# Simulating the Universe using Diffusion Models

Chamod Kalupahana Mesthrige, Kaung Kyaw, Kieren Mason, Nikita Ravojt
School of Physics and Astronomy
University of Nottingham

24/05/2023

**Abstract**

This project outlined in this report focused on two primary objectives - Parameter Inference and Image-to-Image Generation. The cosmological data from the CAMELS Multifield Dataset (CMD) used in the project was introduced and contextualised, along with an overview of the relevant machine learning background. The Parameter Inference model was able to predict the cosmological parameters to within $4.0\%$ for $\Omega_m$ and $2.2\%$ for $\sigma_8$ for the SIMBA suite. The Image-to-Image Generation task found that a diffusion model with an RU-net and ARU-net had the highest performance for image-to-image generation. Adding the diffusion mechanism to these models resulted in superior performance, resulting in the generation of novel images. This provides a useful groundwork for testing the ability for a diffusion model to impaint N-body to hydrodynamical simulation. This diffusion model showed that diffusion can successfully learn supernova and black hole feedback features from hydrodynamical maps. Future studies would aim to refine the accuracy of the generated images and to use them as real cosmological data, hence providing a new, more efficient pipeline to obtain simulated cosmological data than generating full gravo-magneto-hydrodynamical simulations from scratch.

# Contents

**1 Introduction** **4**

**2 Data Context** **4**
2.1 The CAMELS Multifield Dataset . . . . . 4
2.1.1 N-body Suite . . . . . 5
2.1.2 IllustrisTNG and SIMBA Suites . . . . . 5
2.2 N-Body to Hydrodynamical Simulations . . . . . 6

**3 Computational Theory** **7**
3.1 Machine Learning Background . . . . . 7
3.1.1 Activation Functions . . . . . 8
3.1.2 Loss Functions . . . . . 8
3.1.3 Back-Propagation . . . . . 8
3.1.4 Optimisation . . . . . 9
3.2 Convolutional Neural Networks . . . . . 13
3.3 Probablistic Diffusion Models . . . . . 13
3.3.1 Markovian Forward Process . . . . . 13
3.3.2 Backward Process . . . . . 14
3.3.3 Non-Markovian Forward Process . . . . . 14
3.3.4 Diffusion Schedule . . . . . 15
3.3.5 Training in Practice . . . . . 15

**4 Parameter Inference** **15**
4.1 Method . . . . . 16
4.1.1 Input Pipeline and Pre-Processing . . . . . 16
4.1.2 Model Architecture . . . . . 16
4.1.3 Hyperparameters and Training Procedure . . . . . 18
4.2 Results . . . . . 19
4.3 Discussion . . . . . 22

**5 Image-to-Image Generation** **23**
5.0.1 U-Net Architectures . . . 23
5.0.2 Novel N-body Diffusion Method . . . 29
5.0.3 N-body to Hydro Diffusion Method . . . 31
5.0.4 Parameter to Map Diffusion Method . . . 32
5.1 Results . . . 34
5.1.1 N-body to Hydro U-net . . . 34
5.1.2 Novel N-body Diffusion generation . . . 35
5.1.3 N-body to Hydro Diffusion . . . 37
5.1.4 Parameter to Map Diffusion . . . 40
5.2 Discussion . . . 41
5.2.1 N-body to Hydro U-net . . . 41
5.2.2 Novel N-body Diffusion generation . . . 42
5.2.3 N-body to Hydro Diffusion . . . 43
5.2.4 Parameter to Map Diffusion . . . 46

**6 Conclusions** **47**
6.1 Review of Results . . . 47
6.2 Next Steps . . . 47
6.3 Wider Implications of AI Image Generation . . . 48
6.4 Summary and Parting Words . . . 48

# 1 Introduction

Observations of the Universe on large scales (of order 10 $h^{-1}$ Mpc) require sophisticated, state-of-the-art apparatus. Such apparatus is expensive and in high demand, making it difficult to have time allocated to use the apparatus. Observations of this nature also have large degrees of error associated with them due to errors that arise with observational cosmology, such as noisy photons. Instead, it is often more convenient for cosmologists to use simulated maps of the large-scale Universe. Computational models which attempt to simulate the Universe have varying complexity. The simplest type of model which attempts to simulate the Universe is the gravitational N-body model; a model which iterates over many time steps and updates the attributes (e.g. position, velocity, acceleration) of each particle accordingly [1]. The other, more sophisticated type of simulation is the hydrodynamical simulation. Hydrodynamical simulations are a much more comprehensive type of cosmological simulation. As well as considerations of gravity, hydrodynamical models also simulate various astrophysical phenomena arising due to hydrodynamic interactions between particles. Both types of simulation have advantages and disadvantages. N-body simulations are much cheaper computationally than hydrodynamical simulations as they are far less sensitive to small changes and hence can be iterated over larger time steps. However, they are much less representative of the Universe as they neglect astrophysical effects. Hydrodynamical simulations, on the other hand, are much more computationally expensive. This is due to the non-deterministic nature of the equations which govern cosmological hydrodynamics, which is very sensitive to small changes and thus requires iterations over many very small time steps. Hence, a state-of-the-art supercomputer with unprecedented processing capability would be required to reach feasible timescales. However, hydrodynamical simulations provide a more accurate representation of the Universe by being able to simulate astrophysical phenomena such as stars, galaxies, and clusters.

It is desirable to obtain new hydrodynamical data, yet the large computational expense makes this difficult. To tackle this, different routes to obtain hydrodynamical simulation data should be explored. One such route is to construct and train a machine learning model to take N-body data as input and output the corresponding hydrodynamical data. Such a model would cut the computational expense of obtaining hydrodynamical data massively. The aim of this project was to construct this proposed model, in order to obtain new hydrodynamical data from input N-body data.

This report provides a comprehensive overview of the relevant contextual information which was required for this project. The two-part structure of this project is also discussed with clear methodologies and results for both the Parameter Inference and Image-to-Image Diffusion parts of the project. The results are then evaluated and summarised, then possible modifications which could be made were the project to be repeated are suggested. Finally, the implications and possible next steps are discussed as the future of cosmological applications through generative machine learning models is explored.

# 2 Data Context

## 2.1 The CAMELS Multifield Dataset

The Cosmology and Astrophysics with Machine Learning Simulations (CAMELS) collaboration are responsible for the dataset called the CAMELS Multifield Dataset (CMD). The CMD contains three simulation suites, namely N-body, SIMBA ,and IllustrisTNG. Each suite contains

both 2D maps and 3D mesh grids. Due to technological and time constraints, this project neglects the 3D grids and only concerns itself with the 2D maps. This project used primarily data from the N-body and IllustrisTNG suites (SIMBA is just the same as IllustrisTNG with the omission of magnetic fields) [2, 3]. Each simulated map has a pixel area of $256^2$ which represents a physical periodic area of $(25\ h^{-1}$ Mpc$)^2$ where $h = 0.6711$ at redshift $z = 0$ ($t =$ present time), which is the case for all 2D maps.

### 2.1.1 N-body Suite

The CMD's N-body suite contains 30,000 2D maps of one single field, the total matter density field. Each map is characterised by two cosmological parameters. These cosmological parameters are the matter fraction $\Omega_m$, which characterises the fraction of matter in the Universe, and the amplitude of fluctuations $\sigma_8$, which characterises the smoothness of the matter distribution in the Universe. The suite consists of different sets, with each set having a different systematic approach to simulation. N-body consists of two sets, the LH (Latin-Hypercube) set and the CV (Cosmic Variance) set. The LH set is the largest set in the CMD and contains 1,000 simulations, each with different values of cosmological parameters arranged in a Latin-Hypercube with randomised initial condition seeds. The CV set contains 27 simulations. All simulations in the set share the same cosmological parameters and only differ in their initial conditions defined by a random seed, hence, making the CV set useful for studying cosmic variance. The N-body suite uses the Gadget-III code, which follows the evolution of a self-gravitating collision-less N-body system which makes it good for simulating the large-scale structure of the Universe.

### 2.1.2 IllustrisTNG and SIMBA Suites

The CMD's IllustrisTNG suite contains 195,000 2D maps representing 13 different astrophysical fields. Namely, these fields are gas density, gas velocity, gas temperature, gas pressure, gas metallicity, neutral hydrogen density, electron number density, magnetic fields, magnesium over iron, dark matter density, dark matter velocity, stellar mass density, and total matter density. Each map is also characterised by the same two cosmological parameters as N-body maps, as well as a further four astrophysical parameters. These parameters are $A_{SN1}$, which characterises the galactic wind energy per unit star formation, $A_{SN2}$, which characterises the galactic wind speed, $A_{AGN1}$, which characterises the energy feedback per unit black hole accretion rate and $A_{AGN2}$ which characterises the black hole ejection speed [4]. Like N-body, the IllustrisTNG suite consists of the LH and CV sets but also contains the BE 1P set which contains 27 simulations. The BE 1P set (1 Parameter at a time Butterfly Effect) uses different time steps to show the sensitivity to initial conditions, which without careful monitoring causes divergent results. The IllustrisTNG suite uses the Arepo code [5] which is a finite-volume magnetohydrodynamic algorithm that can successfully simulate a range of astrophysical phenomena, such as both stellar and galaxy formation.

The SIMBA suite is very similar to the IllustrisTNG set as it contains the same sets and breakdowns of sets, with the main difference being the omission of magnetic fields. This means that the SIMBA and IllustrisTNG maps share much resemblance, with any differences being small and arising due to magnetic interactions.

## 2.2 N-Body to Hydrodynamical Simulations

From the CMD data, it is straightforward to see that the IllustrisTNG suite paints a more comprehensive image of the Universe. Each IllustrisTNG simulation returns 13 distinct maps which each represent fields which would be impossible to generate through solely gravitational considerations, such as in the N-body suite. This implies that hydrodynamical simulations are much more effective at generating realistic cosmological data than N-body simulations, which is true. However, carrying out hydrodynamical simulations requires several thousand CPU hours [6, 7] whereas gravitational N-body simulations require significantly less. It is then vital that new ways to generate high-quality, realistic hydrodynamical data should be explored. One such way to do this is to manipulate a very useful aspect of the CMD. The simulations have been carried out such that each IllustrisTNG simulation has a corresponding N-body simulation associated with it as both simulations contain the same values of $\Omega_m$ and $\sigma_8$ and the same initial condition seeds. It is then possible to be able to infer the corresponding 13 hydrodynamical maps from a single N-body total matter density map through means of an Image-to-Image machine learning model. Training such a machine learning model is much less computationally expensive than carrying out a full hydrodynamical simulation (our models never took more than a couple of hours at most to train). The motivation is then that an N-body simulation coupled with an image-to-image generation model is a more computationally efficient way to obtain accurate and useful hydrodynamical maps; much more than executing a full cosmological hydrodynamical simulation. Hence, the primary goal of this project is to construct and train an Image-to-Image machine learning model. The purpose of which is to take total matter density maps from the N-body suite and output corresponding new accurate hydrodynamical maps. The data contained in these output maps should be representative of what would have been generated in the corresponding full hydrodynamical simulation. The described fields are shown in Figure 1.

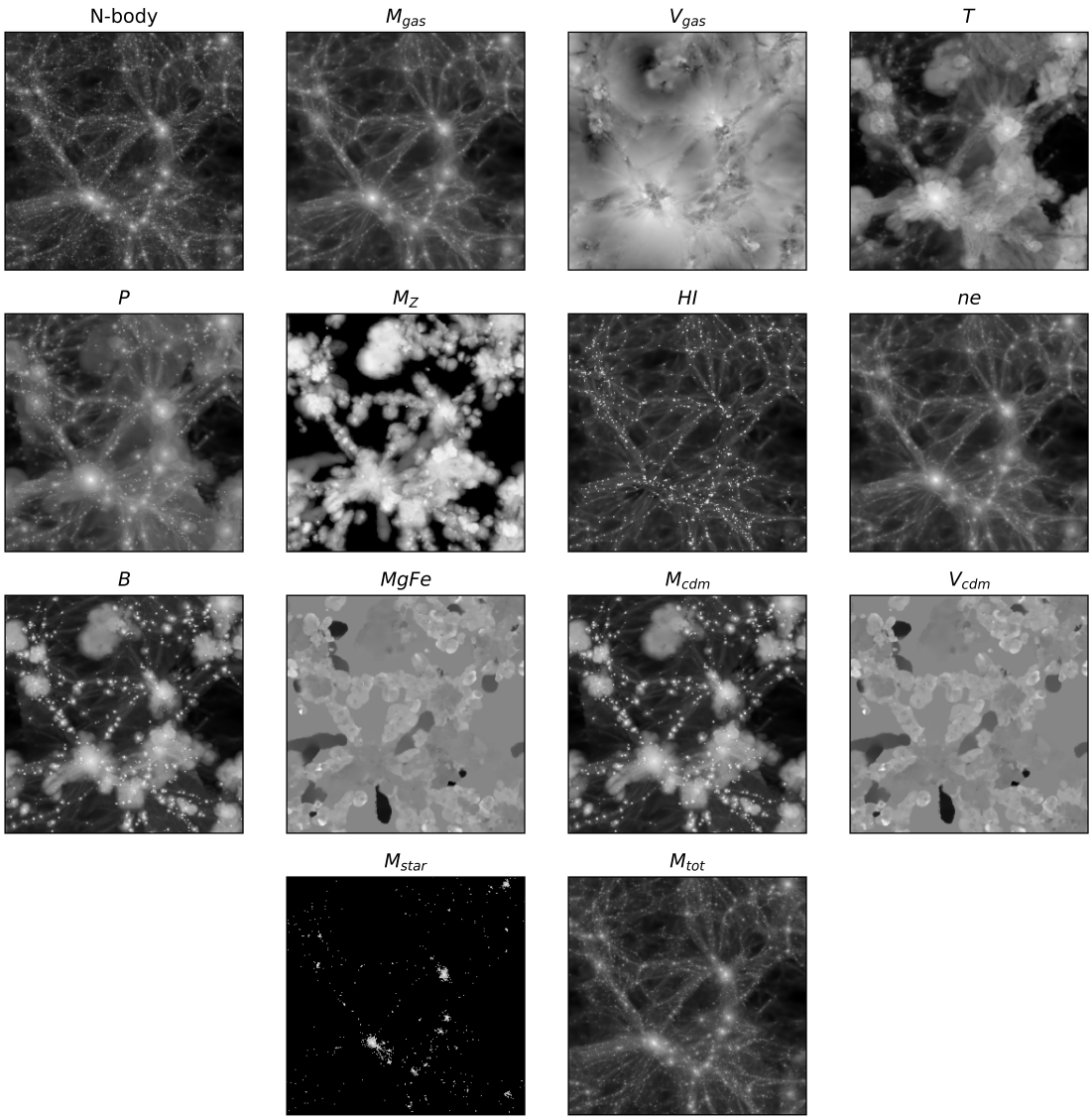



Figure 1: Example N-body and corresponding hydrodynamical fields from the CAMELS IllustrisTNG dataset.

# 3 Computational Theory

## 3.1 Machine Learning Background

Machine Learning models are built using neural networks. These are networks containing layers of neurons, each with its own weights and biases. Subsequently, this intricate network of neurons is used to transform the input data into the output data. The network begins with an input layer which is where the input data is inserted into the network. Next, there are hidden layers where the network transforms the data. Finally the output layer produces the network's outputs data. Mathematically, neurons take an input vector $\mathbf{x}$ and output a scalar

$a(\mathbf{x})$ given by:

$$z = \mathbf{x}^T\mathbf{w},\ a = \Phi(z). \tag{1}$$

where $\mathbf{x}$ encodes the input and $\mathbf{w}$ encodes both the neuron weights and biases. $\Phi$ is known as the activation function with $a$ being the activation.

### 3.1.1 Activation Functions

Activation functions are used to modify the output of a neuron. For example, the Rectified Linear Unit function (ReLU) forces all negative values to become zero. Such functions are chosen based on the context of the data and the desired output of the network. Activation functions are often non-linear and typically have trivial derivatives, which is useful for training the network. Activation functions are also used to prevent saturation. Saturation is a phenomenon that occurs when the neuron output $z$ is too large. In the case of poor choice of activation function, this will cause the activation scalar $a$ to tend to 0. This renders the neuron useless and saturated and can lead to deficiencies when training the network. It is desirable for phenomena such as saturation to be avoided, which can be achieved through a careful choice of the activation function.

In the case that $z$ is too large then an improper choice of activation function can cause the activation $a$ to tend to 0. Consequently, the neuron in question is rendered useless as it will contribute no weight towards the network.

### 3.1.2 Loss Functions

The main objective of a machine learning network is to update its weights and biases such that the output is as closest to the ground truth as possible. This is typically carried out by minimising the loss function, with the loss function given by:

$$J(\theta) = \frac{1}{N}\sum_{i}^{N} \ell(\hat{\mathbf{y}}^{(i)}(\theta), \mathbf{y}^{(i)}), \tag{2}$$

where $J(\theta)$ is the loss function with respect to the network parameters, $\ell(\hat{\mathbf{y}}^{(i)}(\theta), \mathbf{y}^{(i)})$ represents the loss as a function of the model prediction where $\hat{\mathbf{y}}^{(i)}(\theta)$ is the model prediction and $\mathbf{y}^{(i)}$ is the ground truth. Hence, the optimal set of weights for a given loss function are given by:

$$\theta^* = argmin_\theta J(\theta). \tag{3}$$

Finding this optimal set of weights is the primary goal of network "training". In training, the model is fed data periodically and refines its parameters to minimise the loss function. The optimal set of parameters is found using a technique called gradient descent.

### 3.1.3 Back-Propagation

To perform gradient descent, the derivatives of the loss function with respect to the changing network parameters must be calculated. This is done using the back-propagation algorithm, which can be thought of as the chain rule applied to neural networks. The algorithm is as follows:

1. **Forward Pass**: From the input layer, execute the network to compute $\mathbf{z}^{(l)}$ and $\mathbf{a}^{(l)}$ for each layer.

2. **Error at output layer**: Calculate the error associated with each error with:

$$\Delta^{(l)} = \frac{\partial J}{\partial \mathbf{a}^{(l)}} \Phi'(z^{(l)}). \tag{4}$$

3. **Backward pass**: Calculate the error of all preceding layers with:

$$\Delta^{(l)} = (\mathbf{W}^{(l+1)} \Delta^{(l+1)}) \odot \Phi'^{(l)}(\mathbf{z}^{(l)}). \tag{5}$$

4. **Calculate gradients**: Calculate gradients of the loss function with respect to the network parameters with:

$$\frac{\partial J}{\partial \mathbf{b}^{(l)}} = \Delta^{(l)}. \tag{6}$$

$$\frac{\partial J}{\partial W_{kj}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial W_{kj}^{(l)}} = \Delta_j^{(l)} a_k^{(l-1)}. \tag{7}$$

5. **Bias and Weight Update**: Update the parameters of the network to minimise the loss. This is an iterative process which can be defined as:

$$b_j^{(l)} = b_j^{(l)} - \alpha \frac{\partial J}{\partial b_j^{(l)}}. \tag{8}$$

$$W_{kj}^{(l)} = W_{kj}^{(l)} - \alpha \frac{\partial J}{\partial W_{kj}^{(l)}}. \tag{9}$$

Where $b_j^{(l)}$ is the bias to the $j^{th}$ neuron, and $W_{kj}^{(l)}$ is weight connection from the $k^{\text{th}}$ neuron in the $l-1$ layer to the $j^{\text{th}}$ neuron in the $l$ layer respectively. The variable $\alpha$ is the learning rate.

6. **Repeat**: The steps outlined from 1-5 are repeated until the number of predefined epochs or set conditions are met.

It is via this algorithm that machine learning packages identify the gradient of the loss function with respect to both the weights and biases of the network. From this, the network can update the weights and biases to navigate to a minimum in loss function space. This is defined as gradient descent.

### 3.1.4 Optimisation

Optimisation is the process of finding the optimal set of weights and biases of the network such that the loss function is at a minimum. This can be visualised by imagining a 3D surface representing loss function space. This surface will have valleys and peaks. The valleys represent local minima, with the deepest valley representing the global minimum, which is the minimum that provides the best possible optimisation of the network parameters. It is desirable to have the network sit in one of the minima; preferably the global minimum. As aforementioned, the network navigates to these minima using gradient descent, however, there are other factors which need to be considered when optimising.

One of these factors is the learning rate. The learning rate determines the magnitude of the step changes a parameter iteratively takes during training as a function of the loss' gradient (equations 8 and 9). A large learning rate means that the network updates its parameters in larger steps. The advantage of a high learning rate is that the network will see greater change in the loss over the same amount of time, however, due to the large steps the network may continuously overshoot the minima taking longer to converge. It is even entirely possible for the loss to diverge and miss the minima completely, leaving the region of the optimal minima.

On the other hand, a small learning rate will result in smaller steps in parameter updates, and the parameters may take inefficiently long to converge to the minimum. This is a consequence of requiring more iterations to achieve a stable convergence. It is often good practice to employ an "adaptive learning rate" which begins large and becomes smaller as the network begins to approach the minimum loss. This can be implemented by pre-defining a maximum number of epochs with no loss improvement, after which the learning rate decreases by a chosen fraction.

Another factor is a technique known as Stochastic Gradient Descent (SGD) which instead of finding the gradient for all training examples, works in batches of training data instead. This is very useful because it conserves CPU memory and allows for training to be carried out on lower-end machines, at the cost of a slightly less accurate estimation of the gradient. It is common to add a further mechanism to SGD known as momentum. Momentum, much like physical momentum, adds "push" to the model. For example, due to the way gradient descent is carried out, the network will plateau at any local minimum. For a lot of applications, this is fine, but for the most precise models, it is desirable to navigate to the global minimum. This is not possible through regular SGD if the model has become stuck in a local minimum. Momentum allows the network to become unstuck and "push" itself out of the local minimum. Once the network is unstuck it can continue its path to the global minimum.

It is also important to consider the rate by which the parameters are updated for model training. Updating the weights and biases only once the whole training dataset is completed would be very slow and inefficient, particularly if the dataset is large and the images are massive. One solution is to use mini-batches to adapt parameters faster as the model passes through the dataset. The dataset is divided into batches of a predefined size, and the parameters are adapted at the end of each batch. However, this does have the negative effect of being unable to predict the irregularities that may be present in the next batch. The path to minimise the loss will not be completely optimal and it will have a zigzag effect towards the minima. The convention is to use the whole dataset as a batch if it is less than 2000. Each field in the CAMELS dataset has 15,000 images with 256 by 256 dimensions. Therefore it is more optimal to use mini-batches for updating the parameters.

A remedy to reduce this unwanted zigzag motion in the weights and biases plane, when minimising the loss function, is to use an optimiser. An optimiser alters the way the parameters are updated to enhance efficiency.

The zigzag motion means that there is an unwanted motion in another direction that is not in the same path as the trajectory to the loss. This momentum may be considered in the mini-batch gradient descent. By using Exponential Weighted Moving Averages (EWMA), how much weightage older and newer data points receive may be separated. This obeys the equation:

$$V_t = \beta_1 V_{\text{t-1}} + (1 - \beta_1)\theta_{\text{t}} \tag{10}$$

Where $V_t$ is the average calculated at iteration t, $\beta$ is a hyperparameter between 0 and 1,

and $\theta_t$ is the data point value at the current time step.

Expanding equation 10 for a chosen t will show that newer data points have smaller orders of power for $\beta$ than older data points. Since $0 < \beta < 1$, this means that older data points will be given lower weightage compared to newer data points.

In machine learning, it is not desirable for the model to overreact to new data and ignore the older data completely. Therefore to have a smoother trajectory a large $\beta$ is chosen, often this value is $\beta_1 = 0.9$.

One implementation of this is a momentum optimiser improving upon equation 7:

$$W_t = W_{t-1} - \alpha V_{dw_t} \tag{11}$$

$$b_t = b_{t-1} - \alpha V_{db_t} \tag{12}$$

Where W and b in this case are represented as single parameters rather than a matrix of parameters. $V_{\text{dw}}$ and $V_{\text{db}}$ can be expressed as:

$$V_{dw_t} = \alpha V_{dw_{t-1}} + (1 - \beta_1)\frac{\partial J}{\partial W} \tag{13}$$

$$V_{db_t} = \alpha V_{db_{t-1}} + (1 - \beta_1)\frac{\partial J}{\partial b} \tag{14}$$

The consequence of applying equations 13 and 14 is that the average of the movement perpendicular to the direction of the loss' minimum will be approximately zero. However, the plane in parallel with the loss' minimum will have an average greater than zero. Therefore, more weighting will be given to the momentum in the direction that maximises a decrease in the loss function.

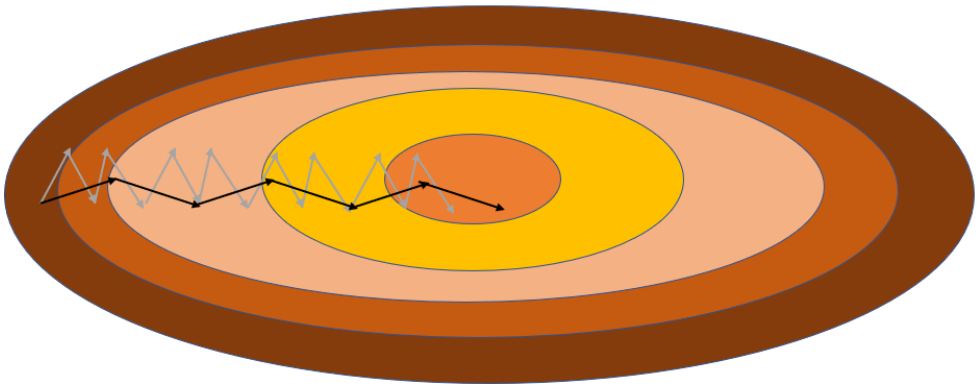


Figure 2: Displays movement towards the loss' minimum and the unwanted trajectory. The grey arrows represent the movement without considering optimisers, and the black arrows show the implementation of optimiser. The pendulum effect of moving up and down perpendicular to the direction of the loss' minimum gives a smaller average and thus reduces the movement in that direction. The opposite is true for the direction towards the loss' minimum.

Another alternative to reducing the movement perpendicular to the plane of the loss' minimum is to use RMSprop. RMSprop stands for Root Mean Square Propagation and expands upon the concept of momentum. The basis by which parameters are updated is now given by:

$$W_t = W_{t-1} - \frac{\alpha}{\sqrt{S_{dw_t} + \epsilon}} \frac{\partial J}{\partial W} \tag{15}$$

$$b_t = b_{t-1} - \frac{\alpha}{\sqrt{S_{db_t} + \epsilon}} \frac{\partial J}{\partial b} \tag{16}$$

Where $\beta = 0.999$ and $\epsilon = 10^-8$, a constant is included to prevent overshooting by the square root term becoming too small. The new parameters $S_{dW}$ and $S_{db}$ are defined as:

$$S_{dw_t} = \beta_2 S_{dw_{t-1}} + (1 + \beta) \left( \frac{\partial J}{\partial W} \right)^2 \tag{17}$$

$$S_{db_t} = \beta_2 S_{db_{t-1}} + (1 + \beta) \left( \frac{\partial J}{\partial b} \right)^2 \tag{18}$$

In effect, considering only one weight and bias, the parameter with the higher gradient $\frac{\partial J}{\partial W}$ or $\frac{\partial J}{\partial b}$, will give a greater corresponding $S_{dw_t}$ or $S_{db_t}$ respectively. Therefore, the update step in that direction will be smaller. The opposite is true for the parameter with the smaller gradient.

It would improve the parameter update efficiency similar to 2. Initially, the grey arrows have a big gradient in the perpendicular plane, but very little gradient in the parallel plane to the minimum. As a consequence, the steps to change the parameters for the parallel direction are greater, and the model will learn faster.

Combining the optimisation algorithm methods mentioned in this section would produce the powerful optimiser known as Adam. Adam stands for Adaptive Learning Rate Optimisation Algorithm. The update for weights and biases can then be expressed as:

$$W_t = W_{t-1} - \frac{\alpha V_{dw_t}}{\sqrt{S_{dw_t} + \epsilon}} \tag{19}$$

$$b_t = b_{t-1} - \frac{\alpha V_{db_t}}{\sqrt{S_{db_t} + \epsilon}} \tag{20}$$

The implementation of weight decay to Adam changes the optimiser to Adam with Weight Decay (AdamW). This obeys an update algorithm of:

$$W_t = W_{t-1} - \frac{\alpha V_{dw_t}}{\sqrt{S_{dw_t} + \epsilon}} + \lambda W_{t-1} \tag{21}$$

$$b_t = b_{t-1} - \frac{\alpha V_{db_t}}{\sqrt{S_{db_t} + \epsilon}} + \lambda b_{t-1} \tag{22}$$

Where $\lambda$ is a weight decay term, a small positive value generally ranging between $10^{-4}$ to $10^{-6}$. The aim of this additional term is to reduce overfitting to the training data and prevent poor generalisation to unseen data.

All of the above factors in this section are combined into what is known as an optimiser which drives the training of the network. Most machine learning packages feature a host of different optimisers suited for different tasks.

## 3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are the most useful type of neural network for image-based machine learning. CNNs make use of convolutional filter layers to transform an input image for feature extraction. Discrete convolution is defined by:

$$s(t) = \sum f(\tau)g(t-\tau). \tag{23}$$

where $f$ is the input image, $g$ is the filter kernel and $s(t)$ represents the feature map. Each layer in a CNN represents a convolution operation, with network weights representing kernel elements. There are various examples of convolutional layers. These include padding, which is where the input image has extra layers of pixels added to its edges to avoid errors when the kernel is applied to pixels at the boundary of the image. Max pooling, is where an image is down-sampled by taking the largest pixel value in a pixel neighbourhood and inserting that into a smaller image. A final example is batch normalisation. Convolution can cause the values of elements in a feature map to grow quickly. This will cause training to be slow as the weights will take a long time to converge to their optimal value. The batch normalisation layer is used to renormalise the input feature map such that the mean pixel value is 0 and the variance of pixel values is 1. There are other examples of convolutional layers but these are the most widely used.

CNNs are currently a popular type of machine learning model and have many practical applications. One such application is object detection and segmentation [8] by which a model is able to identify features of an image and segment them from the original image. This is useful especially in roadside cameras, as they are able to automate the process of identifying the number plates of speeding cars. Other uses of CNNs include image-to-image applications and text-to-image applications. Impainting is an example of an image-to-image application by which features can be added or removed to an input image in such a way that makes the output image look realistic [9]. Text-to-image models have also been popularised, particularly by the release of the Stable Diffusion model which can take text inputs and generate a full image based on the text input. It may also take an input image coupled with a text prompt and generates an output that is representative of both the input image and text prompt. Stable Diffusion is a specific CNN constructed using the (latent) diffusion model.

## 3.3 Probablistic Diffusion Models

Diffusion models are a recent, powerful form of CNN that applies the concept of physical diffusion to perform image-to-image tasks. Diffusion models have two parts, a forward and backward process.

### 3.3.1 Markovian Forward Process

For the forward process, a Markov Chain must first be defined. A Markov Chain is given as a chain of events in which each event is determined only by the previous event. In this case, the Markov Chain is periodically adding a small amount of Gaussian noise determined by a variance schedule given by $\beta_0, \beta_1, ..., \beta t$ where each $\beta$ is slightly larger than the previous. According to the Markov Chain, the state $x_t$ is determined by the previous state $x_{t-1}$. Hence, the probability $q(x_t \mid x_{t-1})$ of going from one state to another is needed. The subsequent noisier image is generated by sampling a small amount of noise from the Gaussian Distribution $\mathcal{N}$. The noise

sampled from $\mathcal{N}$ depends on the mean and standard deviation. The mean depends on $\beta_t$ and $x_t$. Hence, the probability can be defined as:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I). \tag{24}$$

The probability that a Markov Chain occurs from $x_1$ to the final state $x_T$ based on an initial state $x_0$ is:

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}). \tag{25}$$

Currently, equation 25 is not very computationally efficient, so it is good practice to reparamaterise equation 25 such that $a_t = 1 - \beta_t$, $\bar{a}_t = \prod_{s=1}^{T} a_s$. So 31 becomes:

$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{a}_t}x_0, (1-\bar{a}_t)I). \tag{26}$$

so we can express $x_t$ as a linear combination of $x_0$ and a noise variable $\epsilon$ where $\epsilon\ \mathcal{N}(0, I)$:

$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon \tag{27}$$

This is a useful reparameterisation as it allows values of $a$ and $\bar{a}$ to be pre-computed and hence the noised image $x_t$ can be obtained at any timestep $t$ given solely the original image $x_0$ .This is the forward process used by a Denoising Diffusion Probabilistic Model (DDPM).

### 3.3.2 Backward Process

The backward process is where machine learning is used. The main objective of the backward process is to "undo" the noise from the forward process in such a way that the final denoised image represents the desired output. There a multiple models which can do this. Namely, U-Net [10], Denoising Diffusion Probabilistic Model (DDPM) [11] and Denoising Diffusion Implicit Model (DDIM) [12].

### 3.3.3 Non-Markovian Forward Process

The length of $T$ of the forward process is important because a large $T$, such as $T \geq 1000$, allows the reverse process to be close to a Gaussian so that the generative process modelled with Gaussian conditional distributions becomes a good approximation. However, all the T iterations must be performed sequentially in the Markov Chain which leads to the sampling step of DDPMs being much slower than other generative models. Therefore, alternative non-Markovian forward processes should be used to reduce the number of iterations of the generative model. Consider a family of inference distributions indexed by real vector $\sigma \epsilon \mathbb{R}^{T}_{\geq 0}$:

$$q_\sigma(x_{1:T} \mid x_0) := q_\sigma(x_T \mid x_0) \prod_{t=2}^{T} q_\sigma(x_{t-1} \mid x_1, x_0) \tag{28}$$

where $q_\sigma(x_T \mid x_0) = \mathcal{N}(\sqrt{\alpha_T}x_0, (1-\alpha_T)I)$ and for all t $> 1$:

$$q_\sigma(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\sqrt{\alpha_{t-1}}x_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \frac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1-\alpha_t}}, \sigma_t^2 I) \tag{29}$$

where the $\alpha_{t-1}$ represents the noise level and $\sqrt{1-\alpha_{t-1}-\sigma_t^2}$ represents the signal level of the noisy image. This means that the forward process here is no longer Markovian since each $x_t$ could depend on both $x_{t-1}$ and $x_0$. The magnitude of $\sigma$ controls how stochastic the forward process is. In the special case where $\sigma \to 0$, $x_{t-1}$ becomes fixed as long as $x_0$ and $x_t$ is observed and the signal level of the noisy image becomes $\sqrt{1-\alpha_{t-1}}$. This is the forward process used by a Denoising Diffusion Implict Model (DDIM).

### 3.3.4 Diffusion Schedule

Consider the diffusion process starting at T = 0 and ending at T = 1000. This variable is continuous. A diffusion schedule needs to be defined to determine the noise and signal levels of the noisy image at a given T. The diffusion model starts with a image of random Gaussian noise for $x_t$ with means and variances of

$$q_\sigma(x_T \mid x_0) = \mathcal{N}(\sqrt{\alpha_T}x_0, (1-\alpha_T)I) \tag{30}$$

For the Gaussian noise, the noise rate and signal rate can be interpreted as the standard deviation of their components in the noisy image, while the squares of their rates can be interpreted as their variance. This means that the rates will always be set so that their squared sum is 1, meaning that the noisy images will always have unit variance, just like its un-scaled components.

The variable T is continuous so sampling time can be changed so that the number of sampling steps can be changed at inference time. This example only implements the deterministic sampling procedure from DDIM [13].

### 3.3.5 Training in Practice

When sampling at each step we take the previous estimate of the noisy image and separate it into image and noise using our network. Then we recombine these components using the signal and noise rate of the following step using

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I). \tag{31}$$

This equation can be used to derive the function for the network to predict the noise $\epsilon$ in

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha}}(x_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\epsilon_\theta(x_t, t) \tag{32}$$

where the noise $\epsilon$ is measured using the loss function Mean Absolute Error (MAE) for the diffusion model in this experiment.

## 4 Parameter Inference

A potential application of the CMD outlined in the Data Context section is parameter inference, which is the process of estimating the cosmological/astrophysical parameters associated with a set of previously unseen 2D maps or 3D grids. Parameter inference is desirable as it provides a means to test and validate cosmological models. For instance, comparing the predicted parameters obtained from simulations with observed data can give insight into the agreement or disparity between cosmological models and the real Universe. This validation process helps refine and improve our understanding of the underlying physical processes driving the evolution

of the Universe. In addition to validating models, parameter inference is an important tool for constraining the values of cosmological and astrophysical parameters.

## 4.1 Method

As previously mentioned in the Data Context section, there are two cosmological parameters associated with each N-body map, with the hydrodynamical maps also having a further four astrophysical parameters that govern the processes of supernova and black hole feedback. The first model outlined in this section was initially trained to infer the two cosmological parameters, $\Omega_m$ and $\sigma_8$, as an exercise in attempting to extract cosmological information from hydrodynamical maps while marginalising over astrophysical effects (at the field level). Further models were trained on maps from both the SIMBA and IllustrisTNG suites. As well as training on just cosmological parameters, the models were extended to perform inference on just the four astrophysical parameters, as well as all six parameters in a single go. The performance of all models is outlined in the Results section.

### 4.1.1 Input Pipeline and Pre-Processing

The models outlined in this section were developed and implemented in Python using the TensorFlow deep learning library. Other packages were used where appropriate, for instance 'numpy' was used for data handling and manipulation, 'matplotlib' was used for plotting operations, and 'sklearn' was used for splitting data into training, validation and testing datasets, among others. All models were trained on the 15,000 total matter density field maps (2D) available in the CMD, each of size (256x256) pixels. Half of the models were trained on the SIBMA suite (gravo-hydrodynamical), and the other half on the IllustrisTNG suite (gravo-magneto-hydrodynamical). Three models were developed for each suite, one for the cosmological parameters, one for astrophysical, and one for all six parameters to compare performance based on different parameter inputs. The aim of all of these models is to learn the dependencies between the input maps and their associated truth labels (parameters) to achieve parameter inference on previously unseen maps during testing.

Pre-processing was performed on all input image arrays prior to training. The inverse hyperbolic sine (i.e., arcsinh) function was applied element-wise to normalise the images. The mean of this dataset was then subtracted from each image pixel (element-wise) such that all values were centred around zero. All images were greyscale (single-channel) both before and after normalisation. The dimensionality of the normalised image arrays were adjusted such that they were compatible with TensorFlow. After normaliation, 25% of the data was reserved for testing after the training and validation procedures.

### 4.1.2 Model Architecture

The model developed by the CAMELS project team [3] was used as the foundation for the parameter inference model outlined in this report. The architecture comprises of a set of convolutional layers, each immediately followed by a batch normalisation layer and a Leaky ReLU activation function. Multiple sets of convolutional layers are implemented, followed by a dense layer prior to the final output layer of size 6, which represents the six values of the target parameters. A visual representation of this architecture can be seen in Figure 3.

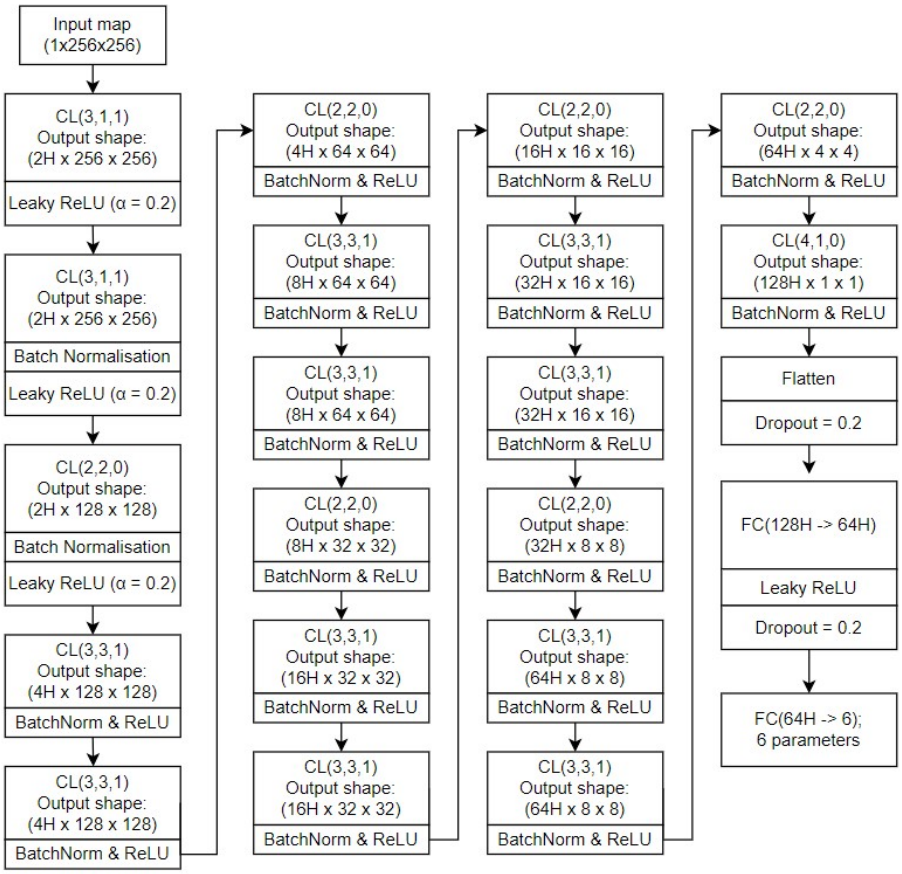



Figure 3: Visual representation of the Parameter Inference model architecture. The model takes greyscale images of size 256x256 as input. The notation CL(3,1,1) denotes a Convolutional Layer with attributes (K,S,P), where K is the kernel size (3 in this case), S is the stride (1 in this case) and P is the padding (1 in this case). Each CL is followed by Leaky ReLU activation function with a negative slope coefficient, $\alpha$=0.2, and then a batch normalisation operation for improved stability. The output shape of each layer is denoted by (C x W x H), where C is the number of channels, W is image width and H is image height. All of the activation functions are Leaky ReLUs, unless specified otherwise. CLs are applied gradually to down-sample the input and extract the salient features, after which it is flattened and converted to an output vector of size 6.

The convolutional layers (CLs) shown in Figure 3 systemically apply learned filters to the input images to create feature maps. These maps effectively summarise the presence of the features in the input images. The notation CL(3,1,1) denotes a CL with attributes (K,S,P), where K is the kernel size (3 in this case), S is the stride (1 in this case) and P is the padding (1 in this case). Many of these layers are stacked together so that the model is able to extract different types of features. For instance, CLs near the start of the network are able to learn lower-level features such as lines and textures, whereas CLs deeper in the network learn more abstract features such as shapes or objects (in this case, galaxy clusters and filaments). The number of CL filters applied increases as the network becomes deeper to allow it to learn more complex and intricate features found in the input maps. The gradual increase in filters as the network progresses is done to help the network learn a hierarchy of features (lower-level in the

beginning, higher-level near the end), in turn allowing it to capture and encode the relevant information for this task. Also, it can be seen in Figure 3 that the size of the input is gradually down-sampled due to the increases in stride in some of the convolutional layers.

After each CL is applied, a batch normalisation operation is performed to normalise the activations of each feature map such that they have a mean of zero and a variance of unity. This helps stabilise and speed up the training process since internal covariate shift is reduced, which is the natural change of the weights and biases in the preceding layers that results in changes in the input distribution to subsequent layers.

After batch normalisation comes the Leaky Rectified Linear Unit (ReLU) activation function, which aims to add non-linearity to the network. The function leaves all positive values of the input unchanged, and introduces a small negative slope to ensure that all negative values are transformed but are non-zero (as is the case in the traditional ReLU). This slope is controlled by the $\alpha$ argument, which is set to a constant 0.2 throughout the network.

At the end of the set of convolutional layers, the output feature map is flattened into a single-dimension vector for processing in the fully-connected layers (FCs). The FCs reduce the dimensionality even further until the output layer simply returns the inferred (predicted) target parameters. At the end of the flatten operation and the first FC, dropout regularisation is implemented, which is a technique used to prevent overfitting and improve generalisation performance. A dropout of 0.2 is used, which ensures that 20% of the neurons are deactivated at random.

### 4.1.3 Hyperparameters and Training Procedure

All parameter inference models share the same set of hyperparameters, which include the learning rate ($\alpha$), weight decay ($\lambda$), convolutional filter multiplier (H), number of epochs and batch size. For all models, the initial $\alpha$ was set to 0.001, the initial $\lambda$ was set to 0.0005 and the value of the multiplier, $H$, was set to 5. These values were established in previous works and were determined by hyperparameter fine-tuning, and were therefore left untouched. During training, the validation loss is monitored such that if it were to plateau for a threshold number of epochs (the 'patience' value), the $\alpha$ would be reduced by a factor of 0.5. This is referred to as adapting $\alpha$ scheduling. This iterative reduction in learning rate allows the model to take smaller steps during parameter updates, allowing for a more stable convergence towards an optimal solution if the loss function is oscillating and failing to converge. A smaller $\alpha$during training essentially allows for a slower and more controlled navigation of the loss function space, particularly if the model is stuck in a local minima epoch after epoch. A large $\alpha$ may cause drastic movements in loss function space and potentially overshoot the optimal solution, which is why it is important to monitor the validation loss to able to intervene in real-time and reduce the $\alpha$ to avoid such overshooting. This is most important during the latter stages of training when the model is close to converging. The epoch patience value for all models was set to 7, and was determined by means of trial-and-error.

The loss used for these models is the Mean Square Error (MSE) function defined as,

$$\mathcal{L}_{\text{MSE}}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{33}$$

where $y$ represents the ground truth, $\hat{y}$ represents the predicted values and $n$ is the sample size. This is a commonly used loss function for tasks such as inference, though many other functions exist.

Network optimisers seek to minimise the loss function and update the model's parameters (weights and biases) during training to guide it towards convergence to a global minimum in loss function space. The inference model makes use of an AdamW optimiser which is an optimisation algorithm that is an extension of Stochastic Gradient Descent (SGD) that adapts the learning rate based on the first and second gradient moments. It incorporates L2 regularisation by means of weight decay which penalises high weight values to prevent overfitting. Unlike the Adam optimiser, AdamW updates the weights using weight decay first, before incorporating the Adam update step with the adaptive learning rate. This allows better control over the magnitude of weight updates. The optimiser works in conjunction with the adaptive $\alpha$ scheduling to better control over the learning rate. While the optimiser already adapts the $\alpha$ based on gradient statistics, the $\alpha$ scheduling acts as a secondary mechanism to further fine-tune the $\alpha$ based on the behaviour of the validation loss.

All models are first trained for 50 epochs using the same initial hyperparameters. If a model shows potential after evaluating its performance, the starting $\alpha$ is updated and training continues for at least another 50 epochs until no further gains in performance are observed. An early stopping procedure is implemented for this purpose during training, as well as to help prevent overfitting. For the initial 50 epochs of training, this procedure is only engaged after the first 20 epochs such that the model is allowed some time to stabilise. The early stopping is set up such that if the adaptive $\alpha$ scheduling produces no significant improvement in validation loss even after reducing the LR, the training terminates after 15 epochs.

Data augmentations were applied to all images in the form of rotations by $-\frac{\pi}{2}$, $\frac{\pi}{2}$, $\pi$, and $\frac{3\pi}{2}$ as well as both horizontal and vertical flips. These operations are implemented at random to further mitigate against overfitting. Applying flipping and rotations to certain maps in the training set changes the positions of the salient features in these augmented maps, such that the model is less likely to simply memorise the positions of features and actually learn them. During this stage, the training dataset is further split into 75% actual training data and 25% validation data into batches of 64 and 32, respectively. Training was performed using Google Colab on an NVIDIA A100 GPU for all models.

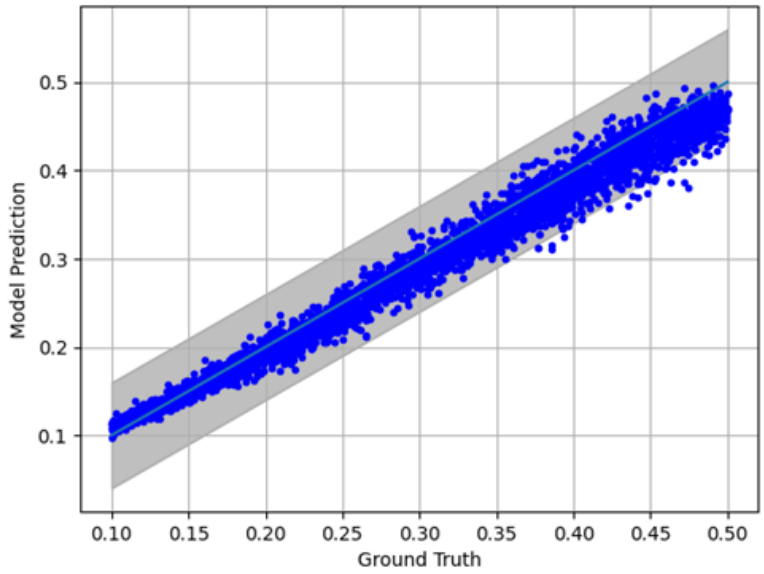
## 4.2 Results

The performance for all six parameter inference models in terms of the relative error on the predicted parameter values can be seen in Table 1. The best performing models appear to be the SIMBA and IllustrisTNG networks trained to only infer the cosmological parameters. Models trained on only the astrophysical parameters or all six parameters seem to perform worse overall.

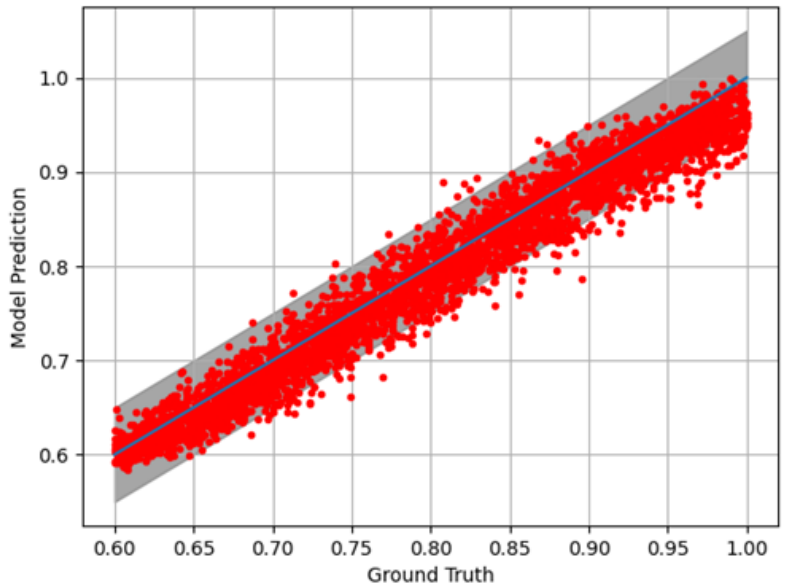
| - | $M_{tot}$ Illustris Cosmo. Params Only | $M_{tot}$ Illustris Astro. Params Only | $M_{tot}$ Illustris All Params | $M_{tot}$ SIMBA Cosmo. Params Only | $M_{tot}$ SIMBA Astro. Params Only | $M_{tot}$ SIMBA All Params | $M_{tot}$ Nbody SIMBA Cosmo. Params Only |
|---|---|---|---|---|---|---|---|
| $\Omega_m$ Mean Fractional Error | 4.8% | - | 22.7% | 4.0% | - | 11.3% | 4.9% |
| $\sigma_8$ Mean Fractional Error | 2.4% | - | 7.9% | 2.2% | - | 7.0% | 2.6% |
| $A_{SN1}$ Mean Fractional Error | - | 60.4% | 90.4% | - | 115.2% | 94.9% | - |
| $A_{AGN1}$ Mean Fractional Error | - | 112.5% | 103.1% | - | 123.0% | 96.8% | - |
| $A_{SN2}$ Mean Fractional Error | - | 34.5% | 37.1% | - | 27.0% | 28.5% | - |
| $A_{AGN2}$ Mean Fractional Error | - | 36.7% | 37.6% | - | 34.3% | 34.2% | - |

Table 1: Table to show the results of all trained parameter inference models, expressed as a mean fractional error, $< \delta x/x >$. Models that are trained on only the cosmological parameters seem to perform the best. The best results appear to be the model trained on the cosmological parameters of the hydrodynamical $M_{tot}$ SIMBA maps, yielding a 4.0% and 2.2% fractional error on $\Omega_m$ and $\sigma_8$, respectively. The overall worst-performing model appears to be the one trained on all parameters using the hydrodynamical $M_{tot}$ IllustrisTNG dataset. The best-performing parameter appears to be $\sigma_8$ which has errors of less than 8% across the board, whereas the worst parameter to train appears to be $A_{AGN1}$.

For the best performing model, plots of model prediction against truth can be seen in Figure 4. These show the performance of the model trained on the hydrodynamical $M_{tot}$ SIMBA dataset for the $\Omega_m$ parameter in sub-figure (a) and the $\sigma_8$ in sub-figure (b). A grey error region is overlaid on the plots which corresponds to $\pm$4% and $\pm$2% relative error, respectively. Performance plots for models trained on only the cosmological parameters using the hydrodynamical IllustrisTNG and Nbody SIMBA datasets look very similar and were therefore omitted for the sake of brevity. The significance of these results is covered in the Discussion section.




(a) Above plot shows the $\Omega_m$ performance of the model trained on the hydrodynamical $M_{tot}$ SIMBA dataset, which has an overall mean relative error of 4.8% for this parameter. A $\pm$4% relative error region is overlaid on the plot for clarity. The model shows better inference for lower values of the parameter, though its performance degrades for higher values. The model also seems to under-predict the higher values, as seen by the larger quantity of points under the line as well as points outside of the error region.




(b) Above plot shows the $\sigma_8$ performance of the model trained on the hydrodynamical $M_{tot}$ SIMBA dataset, which has an overall mean relative error of 2.4% for this parameter. A $\pm$2% relative error region is overlaid on the plot for clarity. Just as for the performance plot for $\Omega_m$, under-prediction seems to take place for higher values.

Figure 4: Performance plots of model prediction against truth for the model trained on the hydrodynamical $M_{tot}$ SIMBA maps.

## 4.3 Discussion

Overall, the models perform well for the parameter inference task, particularly if these models are trained solely on the cosmological parameters. The best model outlined in this report achieved a 4.0% and 2.2% fractional error on $\Omega_m$ and $\sigma_8$, respectively. Previous work [14] achieved consistent error values of 3.4% and 2.4% for $\Omega_m$ and $\sigma_8$, as well as 38% for the $A_{SN1}$ and 17% for the $A_{SN2}$ astrophysical parameters, respectively. Therefore, whilst the model outlined in this report significantly underperforms in inference for the astrophysical parameters, it is on-par with previous literature for the cosmological parameters, in fact marginally outperforming in the inference task for the $\sigma_8$ parameter. However, it is important to note that these results are likely lower bounds since real data may contain noise and other sources of random/systematic error.

While the model seems to have no issue inferring the values of $\Omega_m$ and $\sigma_8$ with high accuracy, there seems to be significant underperformance for $A_{SN1}$ and $A_{AGN1}$, with error values ranging widely between 50-125% for the different models. Inference of the $A_{AGN1}$ parameter seems to fare the worst, with most error values being close to or over 100% error. This underperformance is most likely not due to the size of the training sample, as previous works use the same hydrodynamical dataset for training and testing. It is most likely due to either differences in model architecture or choice of hyperparameters. One major reason could be the choice of loss function - previous works define a custom loss function which is much more suited for this task, whereas the model outlined in this project uses a standard MSE loss. Experimentation in choice of loss function and extensive hyperparameter fine-tuning (for example by means of the Optuna package) would be necessary to match the astrophysical inference performance of previous works, though would require significant computational resources.

In any case, astrophysical parameters are inherently more difficult to infer than their cosmological counterparts since they manifest as much more complex and intricate features in the maps, making them much harder to learn. It is easier for a network to extract salient features in the large-scale structure of each map since that is well-correlated with the $\Omega_m$ and $\sigma_8$ parameters. This is not the case for $A_{AGN1}$ which represents the efficiency of black hole feedback or $A_{SN1}$ which represents the efficiency of supernova feedback, as they have much more complex dependencies between value and what is manifested in each map for the network to learn.

As seen in Figure 3a, the model appears to marginally under-predict $\Omega_m$ for higher values. Maps with higher values of $\Omega_m$ indicate a higher overall matter density in the simulated region, which is manifested physically as more pronounced clustering of matter into galaxy clusters/filaments and more pronounced large-scale cosmic structures. The matter density tends to organise itself more readily into elongated filaments that connect galaxy clusters and cosmic voids, producing a web-like pattern. It could be that this increase in complexity and intricacy in the matter density distribution results in additional features and patterns for the network to learn that are not as prominent in maps with lower values of the parameter. As such, a possible explanation for the model to tend to under-predict values for these maps is that the network simply struggles at successfully extracting all the useful features from such maps during training and therefore performs worse during testing. If there are simply more complex features present in a map with a high $\Omega_m$ value, some of them are bound to be irrelevant or non-informative, resulting in counterproductive feature extraction.

If this is the case, a possible solution could be to increase the depth of the network in attempt to aid the model in extracting more features from the maps, though this would introduce the risk of overfitting, which occurs when a model becomes too complex and begins

memorising the training data as opposed to correctly learning the generalisable patterns and features. Provided that there are effective preventative measures against overfitting such as more strategic applications of regularisation techniques (e.g., dropout layers, batch normalisation operations) as well as more fine-tuned early stopping procedures, the model could be expanded to better deal with the increased complexity of maps with higher values of $\Omega_m$.

The model outlined in this project already incorporates many features that attempt to maximise its performance and minimise the chance of overfitting, such as data augmentation, early stopping and adaptive learning rate scheduling. Improving the model further by means of fine-tuning the architecture is computationally expensive and time-consuming, especially for a large search space as would likely be the case with this model. The search space could include simple hyperparameters such as learning rate and batch size, as well as more specific changes to the convolutional layers themselves (i.e., the number of filters for each layer, the kernel size etc.). A comprehensive search could be performed using packages such as Optuna, but due to the lack of the computational resources and time available for this project, this task is left open for investigation in further works.

The model also seems to under-predict the values of $\sigma_8$, especially for higher values of this parameter where the trend breaks down and the under-prediction is most severe, as seen in Figure 3b. High values of this parameter represent a high amplitude/strength of matter density fluctuations on large scales which, similarly to high values of $\Omega_m$, manifest in more pronounced variations in matter density from one region to another. The explanation for why this occurs may be similar to the one given for $\Omega_m$ - more variations in matter density may result in greater quantities of salient features for the model to learn which it cannot deal with, and some of the features the model does end up learning may not even be useful. Also, it may simply be the case that the overall relationship between the input maps and the $\sigma_8$ parameter are more nuanced than for $\Omega_m$ and is therefore harder to learn, or that there may have a greater dependency on the initial conditions or growth of structure over cosmic time rather than what is represented in each map at $z = 0$. More sophisticated architectures or the implementation of attention mechanisms may improve the model's ability to capture the more subtle dependencies, although as mentioned before this is computationally expensive and is left for further investigation in other works.

# 5 Image-to-Image Generation

### 5.0.1 U-Net Architectures

The name U-Net derives from the shape of the architecture, as it resembles a symmetrical 'U'. The architecture was first proposed by Olaf Ranneberger et Al. [15], for the purposes of biomedical image segmentation which involves the classification of different pixels in an image. In the past, this method has been used for various applications such as pixel-wise regression (e.g., in pansharpening), learning dense volumetric segmentation from sparse annotation, and image-to-image translations [16, 17, 18].

For this project, a Vanilla U-Net model was adapted from its original use in image segmentation to act as an image-to-image synthesis model. In essence, the model is a convolutional neural network that has three distinct sections: contraction, bottleneck, and expansion. These sections work as blocks to successively down-sample and up-sample the initial image to learn its features and their associated spatial information. The contraction phase is used to emphasise the features present in the N-body maps all the way down to a minimum bottleneck dimension.

The expansion phase is then used to regain the spatial information of these features. Additionally, five different types of layers may constitute a block which then form these three sections. These include Convolution Layers, Max Pooling, ReLU Activation layers, Concatenation layers, and up-sampling layers.

In total, there are four blocks in the contraction section. Each block starts by receiving an input and passing it through convolutional layers with a size (3x3) kernel twice, followed by a ReLU Activation layer for neuron activation, and finally a size (2x2) Max Pooling layer. The convolutional layers employ 'same' padding and a stride length of 2. The output of this process is known as a feature map. The number of output feature maps naturally depends on the number of filters applied to the input. The new dimensions of the image due to convolution obeys:

$$n_{out} = \frac{n_{in} + 2p - k}{s} + 1, \tag{34}$$

Where $n_{out}$ is the output dimension, $n_{in}$ is the number of input dimensions, k is the convolutional kernel size, p is the convolutional padding size, and s is the stride length.

The input dimensions are then reduced by Max Pooling, which divides the input into regions of (2x2). The maximum pixel value out of those individual divided regions is then chosen as a singular output. This may be seen in the diagram below:

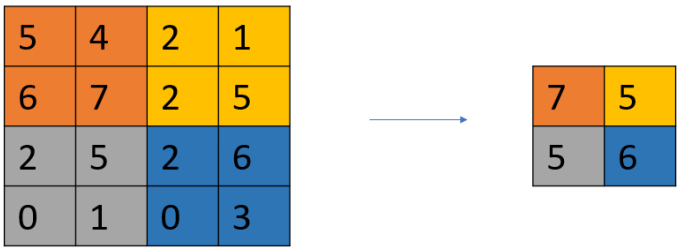



Figure 5: Example of Max Pooling by a kernel size of (2x2), the colours signify the different regions divided into parts to be operated upon.

Figure 5 shows that the dimension of the input is reduced, however important and prominent features should still be retained. The intent of this is to reduce the spatial information while still allowing the model to learn any complicated features that may be present.

After the entry block, there are three remaining contraction blocks with filter numbers of 64, 128, and 256 and each block still contains a max pooling layer. The consequence of this is that feature maps are doubling but the spatial dimension is reducing. Consequently, the model should be learning what complicated features are present in the input but the spatial information of where it is present is reduced. It should also be noted at the end of each contraction block, before the Max Pooling, corresponding cropped inputs are sent directly across to the expansion section (Skip Connections). The purpose of this will be further explained in the expansion section.

Eventually, the contraction phase finishes and max pooling is no longer applied at the bottleneck, which is the minimum dimension of the initial input. There is still an application of convolution and ReLU layers twice in this phase. The minimum shape of the input becomes (28x28) with a total of 1024 feature maps.

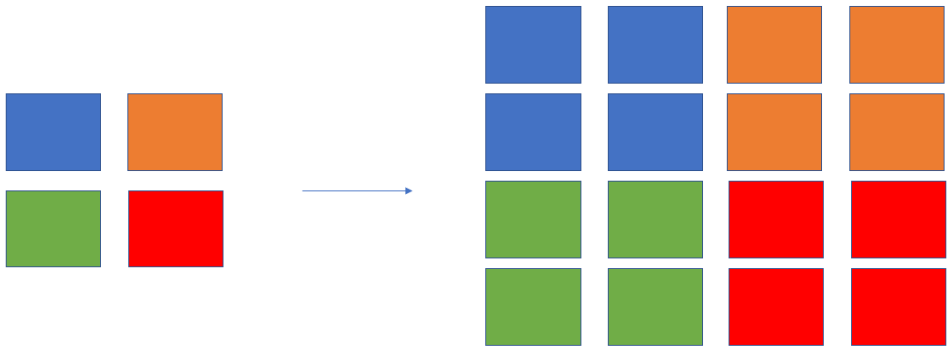


Figure 6: Example of up-sampling where the rows were duplicated twice first and the columns are then duplicated twice afterwards.

Next, the expansion section will take place. The bottleneck is up-sampled to increase in dimension and is concatenated with the corresponding skip connection of the contraction section. This is to regain the spatial information of the features lost during the contraction, and concatenating cropped images before max pooling re-enforces this. The following steps of a block are identical to the contraction section of the model, except for up-sampling at the end of each block instead of max pooling. The up-sampling occurs by duplicating the number of rows and columns of pixels by two for each.

Once the image has been expanded back to the final block, a further (1x1) convolution takes place. This will convert the image into the correct amount of output feature channels to produce a prediction. In the specific case of this model, its output will be of size (256x256x1). This is the target hydrodynamical map, which will be used to compare with the ground truth map. From here, the steps outlined in back-propagation take place until the loss is minimised or other predefined conditions are met.

The architecture for a Vanilla U-net is shown in 7.

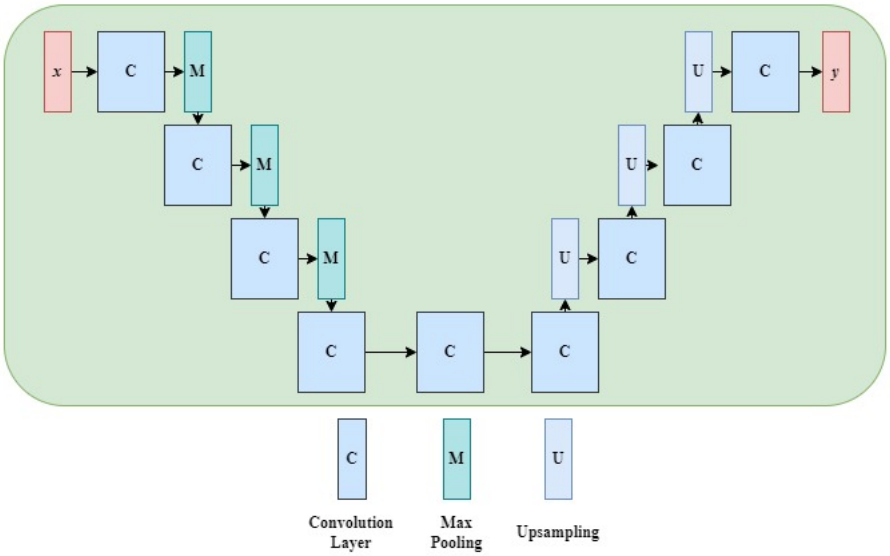



Figure 7: Architecture of Vanilla U-net. The model down-samples and up-samples through image sizes: (256x256), (128x128), (96x96), (64x64) and (32x32) (remove skip connections and replace avg. pooling to max pooling).

For this project, this Vanilla U-net model has been tested alongside other forms of U-net architectures, such as the Residual U-net (RU-net). The RU-net works similarly to a Vanilla U-net with down-sampling and up-sampling blocks but also has additional convolutional layers in-between, which are called residual blocks. The structure of a residual block used for the RU-net is shown in 8. The convolutional layers represent a single convolution with the number of filters depending on where the residual block is placed in the RU-net (i.e., the number of filters for an image of size 256x256 is different from one of size 128x128). The inclusion of residual blocks has been shown to improve feature extraction in U-nets used for image-to-image generation [19]. Images that are passed through a skip connection of a block will bypass the convolutional layers. These are then concatenated channel-wise to the images that have passed through the convolutional layers normally. This helps the U-net retain the original features of the input image, ($x$), after it has passed through the convolutional layers.

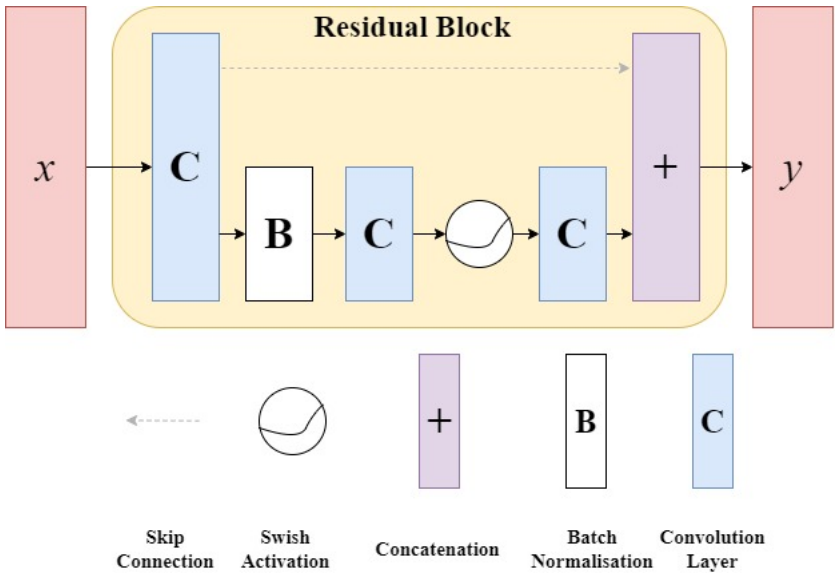



Figure 8: Diagram showing the structure of a residual block used in RU-net. The red blocks depict the input image, $x$, and the output image, $y$, of the residual block. Diagram designed using diagrams.net.

The architecture of the RU-net and how the residual blocks are incorporated can be seen in Figure 9.

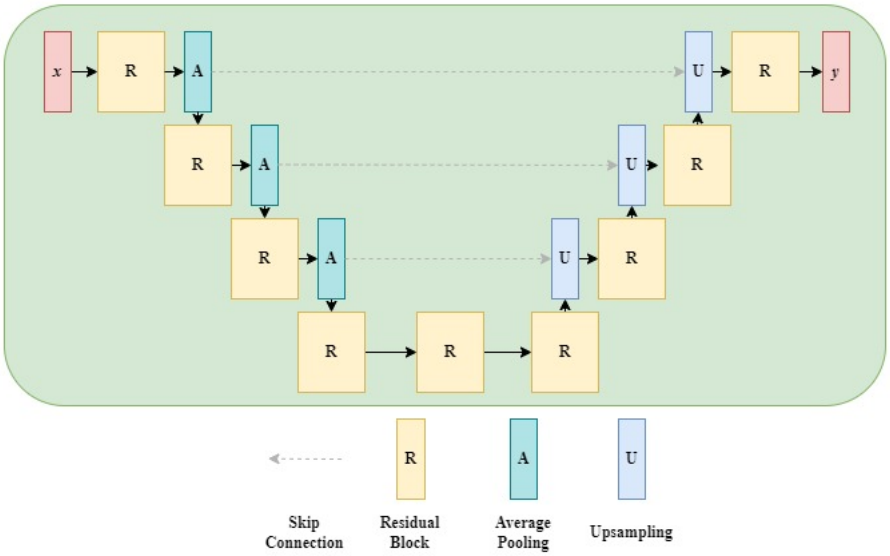



Figure 9: Architecture of Residual U-net (RU-net). The model down-samples and up-samples through image sizes: (256x256), (128x128), (96x96), (64x64) and (32x32). After each Residual Block, and average pooling operation is applied. Diagram designed using diagrams.net.

As an additional component to the Vanilla U-net, the RU-net also incorporates global skip connections where the image is stored separately at each down-sampling layer (average pooling

blocks) and concatenated back to the image at the corresponding up-sampling layer. Similarly, this also helps the U-net retain the original feature of the input image when up-sampling into the generated image.

Another advanced form of a U-net model is the Attention Residual U-net (ARU-net), which incorporates residual blocks as well as attention blocks into its architecture. The attention block uses several residual blocks to extract features from the skip connections before the skip connection images are concatenated onto the up-sampled images. The comprehensive architecture of the ARU-net and the convolutional layers inside the attention blocks are depicted in Figure 10.

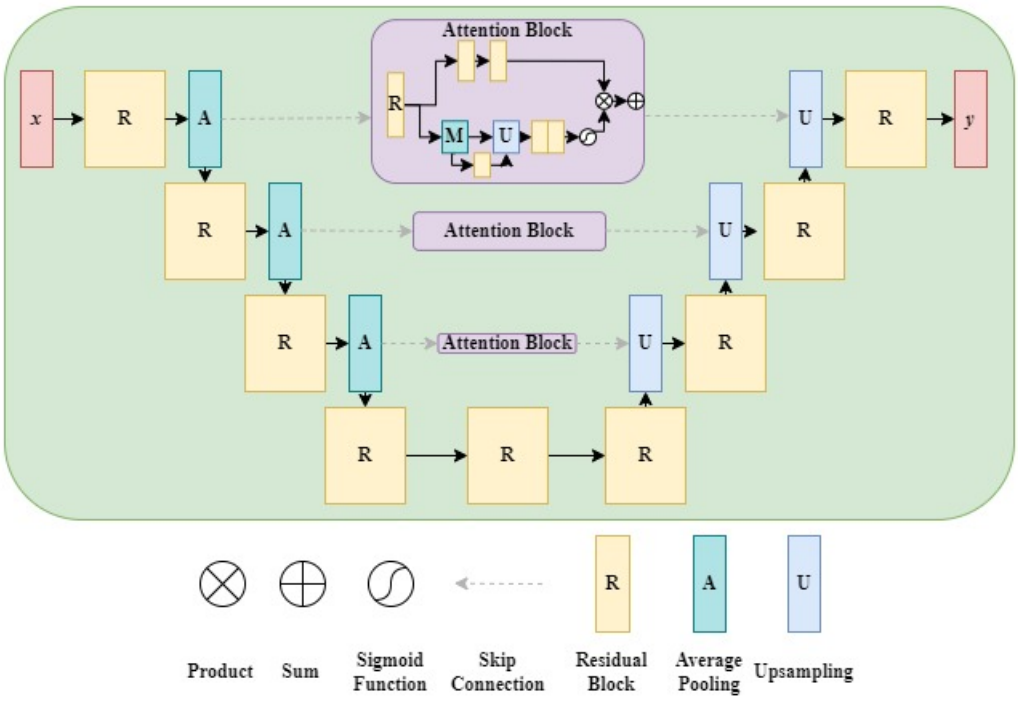



Figure 10: Architecture diagram of the Attention Residual U-net. The model down-samples and up-samples through image sizes: (256x256), (128x128), (96x96), (64x64), (32x32) and (16x16). Diagram designed using diagrams.net.

The attention block runs the image through two different sets of feature extraction procedures before multiplying the two resulting images together at the end. One set of feature extraction contains only residual blocks and the other set runs the input image through a single down-sample and up-sample layer with residual blocks. This adds a layer of additional feature extraction for the U-net to learn for converting N-body maps to hydrodynamical maps.

All three types of U-net (Vanilla U-net, RU-net, ARU-net) have been tested on their performance to convert N-body maps into $M_{tot}$ and $M_Z$ hydrodynamical maps. This is because these two fields provide the two extremes for the model to learn to impaint hydrodynamical features onto the image, with the $M_{tot}$ field being the most similar to the input N-body maps and the $M_Z$ being the least similar.

For this dataset, metallicity at any given point is defined as the ratio of the mass density of metals over the mass density of gas where any element heavier than hydrogen and helium are

considered metals. These metals are formed through stellar fusion and supernovae, therefore the bright regions in the $M_Z$ maps are areas of high stellar and supernova density. Furthermore, black holes in these hydrodynamical simulations drive metal-enriched gas from supernovae outward from the black hole into the surrounding halos [20]. For the model to successfully learn to convert the N-body map into the $M_Z$ field, it must learn to impaint supernova and black hole features onto the image. If the U-nets can successfully learn these features then their predictions can be considered to be physically realistic. Here, physically realistic means the simulations in question represent the Universe in a way that current physical models of the Universe would agree.

Each U-net has been trained for 2 epochs on NVIDIA T4 Tensor Core GPU using Google Colab. The U-nets were trained on the same 15,000 N-body maps and each model predicts 1000 images given a set of unseen 1000 N-body maps. Just as for the parameter inference task, adaptive learning rate scheduling and early stopping were used to avoid overfitting and to ensure a more controlled navigation of the network weights in loss function space. The number of epochs of 2 was chosen as the each model quickly reached a minimum in loss function space and any further training would result in the model over-fitting to the training dataset. Also, the N-body training dataset was kept at its original resolution (256x256) as the models were quick to train, so no down-sampling was required during preprocessing. The predictions obtained from these models were used as benchmarks for the diffusion models outlined later in this report.

When constructing the U-net architecture, the best down-sampling image steps were found to be (256x256), (128x128), (96x96), (64x64), (32x32) and (16x16) for the ARU-net and (256x256), (128x128), (96x96), (64x64) and (32x32) for the RU-net. This down-sampling steps provide a good compromise between retaining most of the features of the original image and allowing the U-net model to add in new detail into the image.

### 5.0.2 Novel N-body Diffusion Method

The architecture of the diffusion model is given in Figure 11.

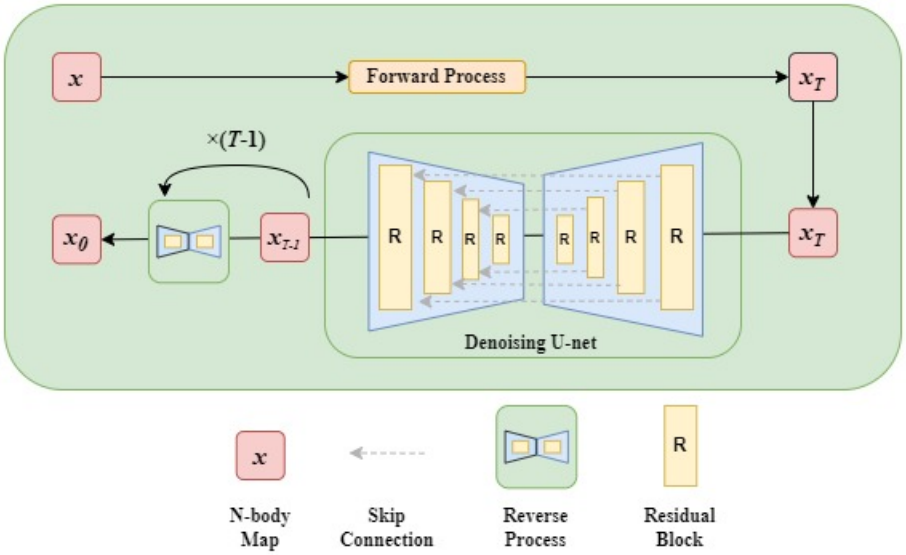



Figure 11: Architecture of diffusion model for generating novel N-body maps using a residual U-net.

The diffusion model implements a U-net to generate new images. In training, the truth N-body map is driven through the forward process which noises the image with Gaussian noise for a given variance. This variance is generated using a random diffusion time. Rather than fully noising the N-body map in the forward process, it is better to only add a random level of noise to the image as this helps with the performance of the model. The variance is necessary because denoising a signal requires different operations at different levels of noise. The noise variance is transformed using sinusoidal embeddings which helps the model to be more sensitive to the noise level. These noisy maps are stored separately to the original N-body map. Then, both the noisy map and the variances of their noise components are given as input into the U-net. The U-net transforms these noise variances into sinusoidal embeddings which are concatenated onto the image before it begins down-sampling.

As explained previously, the U-net can learn to create the truth image through down-sampling and up-sampling the image using the residual blocks. The U-net begins to predict the added noise Gaussian noise to the image and this prediction is used to measure the loss function. The MSE loss function is used to measure the difference between the predicted noise from the U-net and the actual noise from the U-net. The weights of the U-net are then updated using the optimiser AdamW to minimise the loss function and therefore guide the U-net towards successfully denoising the image. The model repeats this training step for all the N-body images in the training dataset.

For sampling, the diffusion model generates an image of random Gaussian noise and uses the trained U-net to denoise it. The model runs the random noise image several times to fully denoise the image. The number of times the random noise image is denoised by the U-net during sampling is given by the Diffusion Timesteps hyperparameter (a value of 20 was used, which was fine-tuned manually). This value was found to be a good compromise between performance and time consumption. A greater number of time steps will result in a higher resolution image but the model will take longer than desired to generate a sample. Since a random Gaussian noise will generate a different denoised image, this method generates novel

images with the same characteristics as the original input N-body maps.

The diffusion model was trained for 20 epochs on 15,000 N-body maps from the IllustrisTNG suite on an NVIDIA A100 Tensor Core GPU using Google Colab. The model was then used to generate 1000 novel images. The spatial frequency distribution between these generated images and their corresponding N-body truths could then be identified.

### 5.0.3 N-body to Hydro Diffusion Method

The previous method discussed how to generate novel images of N-body maps, but it is far more useful for a model to convert the N-body maps into their hydrodynamical counterparts. In training, the diffusion model begins in a similar way to the novel N-body diffusion model, but the hydrodynamical map is driven through the forward process instead. This generates a noisy hydrodynamical map and is stored separately from its corresponding N-body map. For this image-to-image diffusion, a different U-net must be used with 3 inputs; the noisy hydrodynamical map, the original N-body image, and the noise variance added to the hydrodynamical map. The U-net concatenates the noise variance via the sinusoidal embedding and the original N-body map before it begins down-sampling and up-sampling. This conditions the diffusion model with a given N-body map to predict the variance of the Gaussian noise added to a hydrodynamical map. The architecture of the conditional diffusion is given in Figure 12.

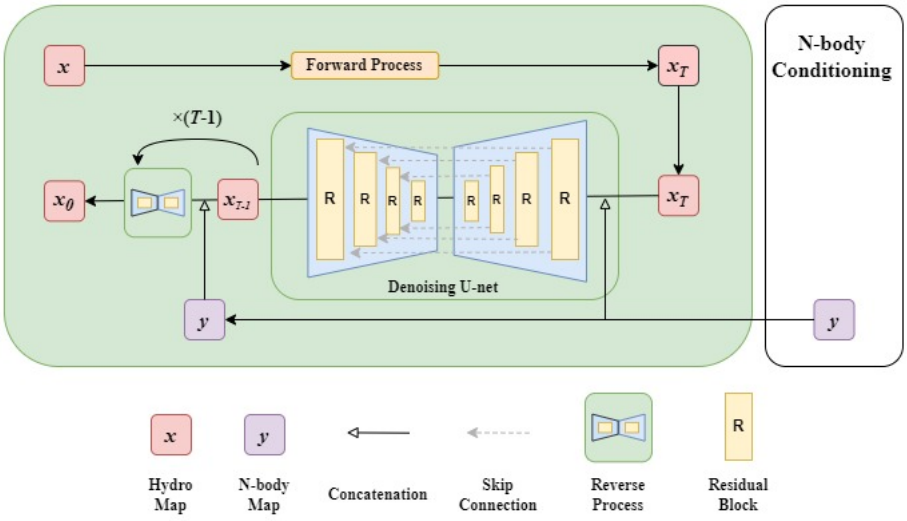



Figure 12: Architecture of diffusion model for N-body to hydrodynamical map generation using a RU-net.

The rest of the model continues in the same way as above, where the U-net prediction is measured using the MSE loss function, and the weights of the U-net are updated to successively denoise the image.

For sampling, the diffusion model takes an input of an N-body and generates a random image of Gaussian noise. These two images are concatenated in the U-net and it will predict the corresponding hydrodynamical map given the N-body map. As previously, the output of the diffusion model will be different depending on the initial random Gaussian image. If the U-net has successfully learned to denoise the image, then it will be able to 'impaint' the hydrodynamical features onto the N-body image.

Two diffusion models were created for this experiment, an diffusion model using the RU-net (Diffusion RU-net) and another diffusion model using the ARU-net (Diffusion ARU-net). This was to measure the performance of each U-net as a denoising U-net. The vanilla U-net was omitted for the task of denoising U-net as the diffusion requires a high feature extraction U-net for the denoising step to be successful.

This diffusion model was trained for 20 epochs on 15000 N-body maps and 15000 hydrodynamical maps from the IllustrisTNG suite on an NVIDIA A100 Tensor Core GPU using Google Colab. The model was then used to generate hydrodynamical 1000 images from a set of 1000 N-body maps from the training dataset. These generated images were then used to find the MSE between the generated images and 1000 hydrodynamical images. In particular, the generated images for the $M_{tot}$ and $M_Z$ fields were used to find the spatial frequency distribution between the generated images and hydrodynamical images. As before, these two fields were chosen since they represent the two extremes of the model learning to impaint the N-body with hydrodynamical features.

Furthermore, the diffusion model was trained separately for every hydrodynamical field for 20 epochs to measure the performance of the diffusion model learn to impaint N-body with different sets of hydrodynamical fields. The model was used to generate 1000 images for each field and the MSE values were calculated using the generated images.

There are 13 total hydrodynamical fields but some fields look very similar so there are 8 distinct fields that are important to measure the diffusion model's performance. These Fields are: Total Matter Density ($M_{tot}$), Gas Metallicity ($M_Z$), Magnesium over Iron ratio ($MgFe$), Gas Velocity ($V_{gas}$), Magnetic Field Strength ($B$), Stellar Mass Density ($M_{star}$) The $M_{tot}$ field is very similar to the N-body maps whereas the $MgFe$ field is very different because the $MgFe$ field represent how the supernovae eject material its surroundings. Iron is synthesized in the process of late-stage evolution of stars when the abundance of hydrogen and helium have been exhausted in the cores of stars. On the other hand, magnesium is primarily produced in supernova stage of star evolution [21]. Therefore the magnesium over iron ratio represent the relative ratio between stars and supernovae in a region in the $MgFe$ maps. Similarly, the $V_{gas}$ represent the ejection speed of gas from supernova and black hole feedback. The $B$ field shows how the intracluster medium (ICM) leads to strong magnetic fields in galaxy clusters [22]. The $ne$ field is also very different to the N-body maps and different from the other hydrodynamical maps where the image is make up of mostly dark regions and few very bright dots. These fields will present the diffusion model different examples of how supernova and black hole feedback can affect the hydrodynamical maps and its performance will be inspected.

For sampling, the same diffusion model can generate different hydrodynamical images based on the initial Gaussian image therefore only the first generated image of each diffusion model output is used to avoid bias of choosing the 'best' output image for measurement.

As a benchmark, one of the U-nets (RU-net) was used to obtain corresponding predictions to measure the performance of the diffusion to a non-diffusion N-body to hydrodynamical model.

### 5.0.4 Parameter to Map Diffusion Method

As an extension to this project, a separate diffusion model was built to generate N-body images given a set of 6 cosmological and astrophysical parameters. In training, this diffusion model begins in a similar way to the novel N-body generation model, however now the input map is a $M_{tot}$ map. The $M_{tot}$ dataset is being used here because these hydrodynamical maps make use of all 6 cosmological and astrophysical parameters whereas the N-body dataset only make

use the cosmological parameters. The $M_{tot}$ maps are passed through the forward diffusion process as the N-body maps are in the novel N-body diffusion model. To condition the image, this diffusion model has a similar method to the N-body to hydrodynamical method where the condition is fed into the RU-net but the condition becomes the 6 parameters opposed to the hydrodynamical maps.The parameters are concatenated onto the noisy $M_{tot}$ image. This is done by passing the parameters into the time embeddings layer to convert the single numerical values into (256 x 256) images, with each parameter being converted into a separate image. Then, every parameter embedding can be successfully concatenated onto the noisy $M_{tot}$ image. The noisy image, therefore, becomes a 7-channel image where the channels are: Noisy $M_{tot}$ time embeddings, $\Omega_m$ embeddings, $\sigma_8$ embeddings, $A_{SN1}$ embeddings, $A_{SN2}$ embeddings, $A_{AGN1}$ embeddings, and $A_{AGN2}$ embeddings. The rest of the model follows the same architecture as the 'N-body to hydrodynamical map' model. The architecture of this diffusion model can be seen in Figure 13.

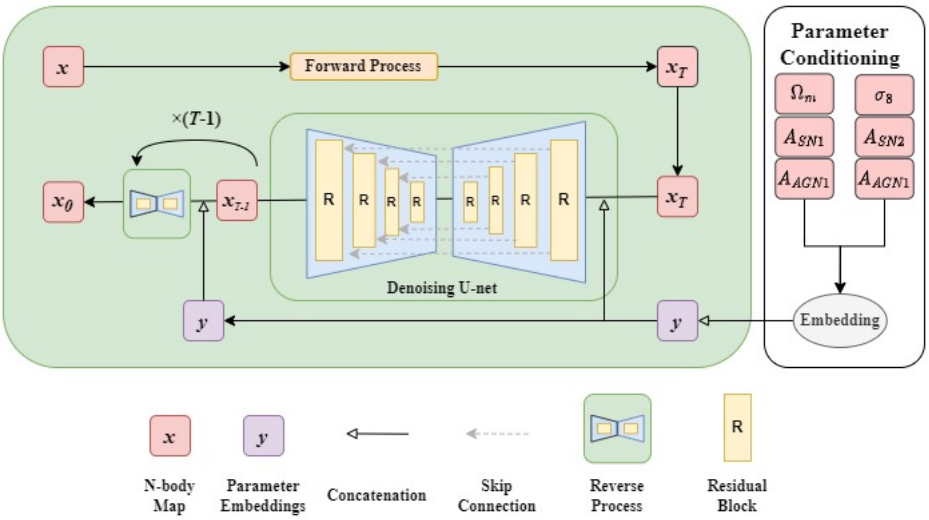



Figure 13: Architecture diagram of 'Parameter to Map' diffusion model, which is able to take a set of 6 cosmological and astrophysical parameters and output a novel N-body map.

For sampling, the diffusion model takes the 6 parameters as input and generates a novel $M_{tot}$ map. As before, the output of the diffusion model will be different depending on the initial random Gaussian image. If the U-net has successfully learnt to denoise the image into an $M_{tot}$ map, it will be able to generate a new $M_{tot}$ map just with the parameter inputs in a similar way that supercomputers generate the $M_{tot}$ maps in this dataset but without needing to simulate the complex gravity and hydrodynamical interactions between galaxy clusters.

The model was trained for 20 epochs on 15,000 $M_{tot}$ maps from the IllustrisTNG suite on an NVIDIA A100 Tensor Core GPU using Google Colab. The set of Cosmic Variance (CV) $M_{tot}$ maps from the IllustrisTNG suite were used as the testing dataset. These 405 CV maps all have the same associated parameters: $\Omega_m = 0.309$, $\sigma_8 = 0.979$, $A_{SN1} = 3.112$, $A_{SN2} = 1.122$, $A_{AGN1} = 0.669$, and $A_{AGN2} = 0.532$. These parameters are fed into the diffusion model after training and are then used to generate 405 $M_{tot}$ images. The spatial frequency distribution between these generated images and their corresponding $M_{tot}$ truths could then be calculated to determine whether the generated images are physically realistic.

## 5.1 Results

### 5.1.1 N-body to Hydro U-net

Each of the U-net predictions for the hydrodynamical maps are given in Figure 14.

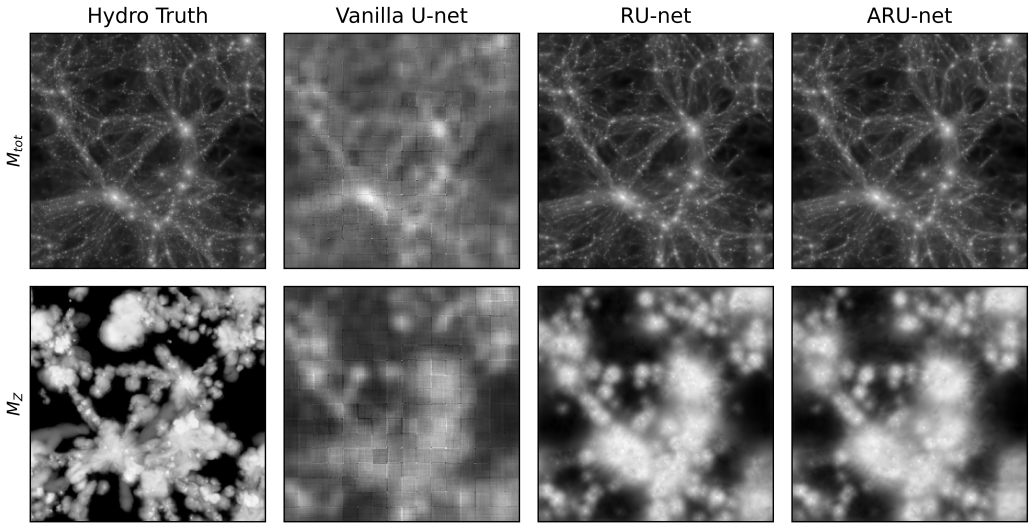



Figure 14: Total Matter Density ($M_{tot}$) and Gas Metallicity ($M_Z$) ground truth maps against each type of U-net and their corresponding predictions.

The spatial frequency distribution of each U-net predictions and the corresponding $M_Z$ hydrodynamical map is given in Figure 16.

| **Hydrodynamical Field** | **Vanilla U-net MSE** | **Residual U-net MSE** | **Attention Residual U-net MSE** |
|---|---|---|---|
| $\boldsymbol{M_{tot}}$ | 0.0152 | 0.0016 | 0.0017 |
| $\boldsymbol{M_Z}$ | 0.2699 | 0.2687 | 0.2561 |

Figure 15: Table showing the MSE values for 1000 generated from each U-net against the corresponding hydrodynamical truth maps.

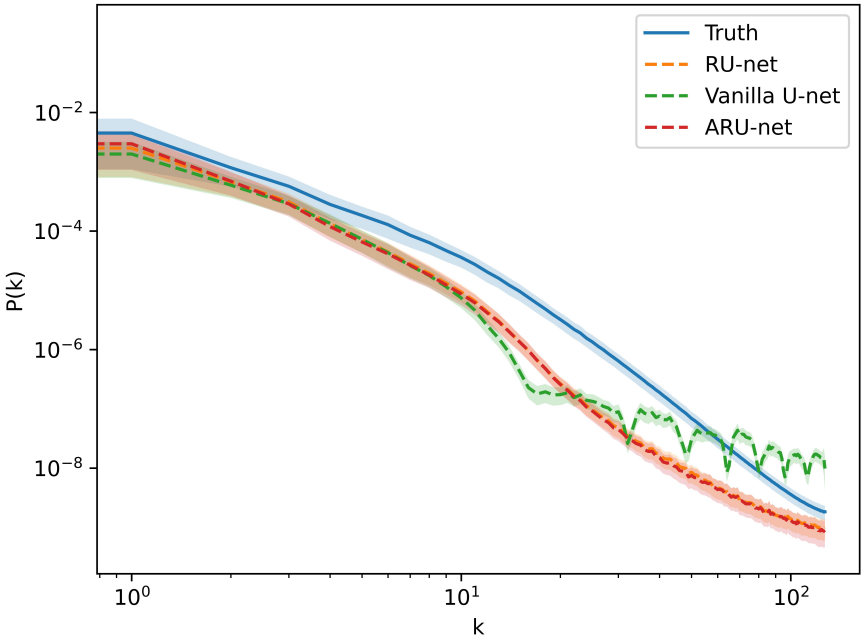



Figure 16: Frequency power spectrum of 1000 $M_Z$ truth images and 1000 of each U-net prediction for the $M_Z$ given the corresponding N-body map. The shaded region of each distribution represent $\pm 1\sigma$ of the frequency distribution.

While constructing each U-net architecture, the down-sampling and up-sampling lengths were varied to find the best depth of the network. The best depths of each U-net are the depths (lowest image size and number of down-sampling steps) used in each architecture.

Using these results of performance for the U-nets, the diffusion models will only incorporate the RU-net and ARU-net as part its architecture.

### 5.1.2 Novel N-body Diffusion generation

An example of how the diffusion model denoises an image from random Gaussian noise over various time-steps is shown in Figure 17.

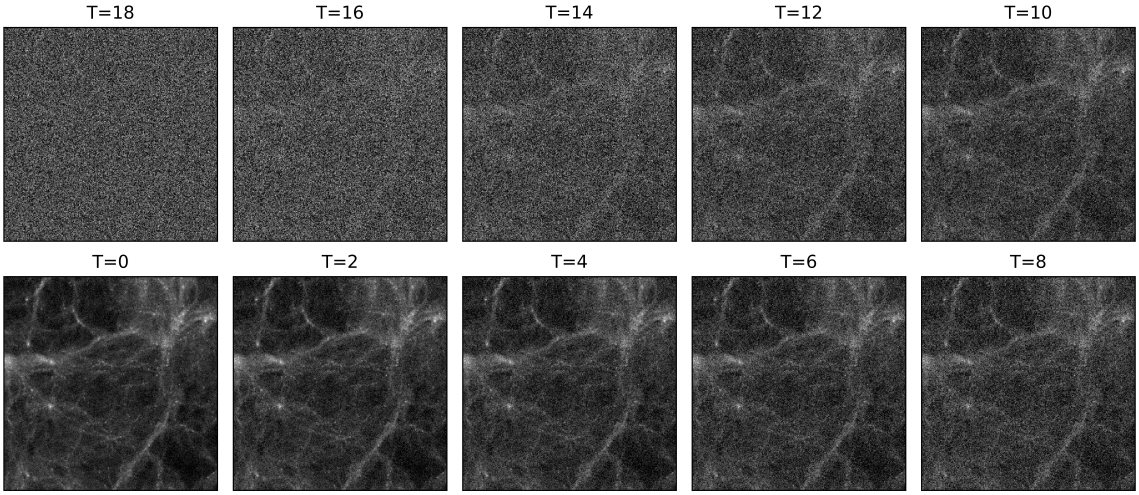



Figure 17: Example of reverse diffusion of noisy image to novel N-body image over 20 timesteps. Diffusion model trained for 10 epochs taking 20 minutes.

Using the diffusion model, the generated N-body images were produced and a sample of the images are shown in Figure 18.

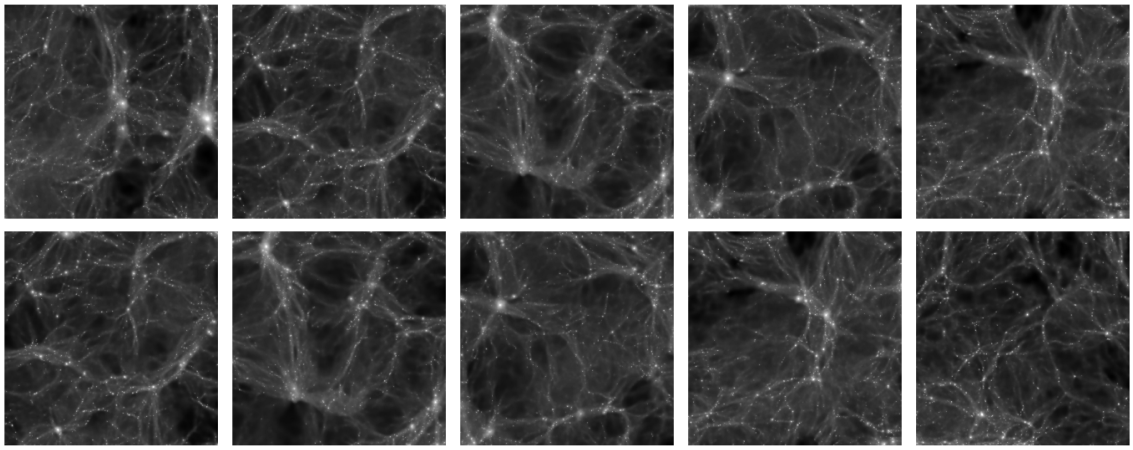


Figure 18: Sample of 10 images from Diffusion model outputs for novel N-body generations. The model was trained for 20 epochs taking 40 minutes.

The spatial frequency distribution of all of the generated images against a sample of 1000 N-body images is shown in Figure 19.

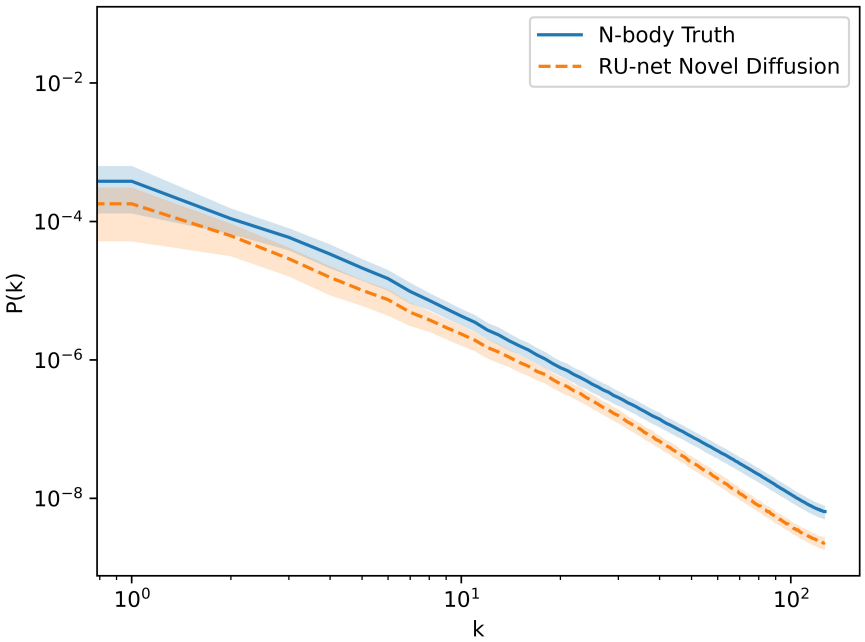



Figure 19: Frequency power spectrum of 1000 N-body images and 1000 diffusion N-body generated images. The shaded region of each distribution represents $\pm 1\sigma$ of the frequency distribution.

### 5.1.3 N-body to Hydro Diffusion

A comparison of each diffusion model with various U-nets performance alongside the performance of the most accurate U-net only model for two hydrodynamical fields from a single N-body image are given in Figure 20.

Figure 21 displays the MSE values for the different diffusion models with various U-net architectures and the MSE values for the most accurate U-net model. This was calculated by taking into account all generated images.

Taking all the generated images from the diffusion models, the spatial frequency distribution for the diffusion models with various U-net architectures alongside the spatial frequency distribution for most accurate U-net were calculated for a single hydrodynamical field and shown in Figure 22.

The diffusion model predictions for the 8 hydrodynamical fields which are various difficulties for the diffusion model to impaint is given in Figure 23.

The MSE values for the diffusion model predictions against every field in the CAMELS is shown in Figure 24.

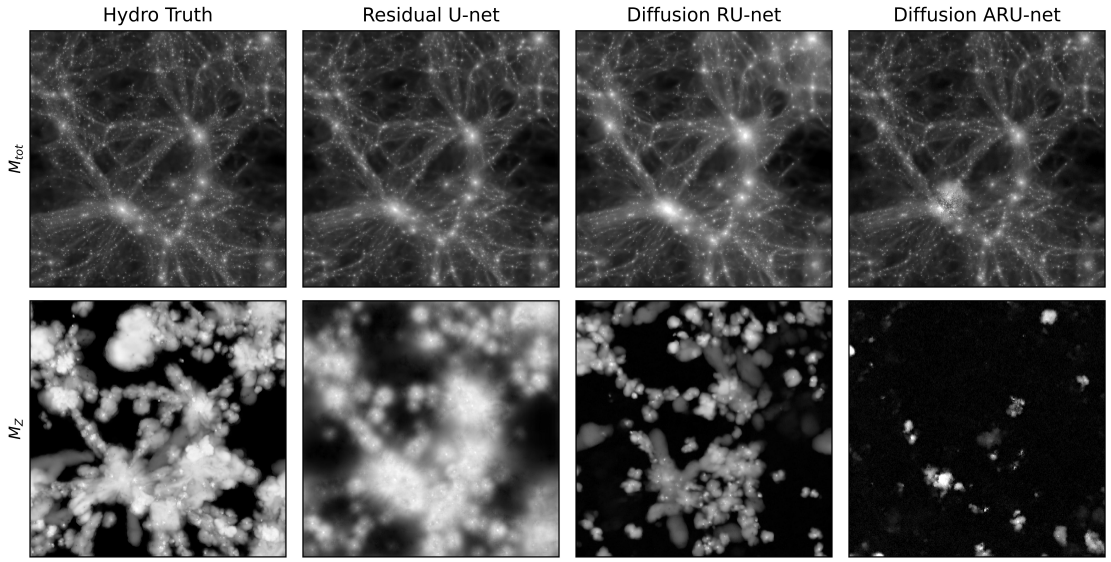



Figure 20: Diffusion with RU-net (Diffusion RU-net) vs Diffusion with ARU-net (Diffusion ARU-net) vs RU-net single field prediction for $M_{tot}$ and $M_Z$.

| **Hydrodynamical Field** | **Residual U-net MSE** | **Diffusion with RU-net MSE** | **Diffusion with ARU-net MSE** |
|---|---|---|---|
| $\boldsymbol{M_{tot}}$ | 0.0016 | 0.0121 | 0.0059 |
| $\boldsymbol{M_Z}$ | 0.2687 | 0.5671 | 0.7627 |

Figure 21: Table showing the MSE values for 1000 generated from each model against the corresponding hydrodynamical truth maps.

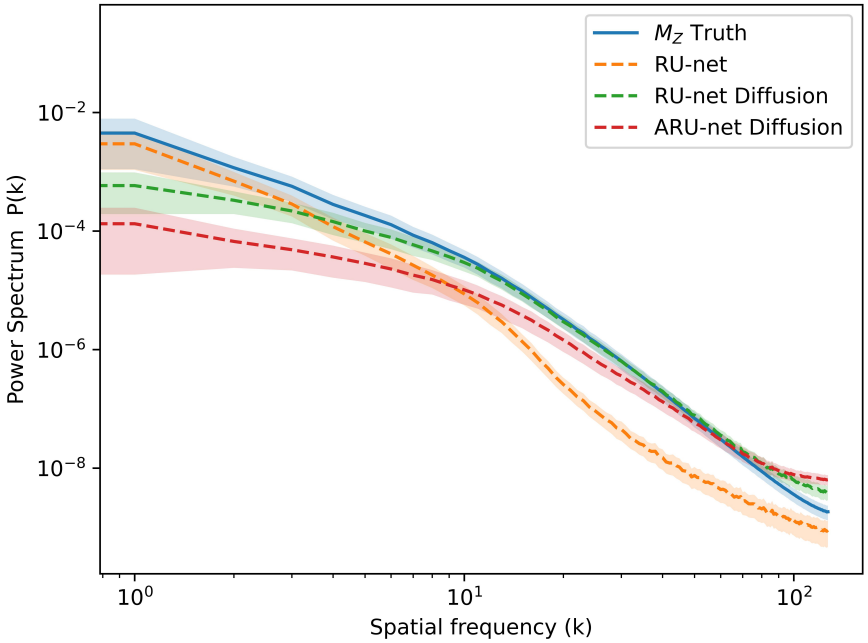



Figure 22: Frequency power spectrum of 1000 $M_Z$ truth images and 1000 of each diffusion predictions and RU-net predictions for the $M_Z$ given the corresponding N-body map. The shaded region of each distribution represents $\pm 1\sigma$ of the frequency distribution.

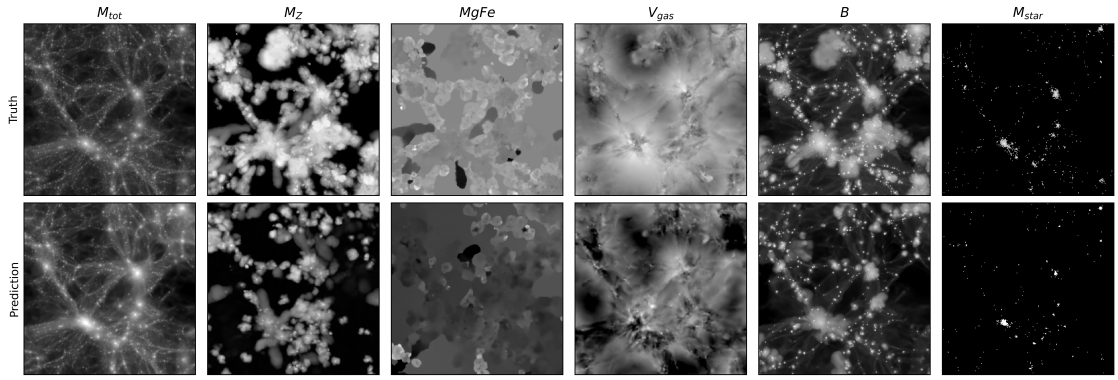



Figure 23: Sample of Diffusion RU-net model predictions for 8 fields with the same N-body map input against hydrodynamical truth maps. Fields left to right are: Total Matter Density ($M_{tot}$), Gas Metallicity ($M_Z$), Magnesium over Iron ratio ($MgFe$), Gas Velocity ($V_{gas}$), Magnetic Field Strength ($B$), Stellar Mass Density ($M_{star}$). Each prediction was sampled using a diffusion model trained for each field separately for 20 epochs.

| Hydrodynamical Field | Diffusion Model MSE |
|---|---|
| Gas Density ($M_{gas}$) | 0.0173 |
| Gas Velocity ($V_{gas}$) | 0.5462 |
| Gas Temperature ($T$) | 0.1422 |
| Gas Pressure ($P$) | 0.0409 |
| Gas Metallicity ($Z$) | 0.5671 |
| Neutral Hydrogen Mass ($HI$) | 0.0196 |
| Electron Number ($ne$) | 0.0266 |
| Magnetic Field ($B$) | 0.1467 |
| Magnesium over Iron Ratio ($MgFe$) | 0.3062 |
| Dark Matter Density ($M_{cdm}$) | 0.0041 |
| Dark Matter Velocity ($V_{cdm}$) | 0.5427 |
| Stellar Mass ($M_{star}$) | 0.0256 |
| Total Matter Density ($M_{tot}$) | 0.0121 |

Figure 24: Table showing the MSE values for 1000 generated images from an N-body to hydro diffusion model trained on each field separately against the corresponding hydrodynamical truth maps.

### 5.1.4 Parameter to Map Diffusion

An sample of the parameter to map diffusion model is shown in Figure 25.

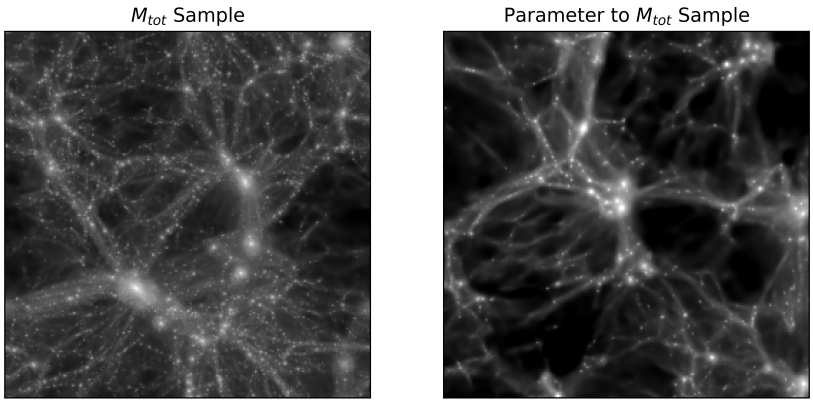



Figure 25: A sample image of the $M_{tot}$ CV dataset compared to the 'parameter to map' diffusion model given the same parameters as the $M_{tot}$ CV image.

The frequency power spectra of generated images is given in Figure 26.

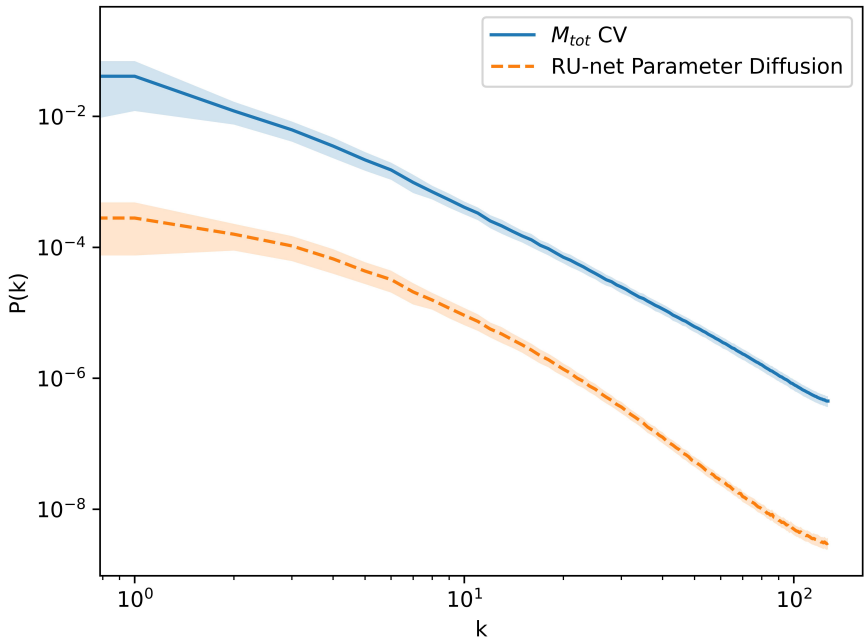



Figure 26: The frequency power spectra of 405 $M_{tot}$ CV dataset against the 405 generated $M_{tot}$ of the parameter to map diffusion model given the same parameters input as the $M_{tot}$ CV dataset.

## 5.2 Discussion

### 5.2.1 N-body to Hydro U-net

Looking at Figure 14, the RU-net and ARU-net successfully recreated the $M_{tot}$ image from the N-body with high resolution whereas the Vanilla U-net recreated the $M_{tot}$ with low resolution. The Vanilla U-net, RU-net and the ARU-net recreated the high-density regions of the $M_Z$ maps despite the resolution of the Vanilla U-net predictions being lower than the predictions of the RU-net and ARU-net.

The lack of resolution of the Vanilla U-net can be explained by the lack of complexity of its architecture. The Vanilla U-net only consists of single convolution layers before down-sampling which is less complex at extracting features of the image than the residual and attention blocks of the RU-net and ARU-net.

Despite the additional complexity of the RU-net and ARU-net, the predictions from these models are not at the high resolution of the hydrodynamical truth for the $M_Z$ field. This is because the $M_Z$ map looks very different from the starting N-body map that these models were given, consequently the RU-net and ARU-net varied the weights of their parameters to keep the sections of the N-body map that are similar but failed to recreate the finer supernova features of the $M_Z$. Whereas with the $M_{tot}$, the RU-net and ARU-net were able to vary to weights of their neurons to keep most of the original N-body and slightly make the image more diffuse to represent the density of gas correctly.

Looking at Figure 15, the RU-net and ARU-net performed much better than the Vanilla U-net for the $M_{tot}$ field for the MSE values with the RU-net and ARU-net performing almost

the same. For the $M_Z$ field, the MSE values for all the U-nets are very similar with the ARU-net performing slightly better. This agrees with the expectation that ARU-net would perform better than the other U-nets since it is the most complex concerning convolution layer structure. However, this difference in performance cannot be seen by inspection of the image.

To analyse the similarity of the predictions to the hydrodynamical truth maps, it is more beneficial to look at the spatial frequency distributions of the two sets of images. Looking at Figure 16, the RU-net and ARU-net predictions have an almost identical power spectrum to each other whereas the Vanilla U-net has a similar power spectrum at low frequencies but deviates heavily at high frequencies from the $M_Z$ truth maps and the RU-net and ARU-net predictions. This agrees with observations of looking at the model predictions in 14 where the Vanilla U-net predictions for the $M_Z$ field are much darker than the truth image, therefore, the Vanilla U-net predictions have fewer high frequencies areas. Furthermore, the Vanilla U-net power spectrum at high frequencies is not smooth or consistent with the $M_Z$ truth maps. This shows that the Vanilla U-net failed to recreate the $M_Z$ field at high densities of metallicity and it cannot create a physically realistic $M_Z$ map. This means that the Vanilla U-net cannot learn the supernova and black hole feedback features of the images and impaint them on the N-body map. This is due to the lack of complexity of the Vanilla U-net architecture. However, the power spectrum of the RU-net and ARU-net show that the predictions have an almost identical power spectrum at low frequencies to the $M_Z$ field map but deviate at high frequencies. This agrees with the observations of looking at the model predictions in **??** where the RU-net and ARU-net learnt the large high metallicity areas of the $M_Z$ map but failed to re-create a few high and low metallicity areas of the image at high resolution. This means that these U-nets were able to learn the large-scale supernova and black hole feedback features but not the small-scale features of the image.

### 5.2.2 Novel N-body Diffusion generation

Looking at Figure 17, the diffusion model starts with a noisy image and slowly denoises it into a novel N-body image. This has shown that the diffusion model can learn to generate N-body given a large dataset of 15000 N-body images from IllustrisTNG. The choice of IllustrisTNG over SIMBA was arbitrary since both datasets contain 15000 N-body images which are very similar to each other. The diffusion timesteps of 20 was chosen to be a compromise of performance and time consumption and has proven to be successful here as the U-net stops denoising as the Gaussian noise disappears. Despite only training for 10 epochs, equivalent to 20 minutes of training, the model has successfully learnt to recreate an N-body map at high resolution. Training the model for longer than 20 epochs did not increase resolution as the training loss plateaued after 10 epochs and would guide the model to over-fitting.

Looking at Figure 18, the diffusion model generates 10 different N-body images despite being sampled using the same reverse process. For every time the reverse diffusion step occurs, the diffusion model generates a new Gaussian noise image for the residual U-net to denoise. The U-net is sensitive to the random noise image meaning that a different image of random Gaussian noise will lead to the U-net denoising the image into a vastly different-looking map. Since the starting random noise image is (256x256), the probability of generating the same Gaussian noise image is very unlikely. Therefore, each time the reverse diffusion step occurs, a new N-body map will be generated. (N-body training data also look very different from each other)

A concern about the images generated by the diffusion model is that the images may not be physically realistic. The meaning of being physically realistic here means the simulations

represent the universe in a way current physical models of the universe would agree with in simulations. If the generated images are not physically realistic, then the new images would have no significance in providing additional N-body simulations. However looking at Figure 19, the N-body images generated by the diffusion model have a very similar frequency power spectrum to original N-body images. This means that the diffusion model learned to recreate physically realistic N-body maps since the frequency distributions of both sets of images follow the same trend. At low frequencies, the errors of the original N-body and generated N-body images overlap and then begin to deviate at high frequencies. This means that the generated images were performing worse at re-creating the high-density sections of the N-body images but performed better at re-creating the low-density sections. Ideally, the generated frequency spectrum should lie on the original N-body frequency spectrum. However this is not the case, and it may be due to the nature of the generated images of the diffusion models being very different based on the initial random Gaussian noise.

A useful application of this novel N-body generation diffusion model is to use the generated N-body images with the N-body to hydrodynamical diffusion model to convert these N-body images into hydrodynamical images. This would effectively remove the need to generate new simulations of N-body and hydrodynamical maps using supercomputers. However, these new simulations would not contain the 6 parameters which describe each simulation so these generated simulations would not be physically useful.

The diffusion model is very slow to generate images, taking $\sim$ 20 minutes to generate 1000 images. Extrapolating this would mean that the diffusion model would take approximately $\sim$5 hours to generate a full suite of 15000 N-body maps. While faster and more practical than generating the simulations on a supercomputer, the diffusion model would likely not be as physically representative of the Universe.

While the diffusion model for generating novel N-body maps used an RU-net as a denoising U-net, the ARU-net performed similarly with no significant change apart from increasing the time taken to sample 1000 images.

The model was only trained for a maximum of 20 epochs, whereas it is common to train a diffusion model for much longer (100 epochs). Although the model was tested and trained for longer, the loss function plateaued and there were no visible changes to image resolution.

Looking at Figure 17, the hyperparameter of Diffusion Timesteps (Diffusion Timesteps $=$ 20) has proven to be a good compromise on denoising the image accurately while keeping the sampling time low.

It is also common to use random augmentations on the training dataset to increase the size of the dataset without needing any new images. However, due to the already high computational complexity of the diffusion model, this was omitted to keep the training time of the model short. Furthermore, using the single un-augmented dataset keeps the outputs of the diffusion model reproducible as using random augmentations would make the diffusion vary even more throughout different training sessions.

Similarly, [7] used a DDPM to generate novel dark matter density field ($M_{cdm}$) maps for this dataset. The DDPM also successfully generated physically realistic images. It has been shown here that the implicit diffusion model can generate N-body maps that are physically realistic.

### 5.2.3 N-body to Hydro Diffusion

Looking at Figure 20, both diffusion models successfully denoised an image of random Gaussian noise into its corresponding $M_{tot}$ map. This matched the performance of the non-diffusion

RU-net to generate $M_{tot}$ maps. As mentioned, this is because the input N-body map and the corresponding $M_{tot}$ map look very similar with the latter being more diffuse. For the $M_Z$ field, however, the diffusion model predictions are very different. The Diffusion RU-net model re-created most of the high metallicity regions of the $M_Z$ image and the Diffusion ARU-net failed to re-create almost all of the high metallicity regions. The Diffusion ARU-net only learnt to re-create the highest metallicity region at the centre of the image as a bright dot alongside other bright dots around the centre. These bright dots correspond with the highest density regions of the $M_{tot}$ image. Therefore, the Diffusion ARU-net model only learnt to extract the highest density regions of the input N-body image and failed to impaint the image with any supernova or black hole feedback features. The Diffusion RU-net performed better as it learnt to extract most of the large-scale features of the input N-body image and successfully in-painted the image with hydrodynamical features at high resolution. Comparing this to the non-diffusion RU-net $M_Z$ image, RU-net performed at extracting all the large-scale features but failed to impaint the hydrodynamical features at high-resolutions. This is because the non-diffusion RU-net model starts with the N-body and adjusts the brightness of the input N-body image through convolutions to match the $M_Z$ field. This means that the RU-net cannot fully impaint the small-scale hydrodynamical features of the $M_Z$ as this information is not contained in the N-body image. Whereas the diffusion model starts with the input N-body and an image of random Gaussian noise and denoises the noisy image to generate a new image. This means that the diffusion model excels at generating new images with the same characteristics of the training dataset as seen here. The diffusion model can learn to impaint the hydrodynamical features of the $M_Z$ field much better than the non-diffusion RU-net.

The diffusion RU-net model learning to impaint the N-body image with hydrodynamical features for the $M_Z$ field is very impressive, as it shows the model is capable of learning the physical nature of supernova and black hole feedback. The diffusion model has learnt to recognise regions of high dark matter density in the N-body as regions of high stellar mass where fusion takes place to produce metals. The $M_Z$ field map in the 20 has large clouds of metal enriched gas from supernovae that produce large amounts of heavy elements. These heavy elements have been spread out through supernova ejection and black hole accretion. To an extent, the diffusion model has learnt to recognise the high dark matter density regions of the N-body image as regions where supernovae occur and metals are ejected from their host galaxy.

Figure 21 shows that both the Diffusion RU-net and Diffusion ARU-net models performed worse for both the $M_{tot}$ and $M_Z$ fields than the non-diffusion RU-net. This was not the expected result as the diffusion predictions for the $M_{tot}$ field look remarkably similar to the truth image in Figure 20. However, on closer inspection, the Diffusion RU-net prediction appears slightly too bright in the high-density regions for the $M_{tot}$ truth, and the Diffusion ARU-net appears too sharp and too dark in the low-density regions. This explains the discrepancy between the MSE values for the diffusion models' predictions being higher than the non-diffusion RU-net predictions. One reason for this is that the diffusion models must denoise the initial random Gaussian noise into the $M_{tot}$ which may leave residual features in the output image that are not in the N-body image whereas the non-diffusion RU-net only starts with the N-body image which only requires adjusting the separate pixel brightness of starting image into the $M_{tot}$. For the $M_Z$, both diffusion models performed worse than the non-diffusion RU-net with the Diffusion ARU-net performing significantly worse. This agrees with the observations of 20 for the Diffusion ARU-net prediction where the model failed to capture any large or small scale features of the $M_Z$ map. The Diffusion RU-net model however did successfully re-create

the small-scale features but not all of the large-scale features. This is represented in the MSE value for the Diffusion RU-net being greater than the MSE value for the non-diffusion RU-net where the RU-net re-created the large-scale features but not the small-scale features of the $M_Z$ field.

Looking at Figure 22, the frequency power spectrum of the Diffusion RU-net is much closer to the power spectrum of the $M_Z$ truth maps than the non-diffusion RU-net predictions. This means that the Diffusion RU-net predictions are more physically realistic than the non-diffusion RU-net predictions. This agrees with the observations seen in Figure 20 but not the MSE results in 21. The power spectrum of the Diffusion RU-net at low frequencies is further from the $M_Z$ truth maps than the power spectrum of the non-diffusion RU-net. Since most of the pixels that would make up the prediction maps are low frequency, the MSE values for each prediction are skewed towards how similar the power spectrum is to the truth maps at low frequencies and insensitive to how similar the power spectrum is to the truth maps at high frequencies. This reveals that the MSE difference between prediction and truth maps does not fully represent the image similarity. Power spectrum analysis is required for proper analysis. The power spectrum of Diffusion ARU-net shows that the predictions are similar to the $M_Z$ truth maps since the power spectra follow a similar trend. This disagrees with the observations of 20, where the model failed to re-create the large or small scale features. One reason for this is that the bright areas in the Diffusion ARU-net predictions can approximate the $M_Z$ truth maps, therefore the power spectra appear similar.

Looking at Figure 23, the diffusion model re-created each field with high resolution. The diffusion model successfully learnt the regions with constant magnesium over iron ratio as well as the regions that vary where the galaxy clusters are. Similarly, the $V_{gas}$ represents the ejection speed of gas from supernova and black hole feedback. The diffusion model learnt to represent the high-density regions of the N-body image with high gas velocities and low-density regions with low gas velocities. The diffusion model also re-created the small-scale structure of the magnetic fields but only most of the large-scale strong magnetic fields. This is consistent with the model's performance with the $M_Z$, as the $M_Z$ and $B$ look similar. Finally, the model re-created the bright and dark areas at the correct positions in the $ne$ field.

All of these model predictions for these fields show that the diffusion model can impaint several types of supernova features onto an N-body image and even more complex features such as velocities of gas and magnetic fields.

Looking at Figure 24, the lowest MSE values for the diffusion model prediction is the $M_{cdm}$ field. This is because the N-body maps only contain the densities of dark matter like the $M_{cdm}$ field, therefore, there are little or no hydrodynamical features for the diffusion model to impaint the starting N-body image. The highest MSE values for diffusion model predictions are the $M_Z$, $V_{gas}$, and $V_{cdm}$ with the $M_Z$ being slightly higher. This is reasonable as these three fields all require a large reconstruction of the N-body images. MSE results imply that the $M_Z$ field is slightly more difficult for the diffusion model to re-create than the velocity fields which imply the supernova and black hole feedback are harder for the model to learn and predict. The MSE results also show that the diffusion model is significantly more accurate at predicting the $T$ and $P$ despite these fields having similar complexity of hydrodynamical features.

Despite the diffusion model learning the supernova and black hole features accurately, most of the model predictions in Figure 23 appear darker than the truth images. One reason for this is that the Diffusion RU-net and Diffusion ARU-net are very sensitive to normalisation and any error in the normalisation of the training dataset can lead to the model over or under-predicting the brightness of pixels in the output image.

Ideally, the Diffusion RU-net would only have to be trained on with all hydrodynamical fields being in the training dataset. This could be done by assigning each field with a label and feeding the label input into the denoising RU-net to denoise the noisy image into the correct hydrodynamical field. However, this was not done as each field contains 15,000 images. Training the model on N-body to a single hydrodynamical field already requires 30,000 images for the training dataset and having N-body to any field would require 210,000 images for the training dataset. This would make training the diffusion model extremely slow and require large amounts of memory that this experiment does not have access to. Furthermore, training the diffusion model to predict every field would encounter problems with cross-training where the diffusion model predictions would be less accurate than if the model was trained separately.

Despite the ARU-net performing better than the RU-net on the $M_{tot}$ and $M_Z$ fields, the Diffusion ARU-net performed remarkably worse than the Diffusion RU-net. The Diffusion ARU-net consistently predicted images that contained only the highest density regions of the input N-body images. This disagrees with the consensus that adding attention blocks increases the performance of the diffusion model [23]. One reason for this is that the attention blocks of the ARU-net were not optimised to denoise images and specialise at extracting features from the input image but not feature retention.

The methods of analysis done on the generated images were inspection by eye, MSE calculations, and power spectrum analysis. However, these methods sometimes showed conflicting results. The power spectrum analysis is the most accurate to measure the performance of each model however it is impractical to show for several fields. Whereas MSE analysis is practical to show as it is a single numerical value however it does not provide the most accurate performance evaluation. Ideally, the best form of analysis is a single numerical value that incorporates the frequency power spectrum such as a value to measure the distance between the prediction and truth spectrum. It is common in literature to use Inception score (IS) and Fréchet Inception Distance (FID) as a form of measurement [24].

Diffusion models are hard to control and guide towards successful impainting. The conditioning method of concatenating the N-body image to a random Gaussian noise image relies on the N-body features propagating throughout RU-net. Having additional concatenations after the down-sampling layers of the RU-net did not increase performance and even prevented the diffusion model from being to denoise the noisy images. Ideally, the best method of conditioning a diffusion model is to implement cross-attention layers into the denoising U-net as seen in StableDiffusion [25]. However, these layers were too computationally expensive to implement with the Diffusion RU-net. Despite the A100 GPU being the most powerful GPU available in Google Colab, the training of the Diffusion RU-net utilised the full capability of the GPU and additional computational resources would be required for cross-attention. Furthermore, it has been shown that randomly removing the hydrodynamical truth maps and replacing them with null labels improves diffusion performance as the diffusion model can more accurately denoise the random Gaussian noise.

### 5.2.4 Parameter to Map Diffusion

Looking at Figure 25, the parameter to map output successfully re-created the large scale structure of the $M_{tot}$ map. However, the model predictions are significantly darker than the expected $M_{tot}$ image, and they have in fact failed to re-create the small scale features. One reason for this is that the model was only able to learn the general structure of the $M_{tot}$ image through the input parameters. This implies that the diffusion model was only able to learn from the cosmological parameters and did not learn the hydrodynamical features from the

astrophysical parameters.

Looking at Figure 26, the power spectra of the two sets of images follow an almost identical trend. This shows that the generated images are physically realistic however, this does not necessarily mean that these generated images are as accurate and reliable as the $M_{tot}$ images in the dataset. The model may have learnt to re-create the large scale structure of a $M_{tot}$ image but the large scale structure may not be physically feasible. Further analysis than inspection by eye and the power spectrum on the physical reality of these generated images is required.

# 6 Conclusions

## 6.1 Review of Results

To conclude, both the parameter inference and image-to-image tasks returned significant results and the project was a success overall. As outlined in 4.2, the parameter inference task was successful, particularly for the cosmological parameters. The lowest mean fractional errors for the cosmological parameters were found when the inference model was trained on the $M_{tot}$ maps from the SIMBA simulation suite. These errors were $4.0\%$ for $\Omega_m$ and $2.2\%$ for $\sigma_8$. These results outperform the paper which the task was based on [14] which is a positive outcome. Unfortunately, the model fell short when inferring the astrophysical parameters with the highest performance being $60.4\%$ for $A_{SN1}$, $96.8\%$ for $A_{AGN1}$, $27.0\%$ for $A_{SN2}$ and $34.2\%$ for $A_{AGN2}$. The main reason for this could be the that on the large cosmological scale both active galactic nuclei and supernovae are relatively small elements, making them much harder to detect and constrain via the inference method. The image-to-image generation task was also successful as a whole. With reference to figure 16, the vanilla U-net was the least successful N-body to hydrodynamical U-net model and did not agree with the ground truth when considering the higher frequency details in the generated $M_z$ images. The residual U-net (RU-net) and residual with attention mechanism U-net (ARU-net) had a much greater deal of success when compared to the ground truth images with the main deficiencies being related to the pixel value scalings. N-body to hydrodynamical diffusion proved to be the most successful adaptation to the models, as can be seen in figure 22. The RU-net and ARU-net diffusion systems replicate the ground truth $M_z$ images well. They perform slightly worse when replicating lower frequency detail but are very effective at replicating high frequency detail, which is where the non-diffusion models fall short. The diffusion process has hence been incorporated successfully to return excellent results and forms an excellent groundwork for which the model can be modified and improved to ensure the highest possible performance. A short insight was conducted into parameter to map diffusion but the results obtained in this project are promising.

Overall, this project's Diffusion RU-net has learnt successfully the hydrodynamical features that arise from each hydrodynamical field in the CAMELS dataset and has been able to impaint N-body maps to hydrodynamical maps.

## 6.2 Next Steps

It would be beneficial to test a diffusion model for a hydrodynamical map to an N-body map to see if it can equally remove supernova and black hole feedback features from the image. This could then be used to convert the generated hydrodynamical maps from previous diffusion models back into N-body and inspect if any information about the original map was lost.

While diffusion models can generate high-resolution images, they are much slower to generate these images. To generate 1000 images, the diffusion model can take up to 40 minutes for sampling. This is one of the disadvantages of using diffusion models rather than U-nets and GANs. U-nets are much less computationally expensive than diffusion models since each image is only down-sampled and up-sampled once compared to the 20 times the reverse process is done in the diffusion model. GANs avoid being as computationally expensive as diffusion models by transforming the input image into latent space. From here the generating and discriminating of images is done and then transform the image back into pixel space for sampling.

Diffusion models are a relatively new deep learning generative model and there are relatively few available resources to learn and create a custom diffusion model as was done in this project. As further research is conducted, more diffusion models will be made and present models which can aim to beat this project's Diffusion RU-net as a benchmark.

## 6.3 Wider Implications of AI Image Generation

It is without doubt that machine learning models will continue to become more complex at an exponential rate. With this, the power of machine learning and its applications in science will become ever more apparent. As shown by this project, machine learning can be used to achieve feats which would have been near impossible prior to the last few years. Many aspects of physics involve the analysis and manipulation of images. This implies that use of machine learning as a scientific method in imaging-based disciplines such as astronomy, nanoscience and biomedical imaging is possible and should be employed going into the future. However, it is important to note that these types of machine learning applications, specifically image related tasks, are in their fledgling days. As mentioned earlier in the report, the Stable Diffusion model has gained immense popularity due to its text-to-image and image (with text prompt)-to-image impainting ability, with many people eager to generate images using this AI technology. Although exciting, there are many issues which have begun to arise due to this now widely-available technology. These models are trained such that the generated images are as realistic as possible. This has led to a surge in images known as "AI Deepfakes" which are generated images (or videos) which depict a person or a group of people doing or saying things which did not actually happen [26]. This has often been seen used against politicians and celebrities and causes apprehension and mistrust in the media. However, this also causes distress for the targeted person(s). A way to curb these deepfakes is to use machine learning to build deepfake detectors [27]. It is therefore important that deepfake detection technology is employed across media services to ensure that consumers are aware of deepfakes. It is vital that work on deepfake detectors continues as the realism of AI deepfakes increases exponentially. In the UK, currently it is illegal to share deepfake images which maliciously target an individual(s), yet there are no laws regarding generating such images in the first place. It should be the job of policymakers to ensure that new laws are introduced to protect individuals who are be targeted by malicious AI deepfakes.

## 6.4 Summary and Parting Words

In summary, this report provides an introduction to the machine learning methods used in the project and has outlined how machine learning has been used to firstly infer the cosmological parameters of astrophysical maps to an unprecedented degree of accuracy, surpassing the performance of previous papers to attempt the same task. Secondly, image-to-image diffusion has been used to generate astrophysical maps from input N-body maps with a good degree

of accuracy. Broader features of the output maps correspond well to the ground truth, yet higher frequency details are often missed, implying that a good amount of work in refining the network would be required in order to ensure the most accurate possible generation of new, useful astrophysical maps. Machine learning has proven to be a very useful tool in this application, and indeed can be used in a similar fashion in various other fields of science. Models such as diffusion are very exciting as they will no doubt continue growing and becoming more powerful over the next few years. This is certainly a new and developing field of research which will undoubtedly grow rapidly over the coming years as new, sophisticated machine learning models are being introduced at an unprecedented rate, setting the stage for very exciting coming years in the field of cosmological simulation generation.

# References

[1] A. Y. Grama, J. Fogarty, H. Aktulga, and S. Pandit, *N-Body Computational Methods*, pp. 1259–1268. Boston, MA: Springer US, 2011.

[2] F. Villaescusa-Navarro, D. Anglés-Alcázar, S. Genel, D. N. Spergel, R. S. Somerville, R. Dave, A. Pillepich, L. Hernquist, D. Nelson, P. Torrey, D. Narayanan, Y. Li, O. Philcox, V. La Torre, A. Maria Delgado, S. Ho, S. Hassan, B. Burkhart, D. Wadekar, N. Battaglia, G. Contardo, and G. L. Bryan, "The CAMELS Project: Cosmology and Astrophysics with Machine-learning Simulations," , vol. 915, p. 71, July 2021.

[3] F. Villaescusa-Navarro, S. Genel, D. Angles-Alcazar, L. Thiele, R. Dave, D. Narayanan, A. Nicola, Y. Li, P. Villanueva-Domingo, B. Wandelt, D. N. Spergel, R. S. Somerville, J. M. Zorrilla Matilla, F. G. Mohammad, S. Hassan, H. Shao, D. Wadekar, M. Eickenberg, K. W. K. Wong, G. Contardo, Y. Jo, E. Moser, E. T. Lau, L. F. Machado Poletti Valle, L. A. Perez, D. Nagai, N. Battaglia, and M. Vogelsberger, "The CAMELS Multifield Dataset: Learning the Universe's Fundamental Parameters with Artificial Intelligence," *arXiv e-prints*, p. arXiv:2109.10915, Sept. 2021.

[4] F. Villaescusa-Navarro, D. Anglés-Alcázar, S. Genel, D. N. Spergel, Y. Li, B. Wandelt, A. Nicola, L. Thiele, S. Hassan, J. M. Z. Matilla, D. Narayanan, R. Dave, and M. Vogelsberger, "Multifield cosmology with artificial intelligence," 2021.

[5] R. Weinberger, V. Springel, and R. Pakmor, "The AREPO public code release," *The Astrophysical Journal Supplement Series*, vol. 248, p. 32, jun 2020.

[6] J. H. T. Yip, X. Zhang, Y. Wang, W. Zhang, Y. Sun, G. Contardo, F. Villaescusa-Navarro, S. He, S. Genel, and S. Ho, "From dark matter to galaxies with convolutional neural networks," 2019.

[7] N. Mudur and D. P. Finkbeiner, "Can denoising diffusion probabilistic models generate realistic astrophysical fields?," 2022.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[9] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022.

[10] V. Jumle, "Image generation with diffusion models using keras and tensorflow," Jul 2022.

[11] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[12] K. Team, "Keras documentation: Denoising diffusion implicit models."

[13] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," 2021.

[14] F. Villaescusa-Navarro, S. Genel, D. Angles-Alcazar, D. N. Spergel, Y. Li, B. Wandelt, L. Thiele, A. Nicola, J. M. Z. Matilla, H. Shao, S. Hassan, D. Narayanan, R. Dave, and M. Vogelsberger, "Robust marginalization of baryonic effects for cosmological inference at the field level," 2021.

[15] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Lecture Notes in Computer Science*, p. 234–241, 2015.

[16] W. Yao, Z. Zeng, C. Lian, and H. Tang, "Pixel-wise regression using u-net and its application on pansharpening," *Neurocomputing*, vol. 312, p. 364–371, 2018.

[17] Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, p. 424–432, 2016.

[18] H. Lu, Y. She, J. Tie, and S. Xu, "Half-unet: A simplified u-net architecture for medical image segmentation," *Frontiers in Neuroinformatics*, vol. 16, 2022.

[19] A. J. R. S. e. a. Kugelman, J., "A comparison of deep learning u-net architectures for posterior segment oct retinal layer segmentation," 2022.

[20] J. K. W. N. Nicole Sanchez1, "Not so heavy metals: Black hole feedback enriches the circumgalactic medium," August 2019.

[21] J. P. Naiman *et al.*, "First results from the illustristng simulations: a tale of two elements – chemical evolution of magnesium and europium," *Monthly Notices of the Royal Astronomical Society*, vol. 477, pp. 1206–1224, 2018.

[22] F. Govoni and L. Feretti, "MAGNETIC FIELDS IN CLUSTERS OF GALAXIES," *International Journal of Modern Physics D*, vol. 13, pp. 1549–1594, sep 2004.

[23] S. Hong, G. Lee, W. Jang, and S. Kim, "Improving sample quality of diffusion models using self-attention guidance," 2023.

[24] E. Betzalel *et al.*, "A study on the evaluation of generative models," 2022.

[25] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022.

[26] T. T. Nguyen, Q. V. H. Nguyen, D. T. Nguyen, D. T. Nguyen, T. Huynh-The, S. Nahavandi, T. T. Nguyen, Q.-V. Pham, and C. M. Nguyen, "Deep learning for deepfakes creation and detection: A survey," *Computer Vision and Image Understanding*, vol. 223, p. 103525, oct 2022.

[27] V. L. L. Thing, "Deepfake detection with deep learning: Convolutional neural networks versus transformers," 2023.