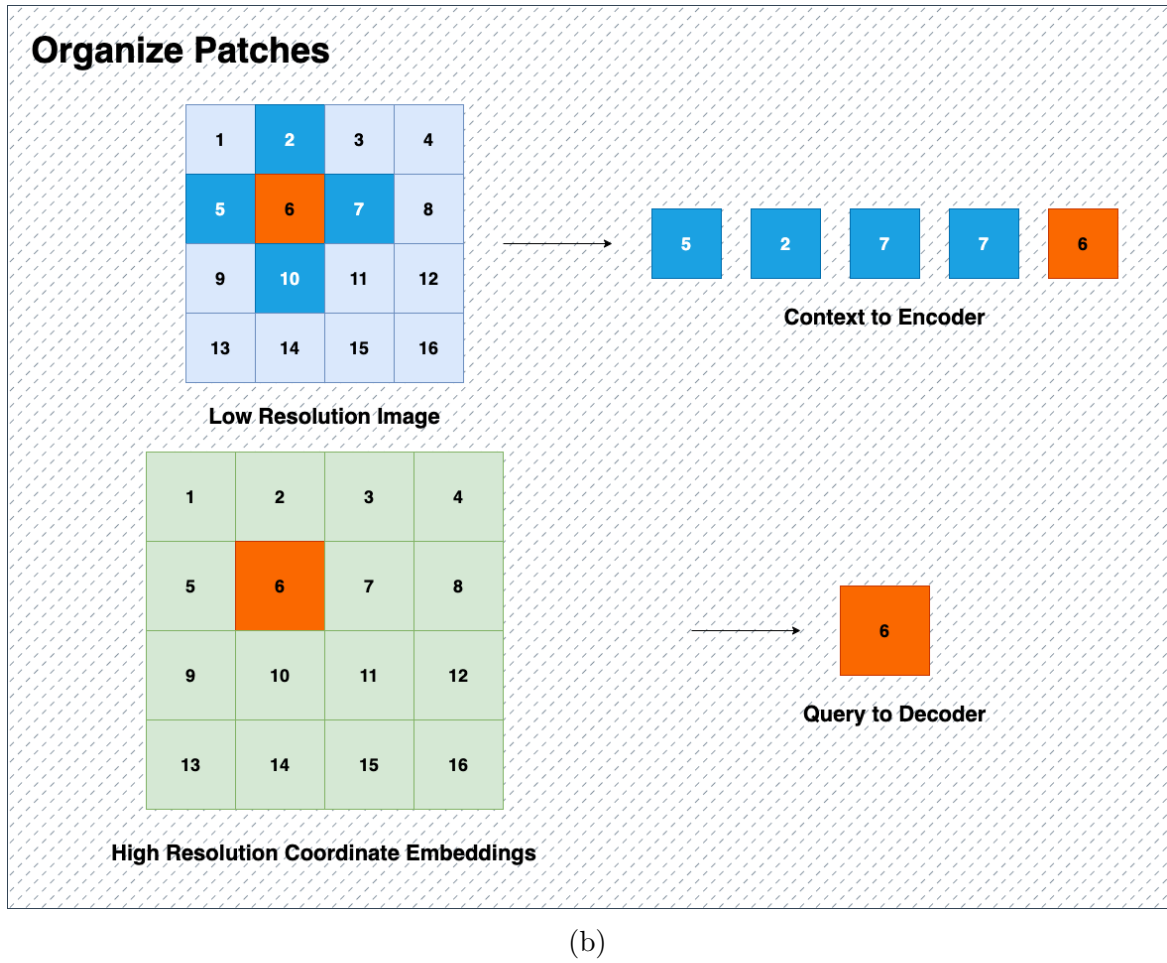
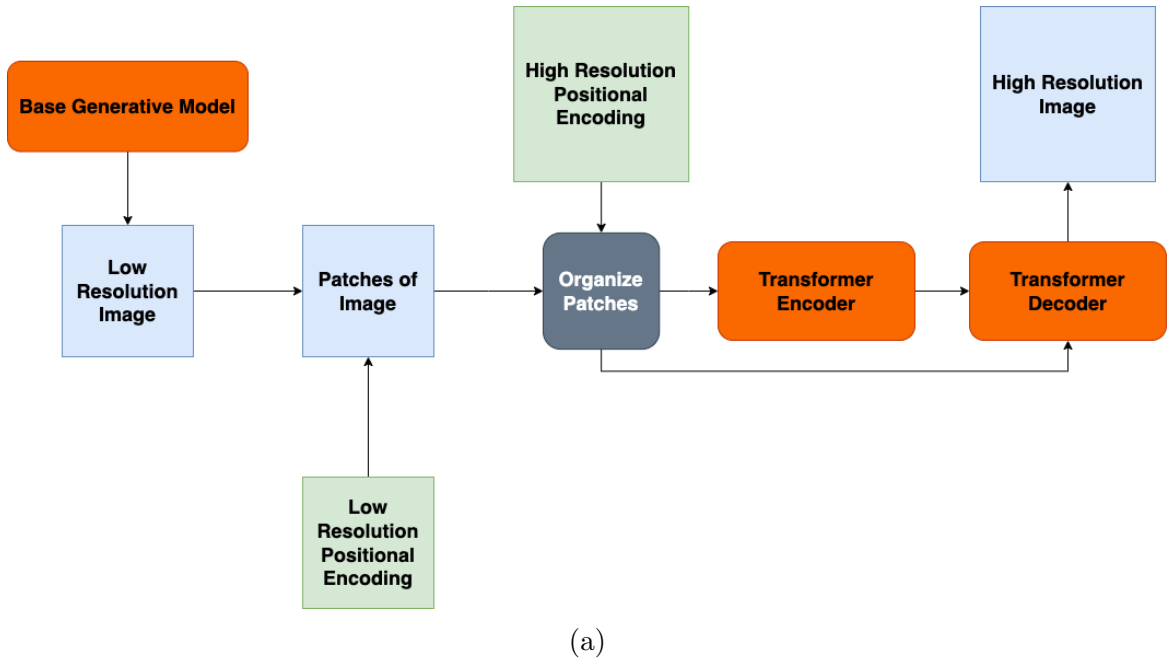


Figure 3: (a): Architecture of PPT. (b): Details of "Organizing Patches" Block.

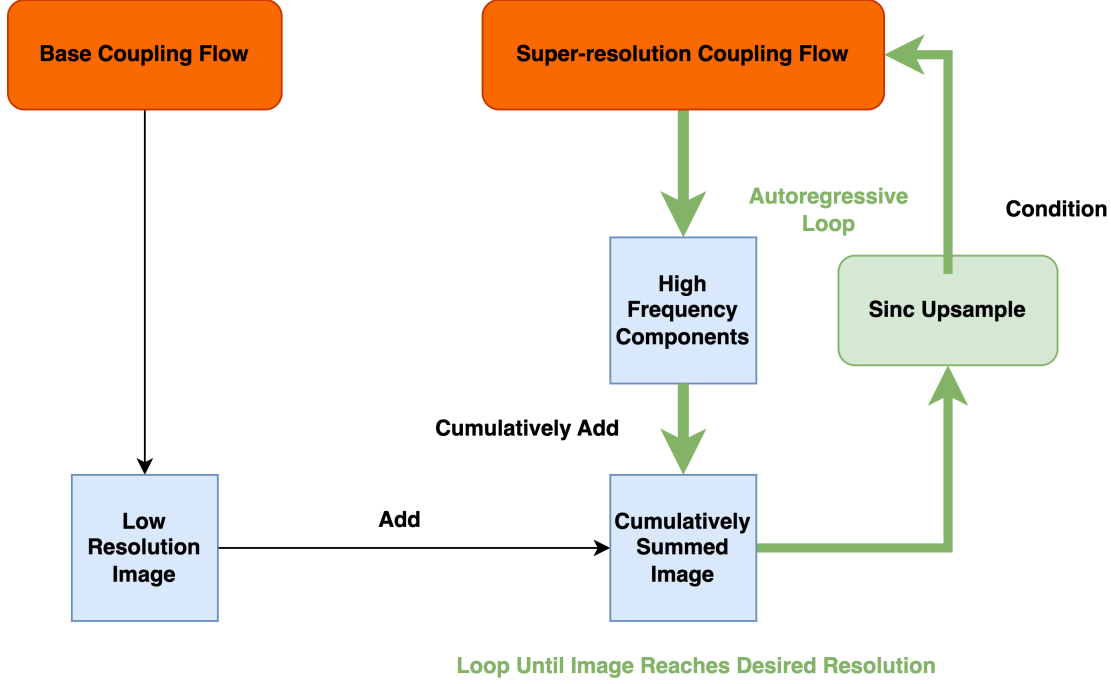


Figure 4: Architecture of CAF.

3.2 Coupling Autoregressive Flow (CAF)

Coupling flow, as previously mentioned, is capable of transforming part of the input using affine transformation conditioned on the other part. This conditioning process, however, doesn't forbid the use of external constant information, thus the model can be conditioned on external information by attaching them to different parts of the segmented input. The basic idea of CAF is that a base flow model first generates a fixed resolution image, and a second model autoregressively upsamples that fixed resolution image to a target resolution. Both models have the same structures as Re-*al*NVP but have included activation normalization and invertible 1 by 1 convolution proposed in Glow [16]. The architecture is shown in figure 4. This model is inspired by WaveletFlow [23], but unlike wavelet flow which only produces images at a fixed set of resolutions, it can generate images at arbitrary even number resolutions. To make the model memory efficient and also provide it with inductive bias, the base image is first sinc upsampled to a higher resolution, equivalent to padding zeros in its frequency domain, then used as conditions to pass into the super-resolution model, which generates the difference between the ground truth image at that resolution and the sinc upsampled image. This difference is then added back to the sinc upsampled image. This step increases the resolution by 2 each time and is repeated until the target resolution is reached. However, initial implementations has shown serious error propagations, where error in initial steps of super-resolution builds up along the autoregressive process. In addition, in most PDE solutions or images, the energy

decays as the frequency increases, but from the experiment results, the model can't easily learn this kind of decay, shown in the existence of high frequency noise on the super-resolved images in figure 19. The model requires some kind of inductive bias to learn the decay in high frequency content effectively. It was discovered that adding a normalization term to scale the high frequency content by a coefficient that decays as a function of frequency helps with the process. The normalization formula is shown in Eq. 14.

$$\bar{\mathbf{y}} = ((1 - \epsilon)\alpha^{-\lambda(\mathbf{r}-b)} + \epsilon) \times \mathbf{y} \quad (14)$$

$\bar{\mathbf{y}}$ is the normalized inferred high frequency contents, $\mathbf{y}$ is the original output, $\mathbf{r}$ is the current resolution of $\mathbf{y}$, b is the resolution of the images generated by the base flow model, and ϵ , α , and λ are tunable parameters. The normalization formula is designed such that it's one for the lowest fixed resolution image, and decays exponentially as resolution increases. The generated images became much clearer and smooth with the frequency normalization term as shown in figure 21.

3.3 Spectral Neural Operator Residual Flow (SNORF)

SNORF incorporates residual flow with FNOs, which are also called SNOs for a broader range of basis functions including Fourier basis. The relatively free architecture of residual flows allow them to use models like FNOs, which doesn't really make sense for coupling flows as explained earlier. Incorporating FNO makes it possible for the model to learn continuous mappings in the spatial domain. To accomodate for this model, like many other FNO powered generative models, e.g., GANO, the base Gaussian distribution needs to be modified to Gaussian random field to account for the continuous assumption. To limit the Lipschitz constant to below 1, the neural network responsible for processing the fixed number of lowest frequency input and the 1-by-1 convolution kernels are regulated respectively to have Lipschitz constant below 1. Since they are transforming different parts of the input that are then added together, the total Lipschitz constant of the transformation is going to be smaller than 1 as well.

3.4 Implicit Neural Representation Flow

Implicit neural representation is a way to store data using neural networks. Data like images or 3D point clouds are stored by training a neural network to regress on their values. This neural network takes as input the coordinate of the pixel values or point cloud and outputs the corresponding intensity value. In addition to efficient storage of data, it was also discovered that such representation has excellent interpolation properties. For example, in NeRF (Neural Radiance Field) [24], a neural network

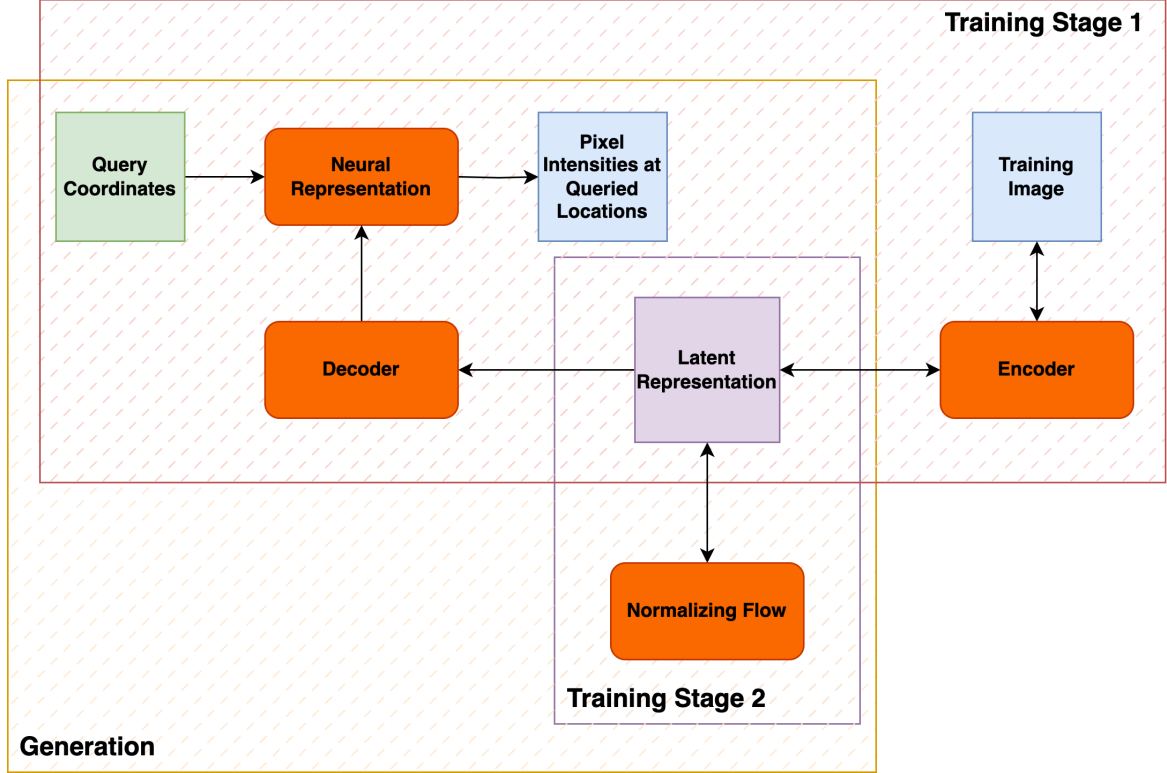


Figure 5: Architecture of Implicit Neural Representation Flow

trained on photos taken at different angles of a scene can synthesize unseen views of the same scene at excellent quality. Due to this property, it is suitable for encoding continuous data, which requires good interpolation behaviors on unseen coordinates in the training set.

In [25], it is proposed that such neural networks serve as good functional representation of the continuous data, and a generative model can be used to generate the parameters of such a neural network. This allows the generation of continuous images. Their work proposed the use of GANs to learn the distribution of these representations. However, with normalizing flows, this isn't readily possible. Adversarial training allows the calculation of loss by simply generating new synthetic data samples, while normalizing flows needs to transform the samples through the flows to calculate likelihood. The difficulty lies in the transformation from images to neural representations. Since this process also needs to go the other way around (from neural representation to image), it has to be invertible. Strictly invertible architectures usually lacks expressiveness, and thus such a structure is not practical for flows. However, if we loosen the constraint to allow for transformations that are approximately invertible, this problem can be solved.

An encoder first encodes the image into a latent representation, and a decoder is then trained to generate the neural representation that recovers the original image from the latent representation, a process that can be pretrained. After this encoder-decoder

structure is trained, the distribution of the latent representation can be used to train a normalizing flow, which is then going to synthesize more such representations. In addition, if the encoder structure is made invertible, the flow will be able to provide likelihood value directly in the spatial domain.

Although the encoder-decoder structure might seem like a good idea, a significant challenge faced is that usual neural networks can't handle varying resolution input. More importantly, we want to allow the model to be trained on any resolution images, but normalizing flow only takes a fixed dimensional input, so this requires the encoder to be able to map varying dimensional input to a fixed dimensional latent representation. This task, however, is quite common in the domain of sequence labelling. In sequence labelling, varying length sequences are mapped to fixed dimensional vectors, which, for example, in natural language processing could represent the categories of emotions. However, in most of these use cases, the fixed dimensional vectors usually represent some kind of discrete embeddings, which doesn't apply to our use cases. Therefore, instead of using the usual encoder-only transformer, the output from the encoder is passed through an attention like mechanism to calculate a weighted average value across all sequence embeddings. This gives a fixed dimensional single vector representation of the whole sequence, whatever length it is, of dimension of the embedding dimension of the sequences. In our experiment, the encoder is simply a transformer encoder, and its output is averaged to get the latent representation of the whole image.

Though sounds tempting, this architecture requires much more extensive research, and so is not included in this study. It was discovered that due to normalizing flow's weak ability in modelling very high dimensional data, the dimension of the latent representation needs to be kept low. However, this representation is also crucial for the reconstruction of the original image, meaning that low dimensional representation would not include enough information for the decoder model to produce neural representation that can reconstruct the original image clearly enough. This dilemma makes this model hard to tune, and thus a better solution is needed to circumvent the challenge.

4 Experiment Results

4.1 Metrics

Since we're dealing with PDE solutions, most inception based metrics for measuring quality of generated images wouldn't be suitable. All of these metrics measure the difference between the distribution of the generated images and the training dataset based on their features. A pretrained model is usually used to extract features from

both generated images and training images, and the difference is calculated using those features. These metrics include IS (Inception Score) [26], , FID (Fréchet Inception Distance) [27], and KID(Kernel Inception Distance) [28]. IS uses a pre-trained Inception model. It measures the KL divergence between the ground-truth label distribution and the label distribution provided by the model, with higher scores indicating better quality. FID compares the distribution of generated images to that of real images by computing the Fréchet distance between their feature extracted from a pre-trained model, and lower FID scores suggest better image quality. KID, similar to FID, measures the similarity between the distributions of real and generated images using the squared MMD (Maximum Mean Discrepancy) between their feature representations extracted from a pretrained model. These metrics are proposed initially for better trainings of GANs. They have then been shown to be excellent metrics for assessing qualities of generated natural images, even for models that's not trained using them as loss functions.

All these scores use an Inception model that is a priori agreed to be Inception v3 [29], which is pretrained on ImageNet database, a dataset of natural images containing more than 20,000 categories of real-life objects, quite different from the PDE solutions we are studying. To overcome this issue, MMD score is proposed to be directly calculated between the samples using Riesz kernel [30].

Riesz MMD score is calculated by using Riesz kernel, defined as below.

$$K(x, y) = -||x - y||^r \quad (15)$$

The MMD score is then calculated from

$$MMD^2(X, Y) = K_{XX} + K_{YY} - 2K_{XY}, \quad (16)$$

where

$$K_{XX} = \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m K(x_i, x_j). \quad (17)$$

This design of metric calculates the score directly in the image space, which doesn't require any pretrained models, and thus can work on measuring difference between any two sets of data from their feature values, making it suitable for measuring the quality of the generated PDE solutions. In our experiments, the value of r in equation 15 equals 1, which as described in [30] provides computational efficiency.

In addition, experiments were also performed on models trained on MNIST (Modified National Institute of Standards and Technology database), a dataset of handwritten digits. The performance of the models on this dataset is also tested using the MMD score, as shown in figure 20.

4.2 Datasets

The models are trained on both PDE solution dataset and MNIST dataset. Although the goal is to design a normalizing flow that works on function space, testing model’s performance on more complex dataset of natural images like MNIST has several benefits. First, since natural images are more complex, testing model’s performance on them could give insight to the model’s capability. Though most PDE solutions are smooth, there could be PDE solutions that are more complex than images of digits, and testing model’s performance on images of digits can indicate the model’s ability in handling these extreme cases. In addition, since most generative models are designed for natural images, using MNIST dataset as an addition allows the comparison of the proposed models on a broader range of existing models. Finally, the choice of MNIST dataset is due to the fact that unlike other dataset of natural images that contain 3 channels, red, green, and blue, MNIST dataset contains grayscale images only, with only 1 channel, which is very similar to PDE solutions.

4.3 Methods

Each of the proposed models are trained on PDE solutions of resolution 32 by 32. The family of PDE studied is Darcy flow, and data is from [31]. The model’s ability to generate samples beyond this resolution is not of great significance. This is due to the fact that extrapolating on unseen data is a seriously ill-defined problem. Even for all of the SOTA super-resolution models, synthesizing high resolution details requires training on images at those resolutions. Instead, the model’s ability to interpolate on seen data would be more interesting, as this studies the model’s ability to reverse the downsampling process.

The MMD score is tested across resolutions of 20, 24, 28, 32, and 48. The test set is resized to these resolutions to calculate MMD scores of the generated samples at these resolutions of the models. The number of parameters of the proposed models were also kept as close as possible to eliminate the effect of parameter count.

In addition, since some of the models studied are purely super-resolution models, i.e. they are only responsible for super-resolution but not for generating the lowest resolution image, what’s important is how the MMD varies as the resolution increases for these models instead of its absolute value, and the images used for super-resolution are the ground truth test set.

4.4 MNIST Result

As already shown in figure 20, CAF has achieved the best performance with the slowest increasing MMD score as the resolution increases and uniformly lower MMD

score than the other two proposed models. The performance of the later proposed PPT is also tested on this dataset and presented here in figure 6. As shown, the model can produce high fidelity image at resolutions as high as 2048 from a base image of resolution 16 on a NVIDIA 3090 GPU with 24GB of memory, significantly surpassed the other two proposed models.

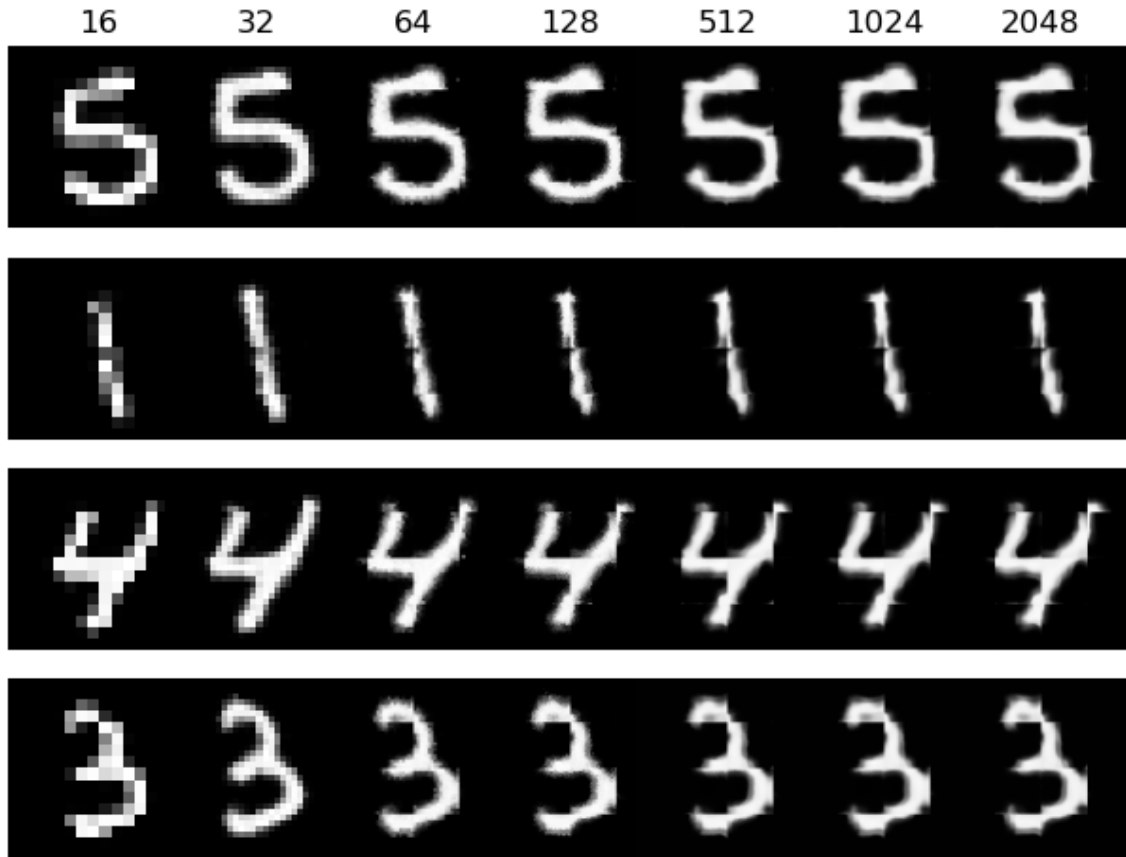


Figure 6: PPT’s generated samples across different resolutions.

4.5 Ablation Study

To prove the necessity of incorporating pixels in the neighboring patches as context information in PPT, as discussed in section 3.1, an ablation study was done to compare the model’s performance with and without the proposed improvement training two versions of PPT on MNIST dataset, one using neighboring patches in context information and the other one not using them. Both models are trained to upsample 16 by 16 images to 32 by 32, and are tested to upsample 16 by 16 images to both 32 and 64. Shown in figure 7, there are visible artefacts at the boundaries of the version of model that doesn’t use neighboring patches in context, even more serious on 64 by 64 image. The version that includes neighboring patches in context, on the other hand, outputs continuous, sharp, and coherent images of digits even at the resolution

of 64.

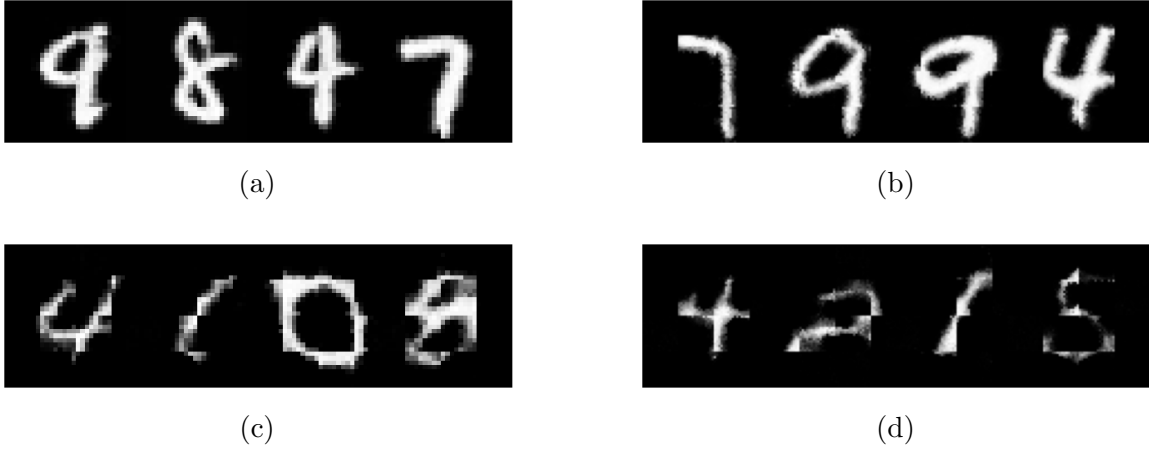


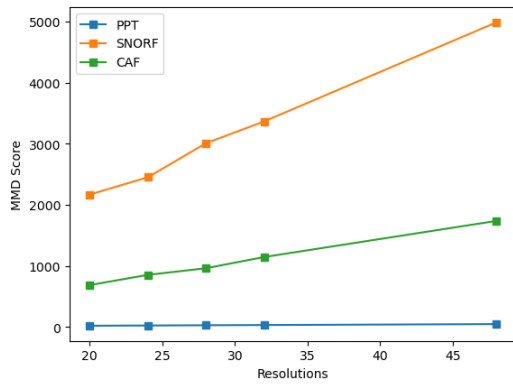
Figure 7: (a): 32 by 32 image of PPT trained with neighboring patches. (b): 64 by 64 image of PPT trained with neighboring patches. (c): 32 by 32 image of PPT trained without neighboring patches. (d): 64 by 64 image of PPT trained without neighboring patches.

4.6 Darcy Flow Result

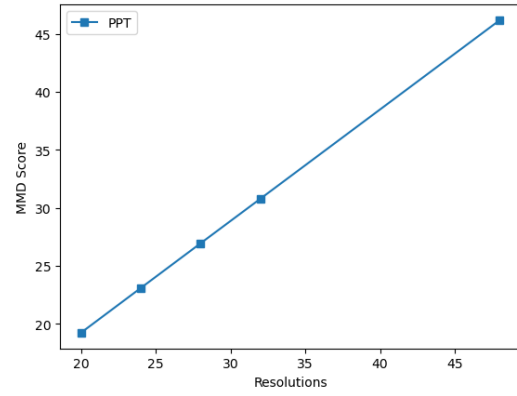
The generated samples from PPT trained on Darcy flow solutions is shown in table 2. The data used is from [1]. All data are normalized to have zero mean and unit variance. The models’ MMD scores are also tested, shown in figure 8a. As shown, all the flow-based solutions have significantly poorer performance on the flow solution dataset, which is quite surprising as they are generally much smoother than the handwritten digits, and the task of learning their distribution should be much simpler. The MMD score of PPT barely increases as resolution increases compared to the other two models. In addition, the discretization error of PPT is also tested using mean squared error across resolutions from 16 to 512. As shown in figure 9, it has similar discretization invariant performance as FNO as the discretization error is constant across a wide range of tested resolutions.

Table 1: Comparison of parameter counts of the three proposed models.

PPT	CAF	SNORF
14.3M	49.7M	34.3M



(a)



(b)

Figure 8: (a): MMD score against resolutions. (b): PPT curve from figure (a) zoomed in.

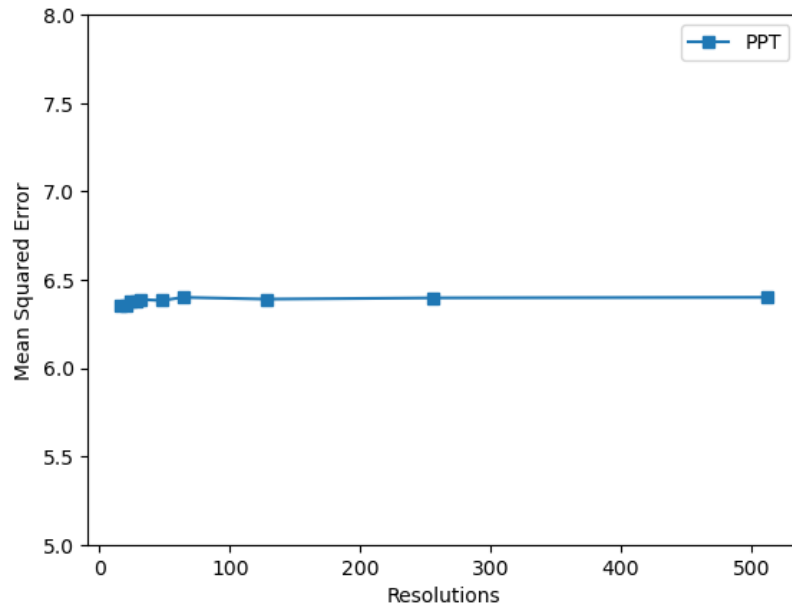
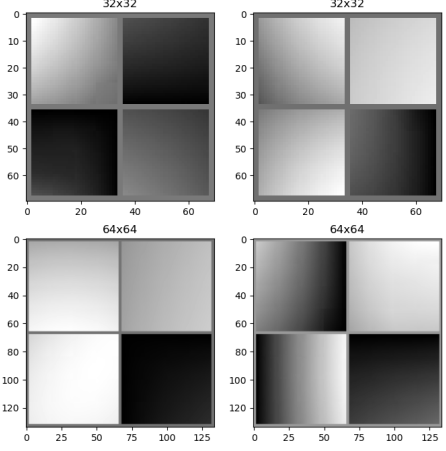
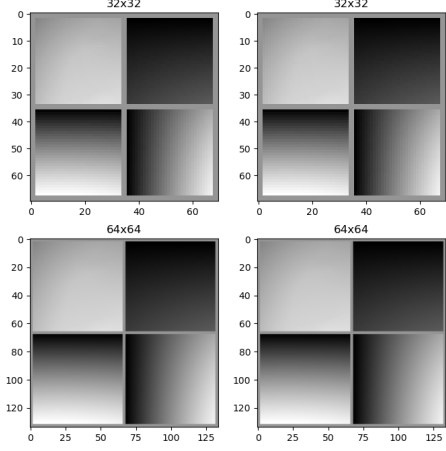


Figure 9: Mean squared error of PPT on resolutions of 16, 20, 24, 28, 32, 48, 64, 128, 256, and 512.

Table 2: Comparison of Darcy flow samples from PPT and ground truth.

PPT				Ground Truth			
							

5 Analysis

5.1 PPT

The greatest advantage of PPT is its mesh-free property. Since it directly learns the mapping from coordinates to solution values, the generated values don't have to be on the grid. Most importantly, if the way the context information is formatted is considered as part of the model, the model's super-resolved image is invariant to the order the patches are queried. In addition, the necessity of incorporating these neighboring patches can be explained by the fact that the edge pixels on queried patches actually requires the model to perform extrapolation tasks, as illustrated in figure 10. In that example, a 4 by 4 image is used as context information for the model to generate an 8 by 8 image. The effective coordinate of the pixels on the underlying continuous image is plotted for both images as the colored dots. After superimposing both grid of pixels, it can be easily seen that the red dots on the edges of the 8 by 8 image requires the model to extrapolate from the context information, while the model only needs to interpolate on the context information to get all the black dots in the centre. This implies that simply the corresponding lower resolution patch won't provide enough information for the model to infer the higher resolution patch pixels. There are several shortcomings of the models, however. The first shortcoming of the model is that it's a deterministic model, i.e. the likelihood would be for the lowest resolution image, not including all the inferred high resolution details. However, the model could be trained to output mean and variance of a base Gaussian distribution, and a separate flow model could be trained to map that distribution to the pixels.

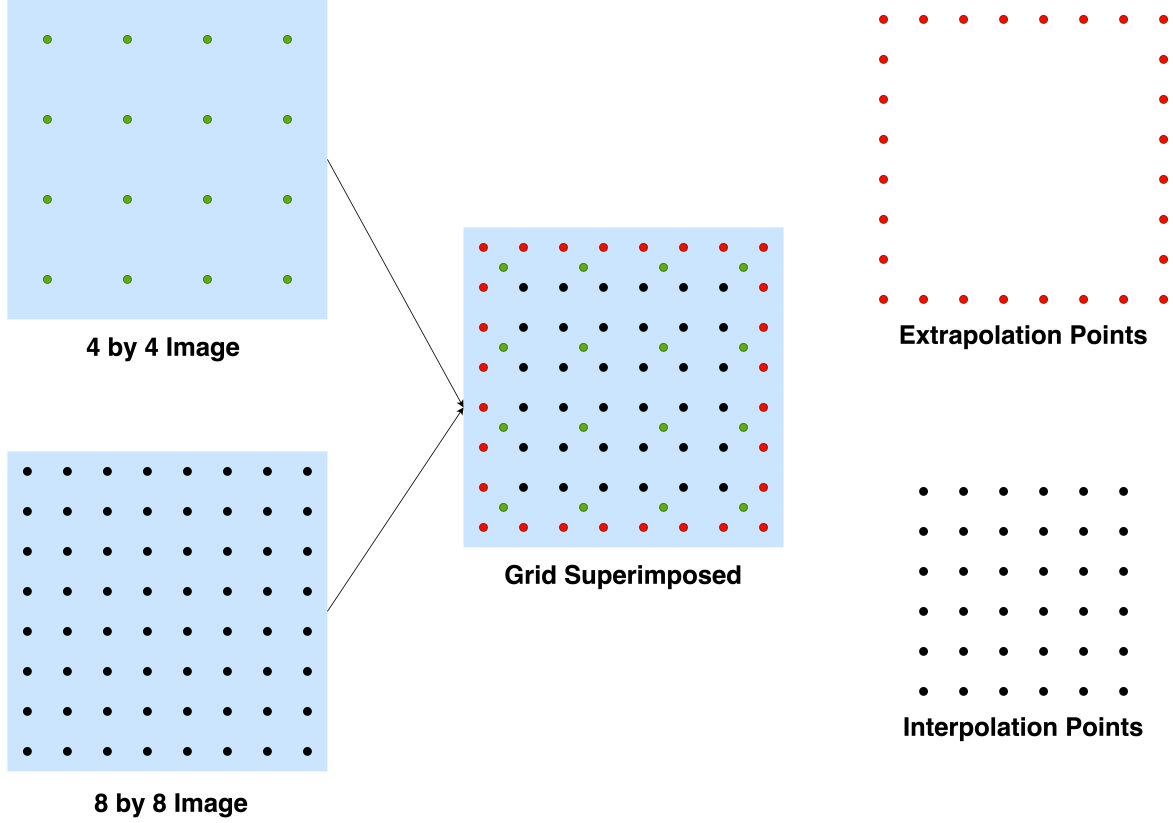


Figure 10: Illustration that some of the queried points require extrapolation tasks.

This is a very simple task as in the extreme case, the model could simply output zero variance and pixel value as mean, and the separate flow model simply needs to learn an identity mapping. Though coupling flow requires at least 2 dimensional data to work (since we’re working pointwisely in the spatial dimension, the dimension of the data would be 1, which is the channel dimension), there are other types of flow that work on 1 dimensional data, for example, augmented flow [32].

An improvement was made to change the ”MoG” of the original pixel transformer to 1 dimensional VAE, very similar to normalizing flows, details are available in appendix. As shown in figure 13, the inferred full image is clear and smooth, suggesting that the proposed task of using the pixel patch transformer to parameterize normalizing flows is simple.

The second shortcoming of the model is that it’s slightly memory-intensive due to the transformer architecture. The memory requirement scales quartically (sequence length scales quadratically as resolution increases and memory scales quadratically as sequence length increases). The segmentation into patches has alleviated this issue but the complexity remains the same. This can be easily solved by using linear-attention mechanisms in transformers. It was shown in ”Linformer” [33], that linear attention mechanism can achieve close performance to the full attention mechanism at linear complexity.

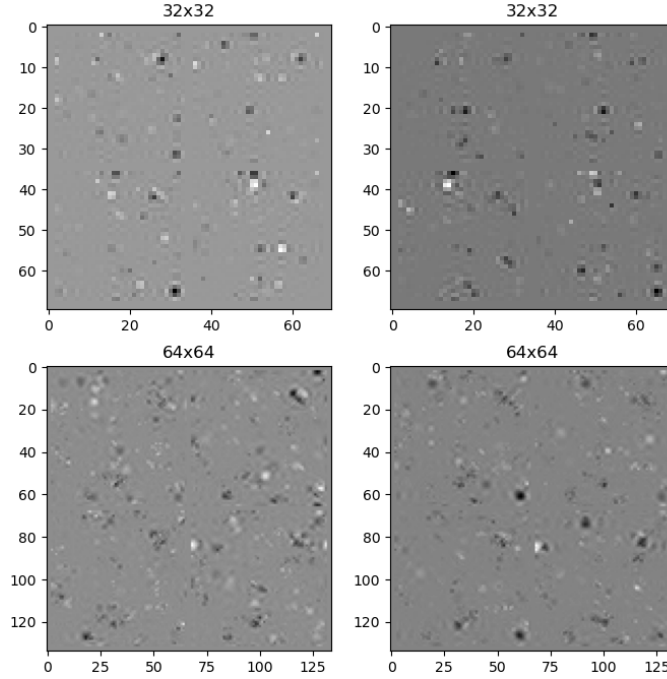


Figure 11: Poor quality generated samples from CAF.

5.2 CAF

As mentioned earlier, all flow based models have poorer performance on continuous data, which is surprising. The generated samples from CAF is shown in figure 11. After careful examination on the intermediate output of the flow layers, it was discovered that the task of modelling continuously distributed values like PDE solutions, though much smoother, should not be underestimated. The existence of dequantization makes the task of modelling discrete value distributions significantly simpler. It allows the flow to output any value, thresholding that value using sigmoid activation to between 0 and 1, multiplying it by 255 and quantize to integers to map to the correct range of between 0 and 255 of pixel values. To reverse the process, noise is added on discrete pixel values, which is then divided by 255, and passed through inverse sigmoid function. This forces the flow to concentrate the probability mass learned to between 0 and 255. Without this layer, however, the flow struggles to find a balance between numerical stability and expressiveness. For example, naively removing the dequantization layer turns out to put too much stress on the coupling layer originally designed for image modelling tasks. The original coupling layer calculates the scale value as follows,

$$\mathbf{s} = \exp(\mathbf{s}_{\text{fac}} \times \tanh(\frac{\mathbf{y}}{\mathbf{s}_{\text{fac}})}), \quad (18)$$

where y represents the output of neural network and s_{fac} represents a learnable constant that controls the magnitude of the scaling factor. It was discovered that the learning of $\mathbf{s}_{fac}$ is quite problematic as it barely changes across the training epochs, and the value of s saturates to e^{-1} and e . This puts the task of moving the range of flow from unit Gaussian to the true data distribution mostly on the shifting term. Especially for the autoregressive part of the model, the shift term turns out to be of the order of magnitude of 10^{25} . This makes training arduous as the model constantly outputs NaN values. In the end, the flow layers had to be modified to use the same parameterization trick on the shift term as on the scale term. In addition, $\mathbf{s}_{fac}$ is randomly initialized to avoid local minimum, as opposed to the original zero initialization [2]. This solved the problem of numerical instability, but has also hindered the model’s expressiveness.

5.3 SNORF

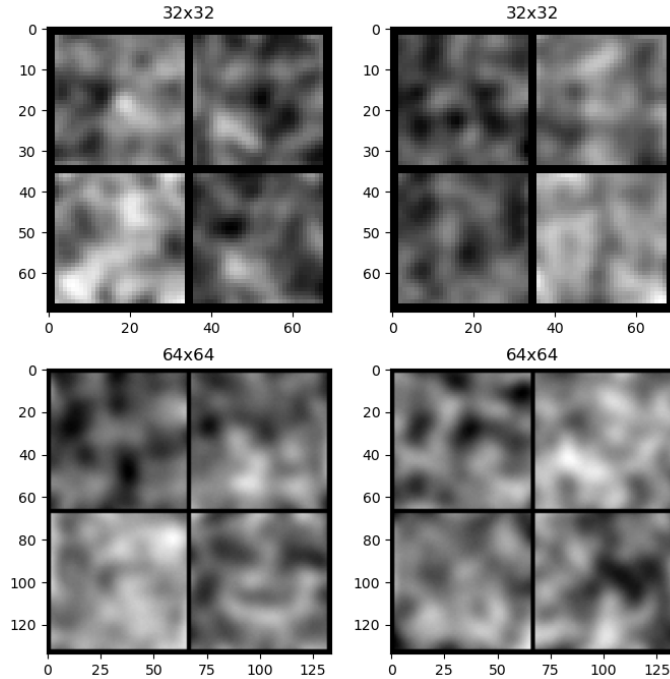


Figure 12: Generated sample of SNORF trained on Darcy flow dataset.

Similar to CAF, SNORF performed poorly on Darcy flow dataset. However, its poor performance on MNIST dataset as shown in figure 22 makes the result not surprising. It was discovered that for all the generative models using Gaussian random field as base distributions, the smaller the characteristic length in the covariance function, i.e. the less smooth and closer to independent Gaussians the random function, the

better the performance on fixed resolution dataset. A hypothesis that worths further studies is made here that forcing the base function to be continuous actually puts extra burden on the model as they have to first decouple the spatial correlation in the base function and then transform the decoupled random function into the data samples. Together with FNOs, which is relatively weak in modelling high frequency components, and with residual flow layers, which poses Lipschitz constant constraint on the already limited FNO, the model’s poor performance can be expected. Shown in figure 12, the generated samples resemble the structure of Gaussian random field more than the smooth varying structure of Darcy flow solutions. This proves the model’s weak ability to transform the base function using Lipschitz constant constrained FNOs.

6 Future Studies

As shown in figure 6, PPT learns to upsample from 16 by 16 image to 2048 by 2048 image, even though it’s trained solely on 32 by 32 image. However, there are visible artefact as the resolution increases. With careful comparison, it could be told that the part of the artefact comes from the sharp discrete edges on the 16 by 16 image. It would thus be interesting to see if this part of the artefact can be eliminated by varying the resolution of the 16 by 16 image used as context information. In addition, another part of the artefact comes from the patch edges. Although neighboring patches are included as context information, when super-resoluting different patches, the context information is different, and it could be difficult for the model to produce consistent pixel values at the patch boundaries on untrained data. A method to force consistency at the patch boundaries is thus of great interest. Possible solutions might approach the problem by including additional terms in the loss function that penalizes inconsistency or simply post-processing the output of the image at the patch boundaries to make them continuous.

For implicit neural representation flow, as mentioned previously, the bottleneck is the performance of encoder-only transformer and normalizing flow. With more powerful flow models, the dimension of the latent representation output by the transformer could be larger, and the performance of the encoder could improve. Shown in a recent study [34], other types of flow models like diffusion models which are based on stochastic flows is extremely powerful and can directly generate neural network with good performance. It would thus be interesting to use diffusion models to model the distribution of the latent representation.

7 Conclusion

This project proposes several models aiming to generate PDE solutions at arbitrary resolutions. Several models, with their own advantages and disadvantages are proposed and analyzed. Base on these analysis, a final better model, PPT, is designed and tested. It has shown excellent performance in super-resolution tasks on resolutions far beyond its training data, as shown in figure 6. It is also relatively memory efficient by processing images only locally, as shown in the model’s ability to generate 2048 by 2048 images using the same GPU as the other tested models which struggled to produce images of resolution above 64 by 64. In addition, it can serve as a backbone to incorporate with many different types of generative models. It has great potential in image super-resolution and continuous data modelling tasks like generating PDE solutions, and subsequent studies can develop wider families of generative models on function space based on this work.

References

- [1] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Fourier neural operator for parametric partial differential equations,” 2021.
- [2] L. Dinh, J. Sohl-Dickstein, and S. Bengio, “Density estimation using real nvp,” 2017.
- [3] Z. Hu, K. Shukla, G. E. Karniadakis, and K. Kawaguchi, “Tackling the curse of dimensionality with physics-informed neural networks,” *Neural Networks*, vol. 176, p. 106369, Aug. 2024.
- [4] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” 2020.
- [5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 2014.
- [6] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2022.
- [7] D. J. Rezende and S. Mohamed, “Variational inference with normalizing flows,” 2016.
- [8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.
- [9] R. J. George, J. Zhao, J. Kossaifi, Z. Li, and A. Anandkumar, “Incremental spatial and spectral learning of neural operators for solving large-scale pdes,” 2024.
- [10] M. A. Rahman, M. A. Florez, A. Anandkumar, Z. E. Ross, and K. Azizzadenesheli, “Generative adversarial neural operators,” 2022.
- [11] P. Hagemann, S. Mildenerberger, L. Ruthotto, G. Steidl, and N. T. Yang, “Multi-level diffusion: Infinite dimensional score-based diffusion models for image generation,” 2023.
- [12] H. Lee, C. Pabbaraju, A. Sevekari, and A. Risteski, “Universal approximation for log-concave distributions using well-conditioned normalizing flows,” 2021.
- [13] F. Draxler, S. Wahl, C. Schnörr, and U. Köthe, “On the universality of coupling-based normalizing flows,” 2024.

- [14] R. T. Q. Chen, J. Behrmann, D. Duvenaud, and J.-H. Jacobsen, “Residual flows for invertible generative modeling,” 2020.
- [15] J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen, “Invertible residual networks,” 2019.
- [16] D. P. Kingma and P. Dhariwal, “Glow: Generative flow with invertible 1x1 convolutions,” 2018.
- [17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023.
- [18] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.
- [19] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [20] S. Tulsiani and A. Gupta, “Pixeltransformer: Sample conditioned signal generation,” 2021.
- [21] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” 2021.
- [22] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, “Masked autoencoders are scalable vision learners,” 2021.
- [23] J. J. Yu, K. G. Derpanis, and M. A. Brubaker, “Wavelet flow: Fast training of high resolution normalizing flows,” 2020.
- [24] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “Nerf: Representing scenes as neural radiance fields for view synthesis,” 2020.
- [25] E. Dupont, Y. W. Teh, and A. Doucet, “Generative models as distributions of functions,” 2022.
- [26] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” 2016.
- [27] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, “Gans trained by a two time-scale update rule converge to a local nash equilibrium,” 2018.

- [28] B. Dai, A. Gretton, Z. Zhou, and B. Schölkopf, “Demystifying mmd gans,” *arXiv preprint arXiv:1711.08561*, 2017.
- [29] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” 2015.
- [30] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, “A kernel two-sample test,” *Journal of Machine Learning Research*, vol. 13, no. 25, pp. 723–773, 2012.
- [31] Z. Li, “Fourier neural operator dataset.” https://github.com/zongyi-li/fourier_neural_operator, 2021. MIT License.
- [32] C.-W. Huang, L. Dinh, and A. Courville, “Augmented normalizing flows: Bridging the gap between generative flows and latent variable models,” 2020.
- [33] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, “Linformer: Self-attention with linear complexity,” 2020.
- [34] K. Wang, Z. Xu, Y. Zhou, Z. Zang, T. Darrell, Z. Liu, and Y. You, “Neural network diffusion,” 2024.
- [35] A. Colton, “Your vae sucks.” <https://theadamcolton.github.io/your-vae-sucks.html>, 2024. Accessed: 2024-05-22.

Appendix A: Pixel VAE

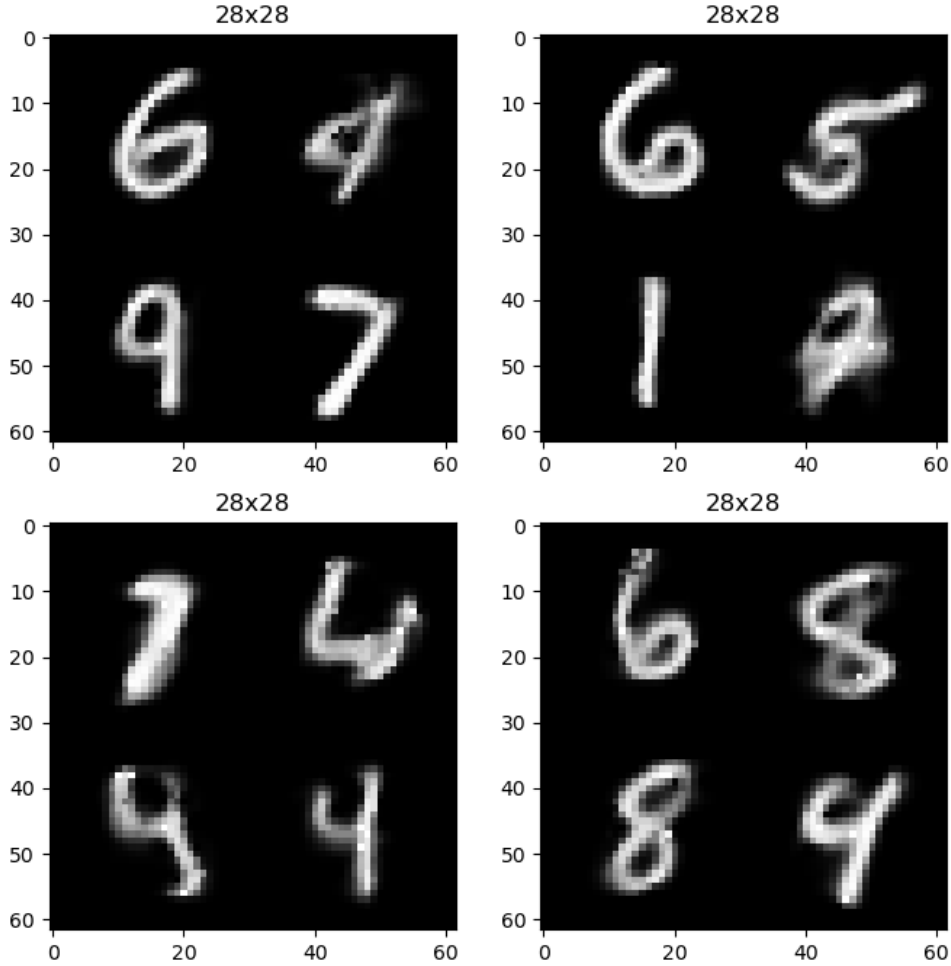


Figure 13: Generated images of Pixel VAE using 64 randomly sampled pixels as context information.

Pixel VAE was proposed to substitute the output distribution of pixel transformer from MoG to 1-dimensional VAE. Just like pixel transformer, this model is trained by taking a 28 by 28 image, randomly delete most of the pixels, using the remaining pixels as context, making the model predict variances and means of a Gaussian distribution, whose samples is then going to be passed through a VAE to generate predicted pixels. The model’s generated images of digits, as shown in figure 13, has similar quality as pixel transformer. However, it is a valid probabilistic model due to the use of VAE, unlike pixel transformer, which is deterministic due to their use of expected value of the ”pseudo” MoG. The main shortcoming of this model, however, is that it struggles to generate higher resolution images due to the memory constraint. Since the query

and context sequence together forms the whole image, this model’s memory burden is heavy as it considers all pixels in the image. This model has inspired the project towards memory efficient models that work on local areas of an image.

Appendix B: Pixel Fourier Transformer Flow (PFTF)

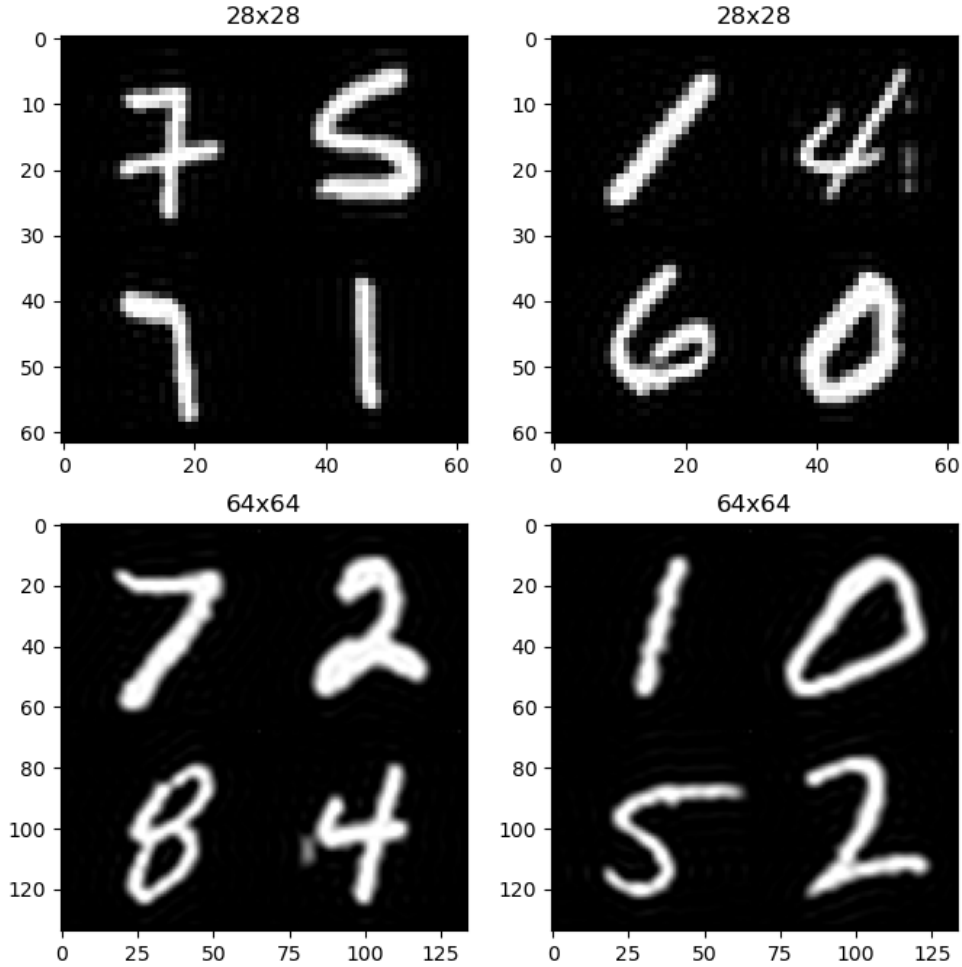


Figure 14: Sample image from PFTF trained on 32 by 32 images.

A model was proposed to work in the frequency domain, predicting higher frequency bins using lower frequency bins as context information. A similar model as pixel transformer was used. The low resolution image of resolution 16 by 16 is transformed into frequency domain via real fft, which dispose of the redundant conjugate symmetric part in the spectrum. The frequencies are used as context information, and higher frequency components are queried. The transformer outputs mean and variance

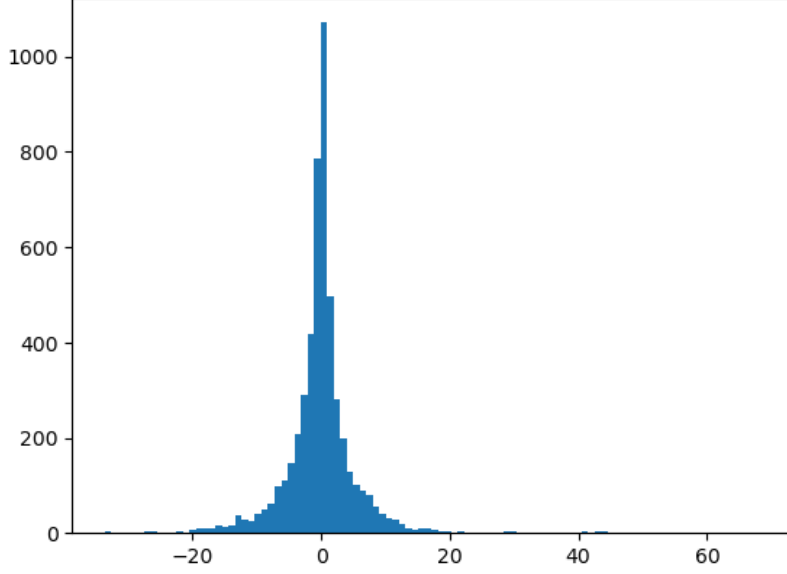


Figure 15: Histogram of frequency bins of an FFT transformed image.

of a 2-dimensional Gaussian (one dimension for real part and one for imaginary part), which is then transformed by a flow into the frequency component corresponding to the queried coordinate.

According to [35], frequency bins follow Laplace distribution. Shown in figure 15, this is indeed the case. Since L1 loss is the most suitable loss for regression tasks on Laplace distributed data, it's adopted as the loss function for this model.

Figure 14 shows the sampled images at different resolutions. At first glance, they might seem fine, but there exist ringing artefact around the edges of the digits, usually caused by unnatural frequency domain changes. The spectrum of predicted and ground-truth image is shown in figure 16a and 16b. As shown, the model fails to learn the decaying energy property of natural images as frequency increases. Unlike in the spatial domain, where close pixels have high correlations, in the frequency domain, almost all frequency levels are correlated, no matter the difference. Shown in figure 17, the mutual information between the 16th frequency level and all equal or higher frequencies is plotted against the frequency levels. As shown, the mutual information values between the 16th frequency level and all higher frequency levels are almost constant, suggesting that there's no strong local correlation between frequency components. This means generation in the frequency domain is much harder than in spatial domain. In addition, the error of prediction would be magnified by the frequency domain when transformed back into the spatial domain, leading to poor performance of the model on very high resolutions. This model's failure and relevant analysis brought the project's focus back to modelling data in the spatial domain.

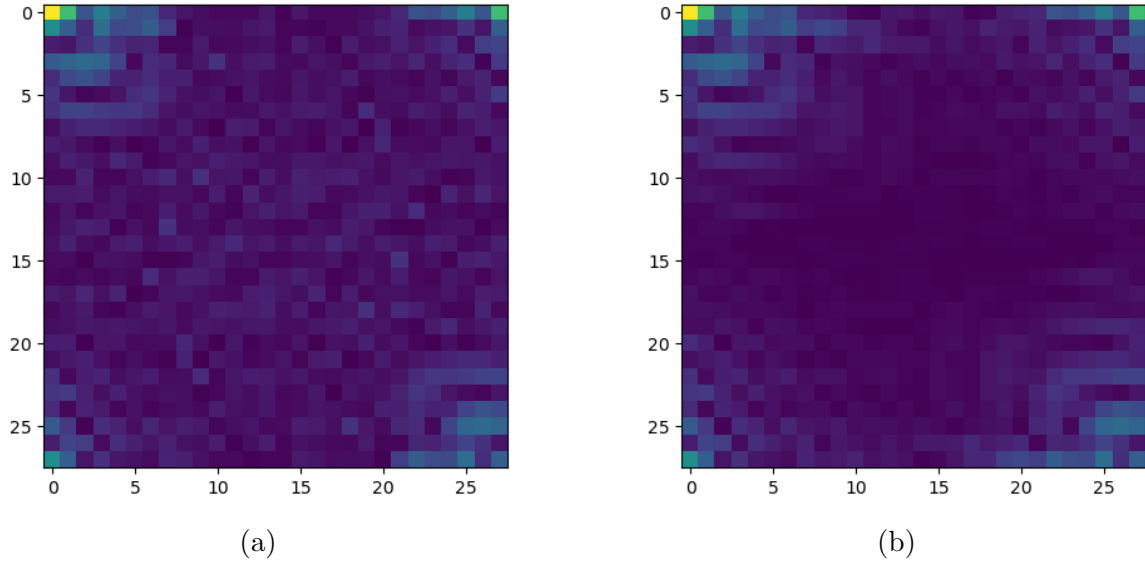


Figure 16: (a): PFTF's predicted spectrum. (b): Ground-truth spectrum.

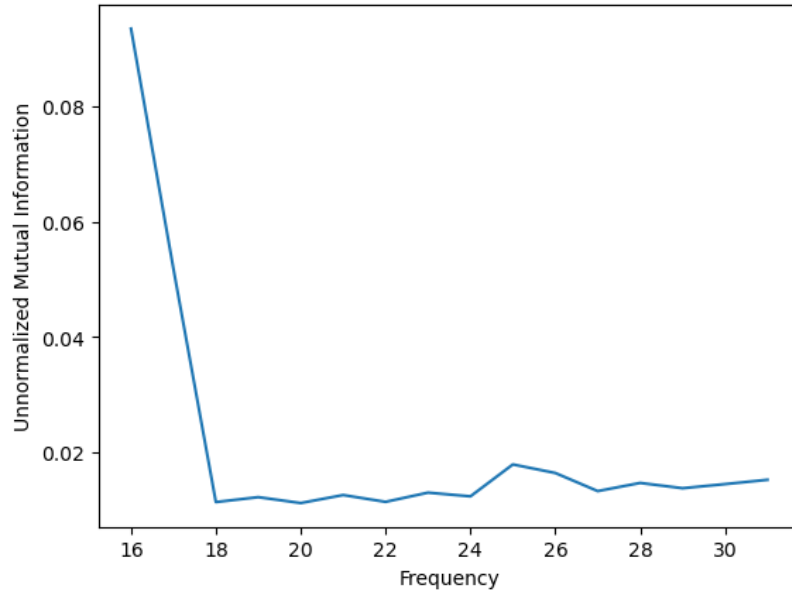


Figure 17: Mutual information between frequency components from 16 to 32 and the frequency component at 16.

Appendix C: Patch Transformer

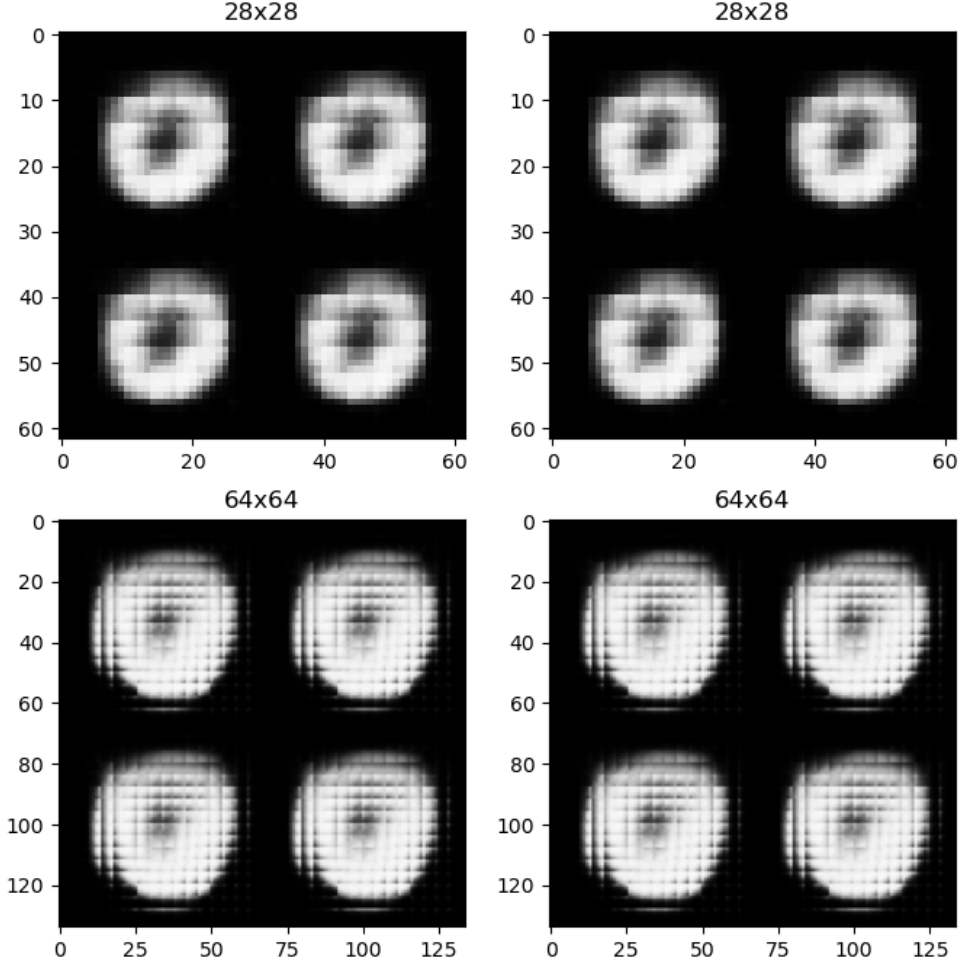


Figure 18: Generated images of patch transformer trained on digits of zero.

It could be argued that directly modelling the patches would be a more memory efficient choice. This is the approach taken by ViT and MAE. The pixels in a patch are concatenated together to be considered as features, different from them being considered as sequences in the proposed pixel patch transformer method. In theory, this could reduce the computation time greatly, as a 32 by 32 image has 1024 pixels to process, which is a sequence of length 1024 and feature dimension 1. If each patch of 4 by 4 pixels is considered as a feature dimension, the feature dimension would be 16, while sequence dimension would be 64. This shortens the total effective sequence length by 16. Following these reasoning, a patch transformer was built and tested. It's essentially a pixel transformer that takes as input patches instead of pixels. It

similarly randomly drops part of the patches and use the remaining patches as context information to predict the dropped patches. A quick test was done to train it on images of only digit zero. The sampled images are shown in figure 18.

As illustrated in the figure, the generated images have clear artefact at the boundaries. Higher resolution version has more apparent discontinuities at the patch boundaries. The model performs very poorly even with 17M parameters and trained to model a single digit.

This suggests that using patches directly as sequences is not suitable for light-weight applications like this. As shown in ViT [21], even the most basic ViT has a number of parameters of 86M. Moreover, ViT was originally designed for image classification tasks, which is much simpler than directly generating images. In MAE [22], which performs similar patch inpainting tasks as pretraining tasks and uses a similar encoder-decoder structure, simply the transformer encoder has a parameter count of 87.8M. Therefore, a more suitable lightweight model should use pixels as sequences instead of patches. These analysis led to the design of pixel patch transformer.

Appendix D: Previous Results

Results from technical milestone report that’s used in this report are presented here.

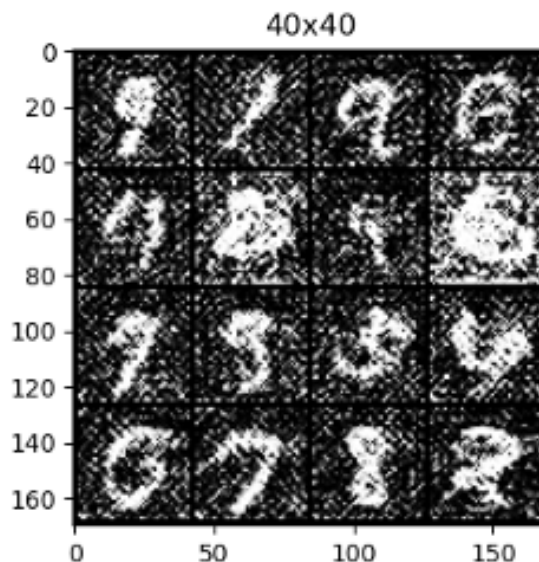


Figure 19: High frequency artefact in raw outputs of CAF.

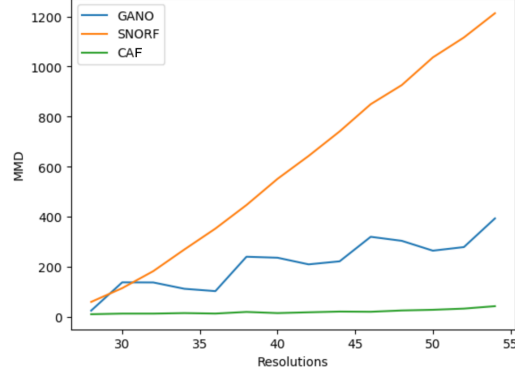


Figure 20: MMD scores of CAF, SNORF, and GANO on MNIST dataset.

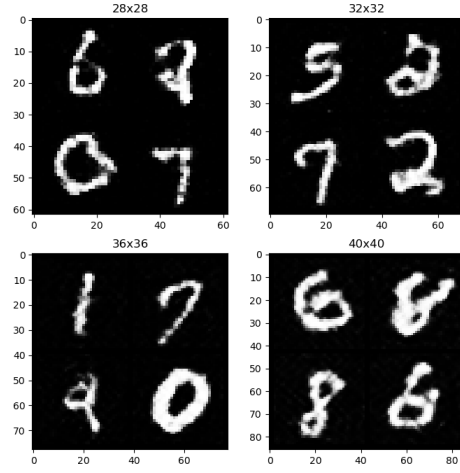


Figure 21: Generated samples of CAF trained on MNIST.

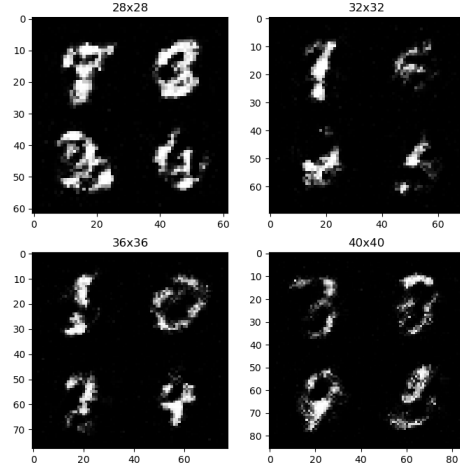


Figure 22: Generated samples of SNORF trained on MNIST.

Appendix E: Pipeline

A pipeline was built to allow for automatic training, testing, and visualisation of the proposed models¹. The model hyperparameters, training and testing details are all

¹All codes used available at https://github.com/shuohantao/benchmark_iib.

configured by a yaml file. An example is shown below.

```
data:
  path: data/
  dataset: mnist #choices are mnist, darcy, and cifar10
  batch_size: 64
  num_workers: 4
  num_samples: 10000
  shape_setting: [[32, 0.5], [16, 0.5]] # allows for mixed resolution batches
                                         # first element resolution,
                                         # second element percentage
                                         # resolutions sampled from
                                         # multinomial distribution
  targets: # digits to include in mnist
    - 0
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - 7
    - 8
    - 9
  mono: True # whether dataset is mono channeled
  upper: 255 # range to normalize the input when visualizing,
  lower: 0 # null to disable normalization
model:
  name: PixelPatchTransformerFlow
  name_sr: null # choice of super-resolution model
  loss_unit: null # null for raw output, bpd for bits per dim
  is_gan: False # if True, return tuple loss in forward pass,
                # first element being generator loss,
                # second element discriminator loss,
                # wrap both generator and discriminator in a single class,
                # set generator training frequency internally,
                # train either generator or discriminator in a single forward pass,
                # loss for part that's not trained needs to be None
  details: # arguments to pass into model's constructor
  CAF:
```

```

shape: [1, 1, 16, 16]
depth_1: 8
depth_2: 8
depth_3: 16
modes: 16
fourier: True
normalize: True
MultilevelDiff:
  prior_modes_1: 28
  prior_modes_2: 14
  prior_scale: 1
  width: 64
Gano:
  d_co_domain: 20
  modes_schedule: [12, 8, 4, 2]
  npad: 2
  n_critic: 5
  default_res: 28
  lambda_grad: 10
  alpha: 1.5
  tau: 1
ResSNO: #SNORF in the report
  n_layers: 64
  dim_hidden: 256
  depth: 7
  k_max: 11
  n_fourier_coeffs: null
  coeff: 0.98
  n_ldj_iter: null
  n_exact_terms: 2
  n_fixed_iter: 50
  use_unbiased: False
  n_spectral_iter: 20
  type: grf
  ls: 0.1
  var: 1
  eps: 0.001
WaveletFlow:
  num_scales: 2

```

```

max_res: 64
step_settings: [16, 16, 16] # depths of flow,
                        # the last one is for unconditional flow
conv_settings: [128, 128, 128] # width of convolution channels
base_split: True
PixelTransformer:
  ndp: 2
  ndv: 1
  nde: 128
  nhead: 4
  num_bins: 256
  num_encoder_layers: 10
  num_decoder_layers: 10
  mode: "fourier"
  init_factor: 2
  context_num: 110
  sample_mean: False
PixelVAE:
  ndp: 2
  ndv: 1
  nde: 64
  nhead: 4
  num_encoder_layers: 10
  num_decoder_layers: 10
  mode: "fourier"
  init_factor: 2
  context_num: 64
  sample_mean: False
PixelTransformerFourierCoupling:
  ndp: 2
  ndv: 2
  nde: 128
  nhead: 8
  num_flow_layers: 32
  num_encoder_layers: 10
  num_decoder_layers: 10
  mode: "fourier"
  init_factor: 2
  context_num: 16

```

```

    sample_mean: False
PatchTransformer:
    patch_size: 4
    dim: 512
    depth: 8
    heads: 8
    channels: 1
    beta: 0.01
PatchVAE:
    patch_size: 16
    n_hid: 256
PixelPatchTransformerFlow:
    ndp: 2
    ndv: 1
    nde: 128
    nhead: 8
    num_flow_layers: 32
    num_encoder_layers: 10
    num_decoder_layers: 10
    mode: "linear"
    init_factor: 2
    context_num: 16
    patch_res: 4 # number of patches to divide the image into per side
    num_rand_samples: 16 #number of randomly sampled patches per forward pass
RealNVP:
    depth_1: 6
    depth_2: 6
    depth_3: 12
train:
    epochs: 10 # number of epochs to train the model
    lr: 0.00001 # learning rate of the optimizer
    lr_d: 0.00001 # discriminator learning rate, ignore if model is not GAN based
    save_freq: 1 # frequency in epochs to save the checkpoint of models
    test_freq: 1 # frequency in epochs to test the model and visualize
    save_path: tmp/runs/ckpt/ # path to save the trained model's checkpoint
    test_path: tmp/runs/test/ # path to save the test results
    test_lowest_resolution: null # lowest resolution in test resolutions,
                                # set to null to use test_resolutions instead
    load_path: null # path to load the model checkpoint, null to train a new model

```

```
load_path_sr: null
save_name: null # name to save the model file, null to save it by its class name
test_resolutions: [32, 32, 64, 64] # customized resolutions to test
MMD_test:
    resolutions: [16, 20, 24, 28, 32, 48, 64, 128, 256] # resolutions used
                                                         # for testing MMD score
    batch_size: 10 # batch size for dataset used in testing MMD score
    num_batch: 10 # number of batches used to calculate average
    pickle_save_dir: test/mmd_test/ # path to save the MMD test result
```

Appendix F: Risk Assessment Retrospective

In the initial hazard assessment process, no hazards are identified to be likely to be encountered during the project. Having finished the project, the same could be said as the project doesn't involve any physical experiment, and none of the listed hazards applies.