Generating Kilonova Light Curves using Normalised Flow Machine Learning

Student Number: 2300431w

University of Glasgow

E-mail: 2300431w@student.gla.ac.uk

Abstract. During a neutron star merger event gravitational and electromagnetic radiation are emitted, the information within these may potentially allow us to further explore the inner workings of these otherwise mysterious objects. In order to effectively observe and study the electromagnetic "kilonova" element of neutron star merger events we require predictions of the kilonova lightcurves such that it is possible to effectively plan observations. Current models for generating said lightcurves are slow, unwieldy, and difficult to quickly implement in the event of a gravitational wave detection. In this paper it is shown that normalised flow machine learning models are a promising alternative, trained on data from said models to provide fast and powerful predictions that are far easier to implement. By creating training data using the Dietrich and Ujevic 2017 (DU17) Model a Machine Learning Flow Model (MLFM) was created which displayed close agreement with the DU17 model taking only the masses and tidal deformabilities of the neutron stars as inputs. Though providing excellent results when inputs are within the confines of the training data the MLFM fails to produce reliable results when data lies significantly outside the parameters used in the training data, this includes the priors from the 170817 neutron star merger event. This suggests that though the method is incredibly effective it requires a much larger range of input parameters to be widely reliable. The results also suggest that other kilonova models could be effectively predicted by a similar method and these models could even be combined into a single model.

1. Introduction

Neutron star merger events and their resulting kilonovae could be the source of heavy elements in the universe by providing a high energy neutron rich environment for the r-process to occus. In order to confirm this we must study the light emitted by these collisions to look for evidence of these elements forming. In order to do this we require a fast and efficient way to generate kilonovae lightcurves. The aim of this paper is to show that it is possible to accurately predict the lightcurves from neutron star merger gravitational wave data using normalised flow machine learning which will aid this rapid observation effort as normalised flow learning is substantially faster than traditional computational models.

In *Section 2* we will discuss the necessary background information on kilonovae and normalised flow machine learning. In *Section 3* we will discuss the method of data creation, preparation and training used to create the machine learning model. *Section 4* will cover the results of the training both against similar inputs and the inputs of a real neutron star merger event. *Section 5* will cover the flaws and potential improvments of the machine and finally *Section 6* will discuss the main conclusion of the paper.

2. Theory

2.1. What are Kilonovae

When two compact objects (such as neutron stars or black holes) merge in the universe they produce enough gravitational radiation to be detected on earth by large scale laser interferometers such as LIGO and VIRGO. If at least one of these objects is a neutron star then the merger will result in large amounts of incredibly neutron dense material being ejected at high velocity. This hot, fast moving, material results in an electromagnetic signal which we can detect through more conventional observation techniques. In 2017 a neutron star binary merger, designated GW170817, was detected via gravitational radiation as well as the electromagnetic counterpart. [2].

Before discussing the importance of neutron star mergers and the questions their observation could potentially help answer it is important to discuss the neutron stars themselves.

2.1.1. Neutron Stars

Neutron stars are made up of incredibly dense matter with masses up to approximately $2M_{\odot}$[11] (Though more recent estimates put the maximum mass to be $2.16M_{\odot}$[14]) packed into a stellar body with radius in the order of 10km [10]. Neutron stars are the remnant cores of massive stars ($M > 8M_{\odot}$). The production of heavy elements in these massive star's core's release large amounts of radiation pressure via nuclear fusion processes. This radiation pressure counterbalances the gravitational force due to the star's mass. The star will produce increasingly heavy elements until fusion results in Iron which does not release any energy, and hence no pressure, when it is fused. Thus, without the necessary radiation pressure the star will begin to collapse. Due to the extreme mass of the star the core becomes extremely dense. The collapse will result in a supernova and the creation of a neutron star (or a black hole if the core is densified sufficiently) formed from the late stellar core [4].

When a neutron star merges with another compact object the neutron rich matter is ejected into the universe. This provides an excellent environment for other elements present to undergo rapid neutron capture (a.k.a. 1the r-process') and form elements much heavier than Iron (e.g. Uranium, Plutonium). Currently theories suggest that this could be the dominant source of heavy nuclei in the universe, but further study is required before this can be confirmed [16].

In order to study the chemical makeup of the ejected material we need to study the electromagnetic emission lines emitted when certain elements are energised. This is an important field of research that can provide new insight into the origins of the chemical makeup of our universe. This will be further discussed in *Section 2.2*.

2.1.2. Kilonovae Light Curves

As previously mentioned, the ejected material of a neutron star binary merger (NS-NS) or neutron star, black hole merger (NS-BH) gives us kilonovae, the electromagnetic counterpart to the gravitational wave signal, allowing us to further probe this potentially vital and dominant r-process site in the universe. Such a search for the electromagnetic spectrum was carried out after the BH-BH Gravitational Wave GW150914 burst was detected by LIGO. However, given that this was a black hole merger event the resulting observations were unremarkable, though the exercise was useful in informing us how the follow up optical search for kilonovae may be conducted [3]. The process relies on alerting multiple observing teams for a short-term observation.

This method proved successful in 2017 when LIGO successfully detected a gravitational wave event and a follow up Electromagnetic observation[2].

The lightcurves of this electromagnetic component are the focus of this paper. They are defined by the absolute magnitude over a time spanning several days. Examples of these lightcurves as produced by mathematical models can be seen in *Figure 2.4*. The model we

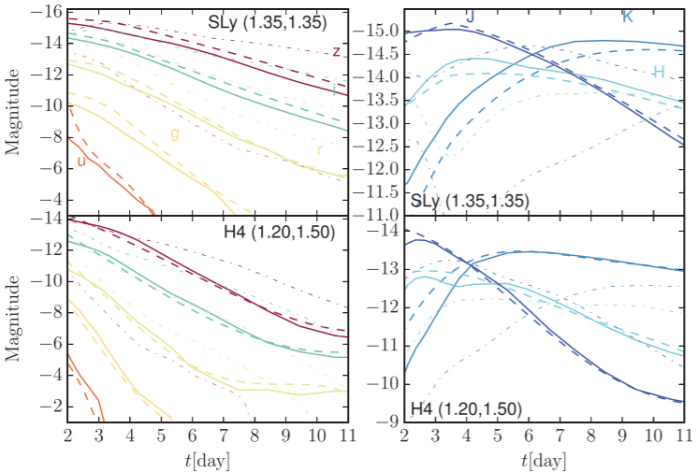


Figure 2.1. Kilonova lightcurves in the ugridz-bands (left panels) and the JHK-bands (right panel). Dashed lines represent the predictions from the DU17 model [8] while the solid lines represent data reported in [15]. The models show close agreement with the simulation data in both the Equal Mass (SLy) and Uneven Mass (H4) setups. These are an example of lightcurves we will be attempting to recreate with machine learning models. [8]

will focus on in this paper is the Dietrich and Ujevic 2017 (DU17) model. This was chosen as it showed the simplest features which, typically, makes the light curve easier to learn for the machine learning model we will discuss in the rest of the paper. Further examples of only the DU17 model can be seen in *Figure 2.1.* These curves are generated using a computational model that is difficult to implement and takes a relatively long time to run. Thus in order to effectively utilise the transient observation community mentioned in [3] we require kilonova models that can produce lightcurves much faster than current computational models and preferably in a more convenient method that is easier to implement widely.

2.2. Motivation for Studying Kilonova Light Curves

As mentioned in *Section 2.1*, neutron star mergers are a very important candidate for generating heavy nuclei in the universe. In order to explore the viability in greater depth we rely on traditional observations of the kilonova. By studying the lightcurves we can extract information about the nuclear material in the ejecta through their spectral lines and hence information about both the elements produced during the extreme conditions of the merger and potentially information about neutron stars themselves which to this day remain an exciting field of research filled with unknowns and unanswered questions.

Unfortunately, the observation of kilonovae is not as simple as one might hope. The mergers are relatively rare and the light that is received is often too dim for the majority of the observation time making them difficult to organise observations for. As mentioned previously, current models of kilonova light curve generation are computationally heavy and slow, making a fast response to a gravitational wave detection difficult. If it were possible to quickly generate these lightcurves then it could greatly aid the planning of these observations (i.e., by allowing observers to plan for the time of maximum amplitude) and hence allow us to further explore the possibilities held within these mysterious astronomical events. This issue of quickly generating light curves is one we seek to solve in this project utilising normalised flow machine learning models.

Machine learning provides a faster alternative to many complex problems by calculating outcomes not based on complex mathematical models themselves but by quickly applying a

series of weighted transformations.

2.3. DU17 Model Outline

The DU17 model is the simplest model available to us for recreation via machine learning. This section will serve as a brief outline of the model though the full derivation is found in [8] and the model used to create training data is based off of code from [7]. The goal is to demonstrate how we might begin with m_1, m_2, Λ_1, and Λ_2 and get a light curve as we will do computationally in *Section 3.1.* Firstly we convert Tidal Deformability $\Lambda 1$ and $\Lambda 2$ into the compactness using the C-Love compactness relationship[18]:

$$C = \sum_{k=0}^{2} a_k (ln\bar{\Lambda}_2)^k \tag{1}$$

where we use $\alpha_0^{YY} = 0.360$, $\alpha_1^{YY} = -0.0355$, $\alpha_2^{YY} = 0.000705$. We then combine m_1, m_2, C_1, and C_2 to generate the baryonic masses of the neutron stars m_{b1} and M_{b2} using *Equation (8)* of [6]:

$$m_b/m = 1 + aC^n \tag{2}$$

With $a = 0.8858$ and $n = 1.2082$. In order to model the light curve we must estimate the parameters of the ejected mass, this gives us an equation for the luminosity distribution over time as derived in [8]:

$$L(t) = (1 + \theta_{ej})\epsilon_{th}\dot{\epsilon_0} M_{ej} \begin{cases} \frac{t}{t_c}\left(\frac{t}{1\text{ day}}\right)^{-\alpha} & , t \leq t_c \\ \left(\frac{t}{1\text{ day}}\right)^{-\alpha} & , t > t_c \end{cases} \tag{3}$$

where 'θ_{ej}' is the polar opening angle of the ejecta, 'ϵ_{th}' is the thermal emissivity, $\dot{\epsilon_0}$ approximates heating for energy release via radioactive decay, 'M_ej' is the ejected mass, 't_c' is some critical time and 't' is time. 'α' is a constant taken as '$\alpha = 1.3$. We then derive the bolometric magnitude via:

$$M_{bol} = 4.74 - 2.5 log_{10}\left(\frac{L}{L_\odot}\right) \tag{4}$$

Finally to obtain a magnitude for a specific band we apply a bolometric correction defined as a quadratic equation where the coefficients change depending on the band. This correction and coefficients are given in *Equations (20), (21), and (22)* in [8].

2.4. Machine Learning

Machine learning (ML) refers to a type of computer algorithms which aromatically improve themselves as they are exposed to more data. As models and theories become more complex, they similarly become more computationally expensive, meaning they take much longer to run. Machine learning provides an alternative to these models that generally use simpler mathematical transformations that can be computed exceptionally quickly as computers favour this type of computation. If the model is given a large and diverse enough training set it should be able to produce results reliably with high confidence on data it has not seen before. However, there are difficulties with training a machine learning model that must be understood in order to understand the effectiveness of the machine learning algorithm built in this paper.

2.4.1. Basics of Machine Learning

The simplest form of machine learning are so called "neural networks" which will provide a simple basis to understand common terminology linked to Machine Learning.

There are two main types of features in a neural network. Firstly, nodes which represent a number with the leftmost nodes being the inputs and the right most being the outputs with several layers of 'hidden' nodes in between. Secondly, connections which are represented by the lines connecting nodes. These have a set weight value telling the neural network how much to pass the signal from the node the line begins at to the node it ends. These weights are what the machine alters throughout training to improve its performance. An example of a neural network can be seen in *Figure 2.2* which is a neural network with a single hidden layer.

In a purely mathematical sense, the input to a neural network is a column vector which undergoes various matrix multiplications from matrices holding the weight values until we reach the output values.

In order to train any type of machine learning model we must have some data which we know the input and the desired output, we call this dataset the training data. We then pass this data to the neural net, and it will produce results using randomly generated weights which it will then compare to the desired output. The machine then adjusts its weights to produce a result closer to the desired output. We quantify the accuracy of a prediction as the machine's loss, the greater the loss value the "further" the machine's prediction was from the correct value. the calculation of loss varies between different types of machine learning, as this paper does not focus on neural networks the relevant loss calculations will be discussed in *Section 2.4.5*. Throughout training we aim to reduce loss as training continues.

One particular issue ML systems run into is over-fitting, where the machine begins making predictions for a specific set of data rather than learning the broader relationship between inputs and outputs, though the loss will be exceptionally low the model will not be as effective when presented with new data. To determine if our machine is over-fitting, we partition our data such that while most is still used for training, we reserve some for "validation" where the predictions are tested against data which we know the answer to, but the machine has not been trained on. If the loss for the validation data is increasing while the loss falls for the training data, then we are likely seeing a case of over-fitting. To prevent over-fitting there are multiple methods, the simplest is to simply increase the amount of training data, this allows broad trends to be discovered without the required time to develop an over-fit. Another solution is to stop the training when validation loss begins to diverge from the training loss.

There are other important parameters known as hyperparameters that we can change before training a machine learning model that will help us improve training. By experimenting with these we can modify the machine to produce the best results for the given problem. The hyperparameters relevant to this paper are:

- **Epochs**: The number of times the machine runs through the entire data set
- **Batch Size**: The number of data points to cycle through before the machine alters itself
- **Learning Rate**: The step size the machine takes towards reducing its loss. If learning rate is too low, then the minimum loss will be approached too slowly. If it is too large the minimum loss will never be hit as the learning rate continuously overshoots.
- **Patience**: A factor that controls whether the machine stops learning early. If the change in loss is below patience for a set number of epochs the machine will stop learning to save time.

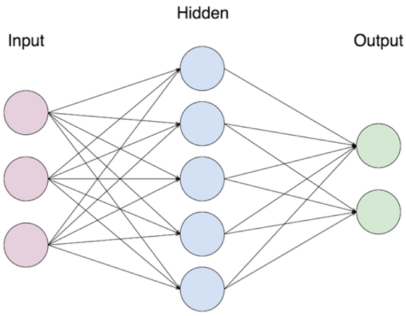


Figure 2.2. Diagram of a simple neural network with three inputs, five hidden layers and two outputs. This could be used to classify something with three input characteristics between two different classifications. [13]

2.4.2. Normalised Flows

While neural nets are the archetypal form of machine learning they are primarily used for data classification. In order to generate light curves however, we require a type of machine learning that can instead generate a complex continuous spectrum of predictions in the form of a probability density over some variable from a few pieces of input data. Normalised flows train by applying transforms to in the training data with weights defined by the input data where the machine attempts to eventually transform the input spectrum into a Gaussian curve (or another base spectrum). When using the model, it then reverses these steps to transform the base to produce a more complex prediction similar to the curves in the training data. These transformations are conducted using the Change of Variable Theorem.

2.4.3. Change of Variable Theorem

Presume we have an arbitrary variable 'z' with a known density function $p_1(z)$. We then want to construct a new random variable using a mapping function $x = f(z)$ where 'f' is an invertible function such that we can recover z using $z = f^{-1}(x)$. Of particular importance to normalised flow models is the ability to then infer the probability density $p_2(x)$ which uses our new variable.

The Change of Variable theorem states[17]:

$$p_2(x) = p_1(f^{-1}(x)) \left| det \left(\frac{\delta f^{-1}(x)}{\delta x} \right) \right| \tag{5}$$

Here '*det*' denotes the determinant and we have used the jacobian 'J':

$$J = \frac{\delta f^{-1}(x)}{\delta x} \tag{6}$$

We can simplify this further using the fact that $det(A^{-1}) = det(A)^{-1}$ for any invertible matrix A giving us:

$$p_2(x) = p_1(f^{-1}(x)) \left| det \left(\frac{\delta f(x)}{\delta x} \right) \right|^{-1} \tag{7}$$

This forms the mathematical basis by which normalised flow machine learning models transform between the complex spectra and the simpler base spectra.

2.4.4. Applying the Change of Variable Theorem
In order to predict a complicated probability density utilising machine learning we start with a simple distribution and apply a series of invertible transformation functions where we substitute the variable according to the Change of Variable Theorem until we reach a final probability distribution that uses the target variable. This is shown in *Figure 2.3*

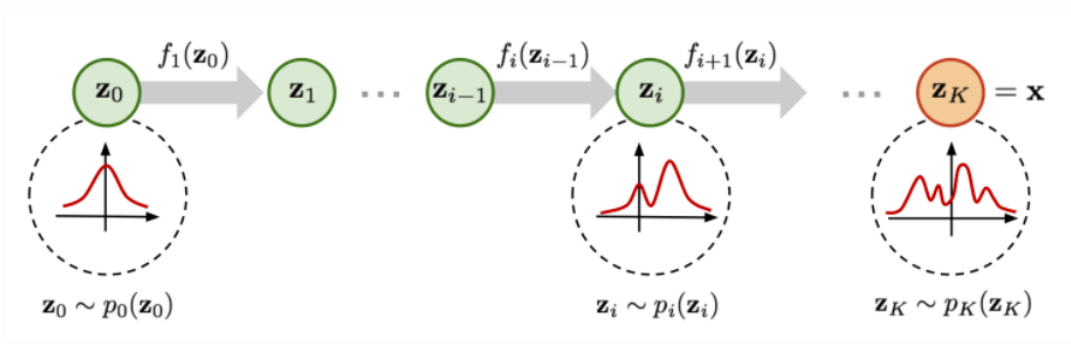


Figure 2.3. Simple diagram of how a normalised flow model transforms a simple base probability distribution into a complex one using reversible transformations. [17]

This can hence be used to help generate kilonova lightcurves where we use transforms to generate the complex shapes and using training data built from currently available numerical models that currently require significant computer processing effort and time. In contrast Models built using machine learning code can run exceptionally fast after the initial training as well as being far more widely accessible.

2.4.5. Loss in Normalised Flow Models
Loss is an important metric of the performance of our machine learning algorithm during training and understanding its calculation is important for our interpretation of the results. The loss used in the normalised flow machine learning algorithm used in this paper is defined via the Kullback–Leibler (KL) divergence[12]. The KL divergence defines the statistical "distance" between the two distributions $p_1(x)$ and $p_2(x)$. We can then determine loss through the KL divergence between the output distribution and the desired output:

$$\mathcal{L}(\theta) = D_{KL}[p_2(x)||p_1(x;\theta)] \tag{8}$$

which can approximated by Monte Carlo to give:

$$\mathcal{L}(\theta) \approx -\frac{1}{N}\sum_{n=1}^{N} log\ p_1(f^{-1}(x_n;\phi);\psi) + log\ |detJ_{f^{-1}}(x_n;\phi)| + \text{const.} \tag{9}$$

where 'ϕ' and 'ψ' represent the parameters of the transformation 'f' and the base distribution respectively. The full derivation is described by Equations (13) and (14) in [12]. It is important to note the constant term in *Equation 9*, this term depends on the data but is not calculated during training thus there exists a possibility for negative loss which is not expected in machine learning but is technically possible in normalised flows.

2.4.6. Flaws with Machine Learning Approaches
The quality of any Machine Learning model is subject to several pitfalls that we must be aware of if we are to fully understand the utility of the outputs. Since the model is entirely built by

training on a data set if the training data is inaccurate or inadequate it can result in a model with similar inadequacies and will hence not be useful to further study.

As mentioned in *Section 2.4* there is also a risk of under or over-fitting the data. Under-fitting occurs when we do not allow the model enough iterations to reach the true minimum loss which represents the maximum accuracy of the machine. Over-fitting, in contrast, can occur when we allow the training to continue for too long and the model begins to train too specifically for the data set provided instead of for the general relationships we are trying to reproduce. We can solve under-fitting by simply increasing the number of epochs over which we train and increasing the learning rate of the machine learning. To combat over-fitting, we must reserve some of our known data as a validation data set and stop training when the loss on our validation set begins to increase as our training loss continues to reduce. Another method for reducing over-fitting is learning rate scheduling where we reduce the learning rate over time while the machine trains.

Finally, we must be wary when training our data to ensure our data is inclusive enough to ensure that it will not be given any features that are out of the range that it was trained and tested with. If the model is given variables it has not been evaluated on it may not be as reliable as stated for that which it has. This can also be avoided by correctly identifying the limits on the features we will be planning to pass to the model after training and providing training and validation data as required to ensure it has the necessary inclusivity to avoid such problems.

Most if not all of these problems can be solved by providing a large amount of accurate training data to the machine learning algorithm, this is the common problem facing all machine learning training but one that is hopefully very manageable in this given context as we are building the training data from known mathematical models and can hence generate as much training data as we need in whatever ranges are necessary or pertinent to such models.

2.5. Applying Machine Learning to Kilonova Lightcurves

In *Section 2.4* we highlighted the need for large amounts of data covering all possible ranges of variables to be used in training our Normalised flow model. Unfortunately, this data doesn't exist as observations as kilonova are relatively rare, much lower than the tens of thousands of data points we would need for training, instead we must rely on theoretical models for kilonova lightcurves. In this project we will focus on a single model for generating kilonova lightcurves: The DU17 [8] model. This model was chosen as it shows the simplest behaviours and easiest to compute methods, thus will serve well as a proof-of-concept. The Figure 2.4 shows the computational results we expect for three separate models [1] (including the Du17 model), the aim of this paper is to demonstrate that machine learning can accurately recreate the DU17 model and hence should in theory be applicable through a similar method to the other models shown. By creating our own code to replicate these models we should be able to generate as much data as we need in order to reliably train our Normalised flow model when using combinations of inputs that follow the necessary constraints. Once this training data has been created and the normalised flow model trained, we should be able to rapidly recreate these models.

3. Method

As mentioned in *Section 2.5*, it was decided that in order to demonstrate the viability of normalised flow models to tackle this task the Dietrich 2017 model of kilonova lightcurves (DU17) [8] was chosen in order to create training data. The method and output were the simplest as well as having a lower computational cost. However, the chosen method of data creation is generalised for any function which can generate kilonova lightcurves given the parameters: mass of the heaviest neutron star, mass of the lightest neutron star, tidal deformability of the heaviest neutron star, tidal deformability of the lightest neutron star (m_1, m_2, Λ_1, Λ_2 respectively). Lightcurves were generated in the:

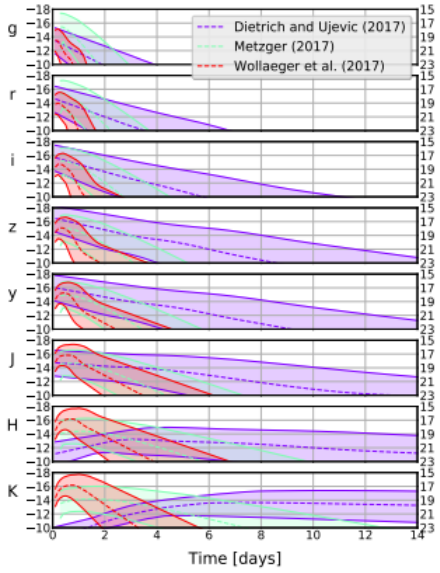


Figure 2.4. Example kilonova Light curves as generated by the DU17, Metzger, and Wollaeger models in the g,r,i,z,y,J,H,K bands. [1].

g ($\lambda_{eff} = 464nm$), r ($\lambda_{eff} = 617nm$), i ($\lambda_{eff} = 748nm$), z ($\lambda_{eff} = 893nm$) bands with 'λ_{eff}' the effective wavelength midpoint of the band.

3.1. Creating Training Data

In order to create the necessary training data, functions (as described in the original DU17 paper [8]) were taken from the source code provide in the gwemlightcurve python package [7]. A new function was created from the functions provided such that an input of m_1, m_2, Λ_1, Λ_2 would return the original input parameters, the time scale, and the associated lightcurves. This allowed a simple loop to process every set of input parameters. Due to the necessity for large data sets when training machine learning there were 100,000 individual combinations of m_1, m_2, Λ_1, Λ_2 provided from simulated data created through neutron star equation of state simulations[9][5]. Due to the high cost and time required to run the data creation function for such a high number of inputs the data was split into 16 sections and multiprocessing utilised to create 16 separate files containing lightcurves that were later recombined into a singular pandas document. The form of the database is shown in *Table 1.*

$M_1[M_\odot]$	$M_2[M_\odot]$	$\Lambda_1[gcm^2s^2]$	$\Lambda_2[gcm^2s^2]$	$t[days]$	g	r	i	z
1.8	1.1	163.3	163.3	[0.,...,10.8]	[-11.5,...,-3.5]	[-13.8,...-8.1]	[-15.5,...,-11.6]	[-16.7,...,-13.8]
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 1. DU17 Model Data in the g,r,i,z wavelength bands as generated by the function created using [7] with inputs from [9][5]. Data shown here is rounded for brevity's sake, in the database light curve values were returned with 8 decimal places and inputs to 17 significant figures

The data created had very high dimensions between 910 and 1101 depending on the time constraints used in the DU17 model. As discussed later in *Section 3.3.1* such high dimension

data proves challenging for Normalised Flow Machine Learning to provide reliable results so it was necessary to further prepare the data before using it in training.

3.2. Data Preparation

The first most necessary step was to reduce the resolution of the light curve significantly until the dimension of the curves were approximately 10 (i.e., the light curve was represented by 10 points). Due to the relative simplicity of the light curve models this significant drop in resolution does not remove any notable features of light curves. Were it the case that the loss in resolution also resulted in a loss of significant details another method of data compression would have to be chosen, namely Principle Component Analysis (PCA) discussed later in *Section 5*.

The second important step in data preparation was the removal of '*nan*' (not a number or undefined) values within the data array returned by the DU17 model code to suggest that the curve was no longer within the boundaries for which the model is designed to predict. This proves a problem when training the Model as the AI determines it's accuracy by calculating it's loss (*Section 2.4.1*, when '*nan*' values are present the loss itself is calculated as '*nan*' There were multiple methods used to generate data without'*nan*' values.

3.2.1. Cropping Method

The first method to remove '*nan*' values was to crop the data. Given the series of lightcurves a simple program was written to duplicate the original data and remove any curves that contained '*nan*' values between a given time range. In order to preserve as much data as possible the time range was defined such that approximately half of the original data would remain. An example of cropped data vs the original is shown in *Figure 3.1*

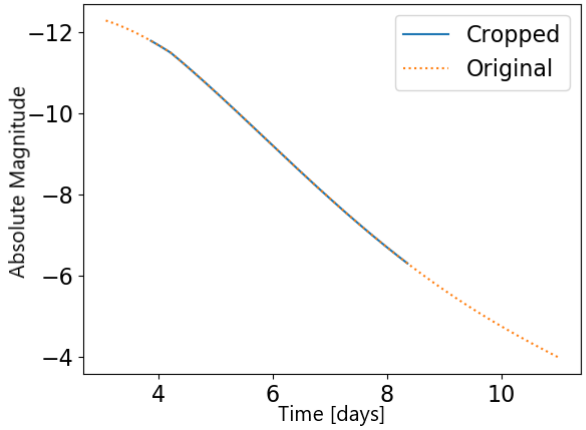


Figure 3.1. Cropped data (solid blue line) vs original data (dotted orange line) showing a light curve with absolute magnitude on the y-axis and time in days on the x-axis. This shows the overall result of the data cropping method as well as demonstrating the loss of information present even when the line has values outside of the cropping range.

There were multiple problems with this method of removing '*nan*' values. Firstly, it resulted in a loss of half of the data points favouring those with long lightcurves which would make our model significantly less useful over a wide range of inputs. Further to this it also reduced the range the model could predict lightcurves at all while the original models that the AI is trying to replace are capable of producing the data at a full range. Finally, this method was incredibly

time consuming, especially when compared to the more elegant solutions discussed in *Section 3.2.2 and 3.2.3* below.

3.2.2. Nan to Zero Method

Another alternative method to generate training data without '*nan*' values was to convert ever '*nan*' value to a 0.0 floating point value. This method was significantly simpler than the others as well as significantly faster than the Cropping Method. This method however did result in artifacts after training as seen in *Figure 3.2* where the 3^{rd} curve, 'Flow[2]', displays clear artifacts at the beginning and end of the prediction. However *Figure 3.2* also shows that now our model can accurately predict lightcurves that are much shorter (shown by 'Flow[1]') and lines that are entirely comprised of '*nan*' values (Flow[0]).

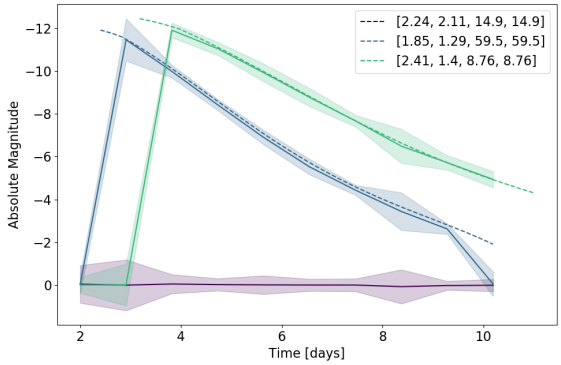


Figure 3.2. Three example predictions made from randomly chosen input parameters shown in the legend with the format '[m_1, m_2, Λ_1, Λ_2]' on a trained machine learning model on the g-band data where *'nan'* values were converted to 0.0. It is clear to see that this method allows for accurate predictions of lightcurves however it results in artifacts such as those seen at the beginning and end of the blue and green lightcurves while some result in constant lightcurves around 0. The filled region represents the 3σ (99.73%) confidence interval

3.2.3. Nan to Number Method

Though the Nan to Zero method worked well there was a final improvement to be made. Instead of converting *'nan'* values to '0.0' they were converted to the nearest value in the curve i.e., *'nan'* values at the beginning of the curve would be converted to the first non-*'nan'* value on the curve while *'nan'* values at the end of the curve were converted to the last non *'nan'* value on the curve. If all values were *'nan'* values then all *'nan'* values were converted into 0.0.

This method had the same main advantage as the Nan to Zero method in *Section 3.2.3* where it maintained the entirety of the data set while also being faster than the Cropping Method in *Section 3.2.1* (though marginally slower than the Nan to Zero method). The major advantage over the Nan to Zero method was that the artifacts produced during training were less noticeable and hence providing a slightly more accurate estimate of the original model even when the training data had a *'nan'* value. For these reasons the data created through this method is what the rest of this paper will focus on and the final model was trained on.

3.3. Training

In order for machine learning models to produce useful predictions it must be trained on the data prepared as per previous sections. An important constraint that this imposes is that the model

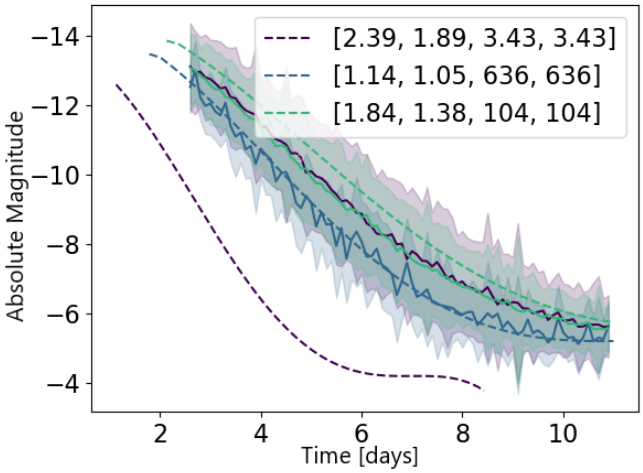


Figure 3.3. Three test lightcurves in the r-band predicted by an early machine learning model using three randomly chosen inputs during training. Inputs are labelled using the format [m_1, m_2, Λ_1, Λ_2]. Dashed lines represent the predictions made by the original DU17 model while solid lines represent the average light curve from 100 predictions made by the machine learning model with the shaded region representing the standard deviation of the curves.

will only be useful when tested on parameters that exist in the training data. If parameters are not sufficiently covered within the training data, the machine learning model will not be able to reliably make predictions.

In order to simplify the training and debugging process four machine learning models were trained separately such that each only predicted one light curve band. After training, these four separate models could be used together to predict a single light curve in multiple wavelength bands. While training was separate, hyper-parameters and processes relating to training were constant across all four models and thus when discussing "The Model" later we refer to both the individual models and their combined result.

A significant measure of the quality of a machine learning model as it trains is it's loss which we expect to fall as the machine becomes more accurate.

3.3.1. Issues During Training

Initial training was performed using cropped data (as discussed in *Section 3.2.1*) with a relatively high dimension of 91 data points per lightcurve. While training the machine showed promising loss curves as they steadily decreased with time using a constant learning rate. Once training was completed the model was tested on three random sets of inputs. 100 predictions were made so that the average prediction could be find along with the standard deviation. This was plotted alongside predictions from the DU17 model. The results of this test are shown in *Figure 3.3*. From *Figure 3.3* we can see that the model has not provided insightful predictions based on the input parameters. Instead, we see the model is providing incredibly noisy predictions, with large uncertainty, around an apparent "average" light curve output in order to minimise loss. The second, blue, curve '*Model[1]*' is not predicted at all by the machine as we would expect if the training had been successful.

3.3.2. Resolving Issues with Training

The initial training showed several issues. The first was a significant level of noise, this likely

stemmed from the high dimensionality of the training curves. Such high dimensions increase the complexity of the analysis required by the machine during training and results in significantly less powerful predictions. Another potential source of the high uncertainty and noise is the low amount of data as more than half was omitted due to the cropping algorithm run to prepare the data. Another evident issue is that the curve is not predicting the overall relationship between the inputs and the output curve and rather finding the average lightcurve. This further suggests that we need a larger set of training data as well lower resolution lightcurves to simplify the issue. It also suggests that a constant learning rate does not allow the machine to learn the necessary complexity of the model and thus we should implement a varying learning rate via a learning rate scheduler.

Therefore, the following were used to improve the training of the machine. Firstly, a learning rate scheduler was introduced to gradually reduce the learning rate as the machine trained. The machine trained for 1500 epochs in total. The first 500 epochs used a learning rate of 1×10^{-3}, the second 1×10^{-5}, and the final a learning rate of 1×10^{-7}. Secondly the light curve resolution was dropped from 91 to 10. Finally the 'nantonum' method discussed in *Section 3.2.3* was used to prepare data and allow for the largest amount of data to be used in training with as minimal a loss in data accuracy given the low resolution. The exact training hyperparameters used when training the final machine learning model can be found in **Appendix A**

4. Results

There are two important aspects to look at when assessing whether the machine was successfully trained. The first is to see how well it performs on data within the training and validation data. Secondly, it is worthwhile testing the model on the data from the 170817 detection of a NS-NS merger where the DU17 model, among others, were used to predict the kilonova lightcurves in *Figure 2.4* and compare the machine model's output to that of the more traditional computational models.

One important goal of this project was to produce a machine learning algorithm that was capable of producing results much faster than traditional computational models, in theory this will be true as machine learning models are typically very computationally quick. In a simple test conducted at the end of this project the DU17 model's code base and machine learning model were compared in how long it took each to complete the following task. Both were asked to create lightcurves for 1000 combinations of inputs, the machine was asked to create 100 predictions for each of these which would be standard when making predictions so that we could create an average value with uncertainty. In this test the machine learning model took 0.64s while the DU17 model code took 150.16s, a 99% reduction in processing time. This is a significant decrease in time and computational resources. However, it is worth mentioning that the model alone was actually marginally faster than the machine when producing a single lightcurve. In the reality of gravitational wave analysis however, it is not likely the model will be needed to produce a single output for a single combination of inputs but rather 100s to thousands based on the distribution of prior values of the gravitational wave analysis.

Further to this we can analyse this difference in computational time by plotting the time taken to complete 'n' tasks as seen in *Figure 4.1.* In *Figure 4.1* we can see that while for single lightcurves the computational models are marginally faster they have a computational complexity of $O(n)$ while the machine learning algorithm has a computational complexity of $O(1)$. This means that as the problem scales the machine learning model takes the same amount of time while the traditional models grow linearly slower.

4.1. Results on Training Data

An important feature for assessing the model's efficacy on the training data is its loss. As we can see in *Figure 4.2* the training loss consistently falls while the data is trained. In the

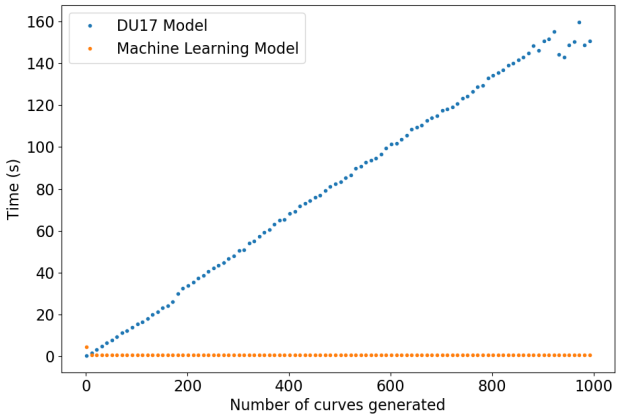
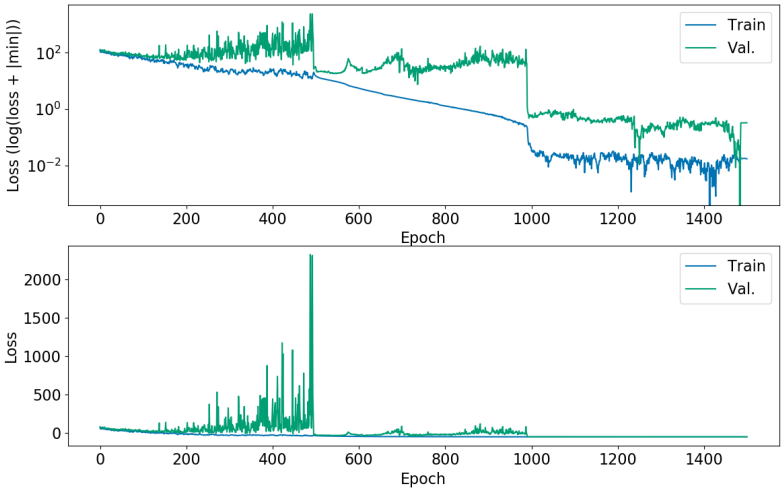


Figure 4.1. The time taken to generate a given number of lightcurves. As we can see the Traditional model increases in time cost at approximately O(n) complexity while the machine learning model shows a complexity of O(1) showing that the ML model is generally much faster, especially for a large number of light curve generations. Modelling a straight line of best fit we find that the DU17 Model has a gradient $m = 0.16$ and intercept $c = 0.82$. The Machine learning model has $m = -1.81 \times 10^{-04}$ and $c = 0.73$



5

Figure 4.2. Loss curves of the machine training in the g band, typical for the machine learning process. The top graph shows the loss plotted on a log scale with the addition of the minimum value as normalised flow curves commonly had negative loss. The Bottom shows the loss with regular y-axis showing the relatively small scale of loss changes at later stages

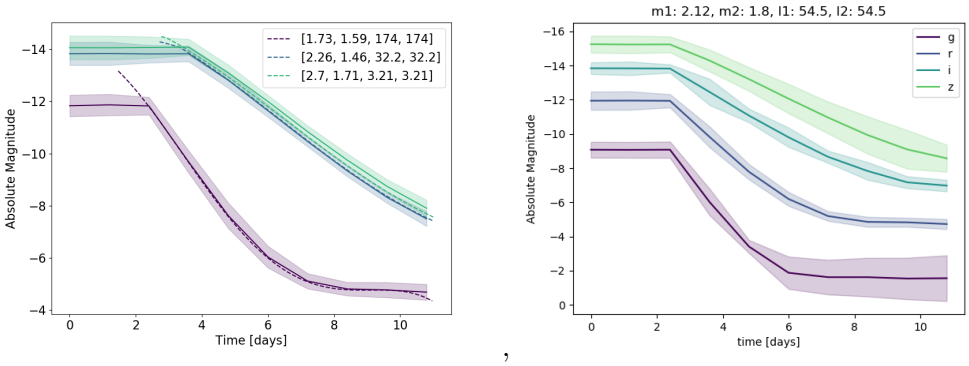


Figure 4.3. Example curves predicted by the Flow model in the g-band (left) and predictions made for all four bands using one combination of inputs (right). Dashed lines represent the predictions made by the DU17 model and the solid lines represent the average of 100 predicted lines from the model, the shaded area represents the 3σ (99.73%) confidence interval. The input parameters of the curves on the left hand diagram are given in the legend using the format [m_1,m_2,Λ_1, Λ_2]. The inputs used to generate the right figure are given in the plot's title. Further examples of the machine learning model on the g, r, i, and z bands can be found in **Appendix B**

first 500 epochs we see the tell-tale signs of over training where the validation data begins to increase while the training loss decreases. This trend becomes somewhat more severe after a significant decrease in validation loss during epochs 500-1000 as the validation loss remains relatively constant while training loss falls consistently. In the final 500 epochs we see a similar sharp decline in the validation loss before both remain constant at low values. This is somewhat promising as loss is very low however the rise in validation loss during the first 1000 epochs suggests there are still improvements that could be made to the training process discussed later in *Section 5*. Beyond the loss curves we can also test the data by giving it values of m_1,m_2,Λ_1, Λ_2 from the data used in the training and validation data. In *Figure 4.3* we can see that the model is capable of recreating lightcurves to a high degree of accuracy. We can also see artifacts from the data preparation method (NanToNum *Section 3.2.3*) in the earlier values of the predictions.

From this we can see that given the appropriate training data a normalised flow model can effectively reproduce the DU17 kilonova light curve model, this also suggests that a similar process of data creation and training would allow for more models to be reproduced via machine learning.

4.2. Results on 170817 Data

Another important test of the model's effectiveness is its ability to create lightcurves for the 170817 NS-NS merger event as the traditional models shown in *Figure 2.4* have been able to, and thus in theory our model should be capable of recreating if training has been successful. As shown in *Figure 4.4* we can see that the machine learning model has failed to generate useful predictions for the lightcurves in the same way as the original model. So, although we have successfully shown in *Section 4.1* that the model can in principle reproduce results it has been trained on there are still substantial flaws with either the training data or training method itself.

5. Discussion

The results mentioned in *Section 4.2* suggest that our machine learning model has fallen victim to some of the flaws described in *Section 2.4.6*. The first possibility is that of over-fitting. In

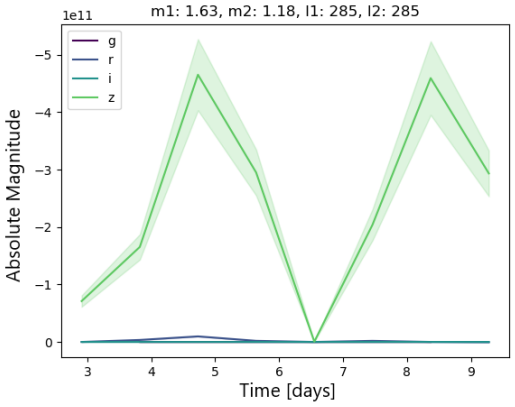


Figure 4.4. Graph showing the g,r,i, and z band predictions on the 170817 data priors by the normalised flow model showing that the machine learning model has produced unreasonable results for the 170817 merger priors

Figure 4.2 we can see three distinct phases in the training, the border between which can be seen as the large drops in the validation loss. Though it is important to note we can also see the characteristic of over-fitting where the validation loss grows, particularly in the first 1000 epochs. However, this effect is no longer seen in the final phase where validation loss drops greatly. This suggests that though over-fitting was a problem in training the learning rate scheduling managed to counteract this issue even with a difference between training and validation loss. To further investigate this potential issue another model was retrained on $\frac{15}{16}$ of the total data. When the model was then tested on the known omitted $\frac{1}{16}$ of data showed similar results to the data which is was trained on. This suggests that over-fitting was not the culprit. However from *Figure 4.2* we can see that there are improvements to be made to the training hyper-parameters, perhaps shortening the length of epochs spent using higher learning rates.

Another potential problem could be that the inputs of the 170817 priors lie out with the training dataset's inputs. To see this we can simply plot the distribution of the four input parameters shown in *Figure 5.4*. In this graph we can see that though in the initial 170817 distribution some values of m_2, Λ_1, and Λ_2 are not represented within the training dataset the vast majority are. However in the NS-NS priors we see that the values of the inputs are entirely contained and represented within the training data despite also producing a similar prediction to *Figure 4.4*. Thus, it is not likely that the individual inputs lying outside of the training data is the reason for the model's inability to provide useful predictions for the 170817 model.

Though the values themselves are represented by the training data there is a potential that the specific combinations of data are not represented within the training data. To investigate this we would ideally plot our distributions in a 4D space but this can prove challenging to represent concisely so instead we look at the plots of m_1 vs m_2 and Λ_1 vs Λ_2 as shown in *Figure 5.2*. *Figure 5.2.* shows that though the mass values are contained within the training data the lambda values from the 170817 collisions vary substantially from the straight line of Λ values used in the training data. This suggests that the machine learning model is not coping well with noisy values of Λ which deviate from the strict combinations provided in the training data. In order to further test this hypothesis that it is the lack of noise in the Λ values causing the issues with our machine learning we should attempt to create new data that adds a small amount of noise to the original data which in principle would improve the model should the noise in the 170817 data be the issue.

Asides from the failures described above and in *Section 4.2* another issue found within the data

is an unreliability when tested on lightcurves where the model returned a very short lightcurve, an example of which can be seen in *Figure 5.1.* This is most likely due to the combinations of inputs that result in short time frames being underrepresented in the training data. In *Figure 5.1* we can see that though the more typical lightcurves commonly seen in our training data are predicted with a high accuracy the shorter light curve shows a lot of uncertainty. This means our model is less accurate on lightcurves which are predicted for shorter timeframes. In order to improve this, it is worth investigating which combinations produce such lightcurves and trying to create more training data to accommodate them. This uncertainty may also arise from the method of removing '*nan*' values as discussed in *Section 3.2.3*, however this does not seem likely as the method has proven successful for the wide variety of other lightcurves.

As shown in *Figure 5.1* the machine learning model can more accurately predict 'long' light curves (lightcurves which last for 5 days or more) while it gives larger uncertainty for shorter lightcurves, since the lightcurves are a result based on our four parameters we should be able to see the length of light curve exist as a function in parameter space. This can be seen in *Figure 5.3* where we have plotted m_1 and m_2 with the colours representing the percentage of the light curve generated which is a *'nan'* values. This varied significantly in the mass space but not as much in the Λ space which can be seen in **Appendix C**. This suggests there are values of m_1 and m_2 for which the model produces short lightcurves or none at all and thus the machine learning model is less accurate. This logically implies that these values are beyond the scope of the model as opposed to the light curve not existing at all. Through use of other models this might be explored further.

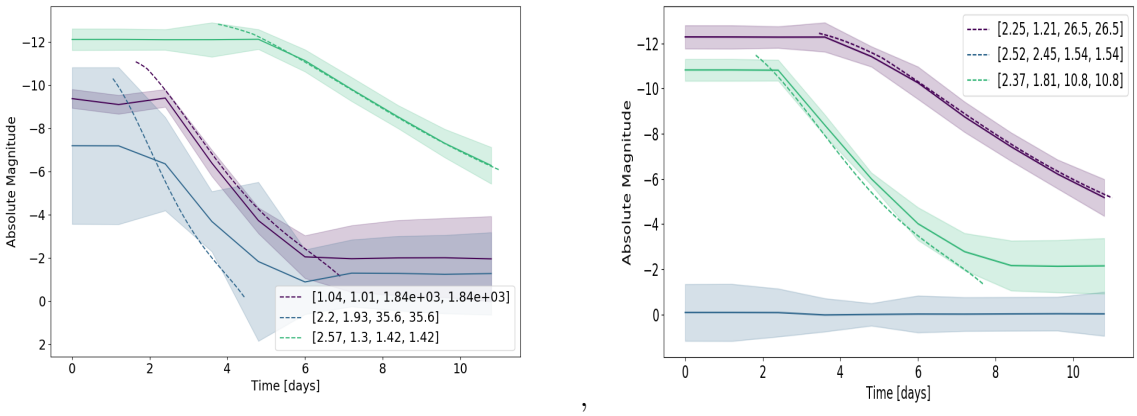


Figure 5.1. Testing the model on six random unseen parameters in the 'g' band. On the left hand graph, we can see the typical result when trying to reproduce a short kilonova lightcurve. In both figures we can see that the model is very accurate for longer lightcurves. The input parameters are given in the format [m_1, m_2, $\Lambda 1$, $\Lambda 2$]

This paper has primarily focused on the recreation of the DU17 model however as we can see in *Figure 2.4* the DU17 model is not the only one available to us. A further improvement to the system in general could be the training of other models using the same pipeline which we have shown can be effective (within the constraints of the training data). This would allow for more complex predictions that reproduce multiple models in an incredibly short timeframe. However, this task is beyond the scope of the project discussed here.

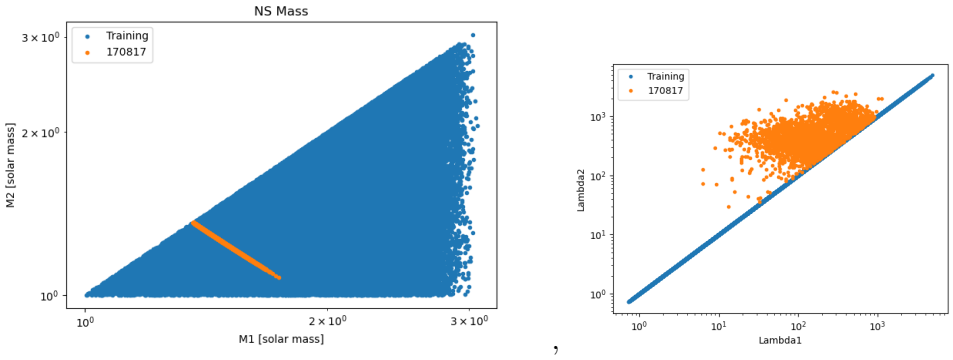


Figure 5.2. Graphs showing the distributions of m_1 vs m_2 and Λ_1 vs Λ_2 for both the training data and the 170817 collision data showing a high overlap with training data and the collision data in mass space but little overlap in the Λ space suggesting the machine learning model struggles with the additional noise

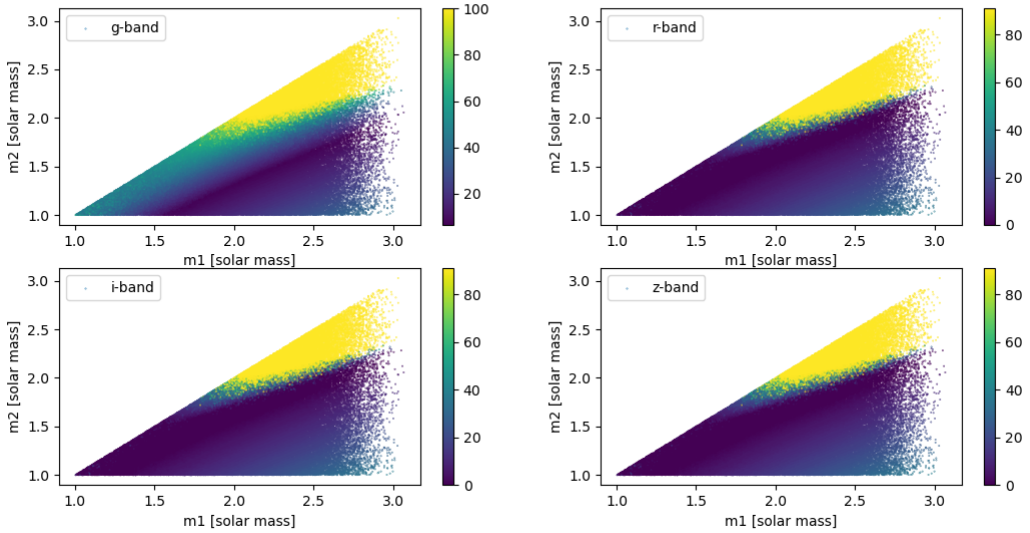


Figure 5.3. Four graphs showing the percentage of graphs made of *'nan'* values in the g,r,i, and z bands. The higher the number of *'nan'* values the shorter the curve showing that the g-band is typically the shortest light curve as expected and for values of m_1 and m_2 the DU17 model produces very short lightcurves.

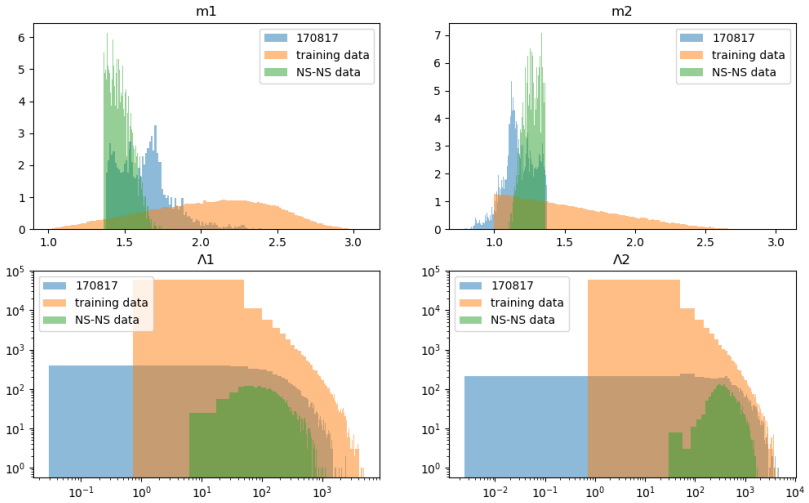


Figure 5.4. The normalised distributions of m_1, m_2, $\Lambda 1$, $\Lambda 2$. The blue (170817) distribution shows the priors of the 170817 observation of potential values without assumption on the nature of the two bodies. The green (NS-NS) distribution shows the priors of the 170817 observation where it assumed both bodies are Neutron Stars. The orange (training data) represents the training data distribution. Graphs have been normalised such that their total areas are equal to one so that they are easier to compare

6. Conclusion

In this paper we have shown that normalised flow machine learning models are a powerful predictive tool that in principle can be used to recreate predictions from the Dietrich and Ujevic 2017 model with sufficiently accurate predictions to allow for a far more rapid method of predicting kilonova lightcurves showing close agreement for a wide variety of mass and tidal deformabilities as seen in *Figure 5.1*. We can also see in *Figure 5.1* that there are some combinations of inputs that result in higher levels of uncertainty even if the overall relationship is estimated to a reasonable degree. This suggests that other models for kilonova lightcurves could be accurately recreated using a similar methodology and possibly even combined using machine learning.

We have also shown that the Machine learning model has a computational complexity of $O(1)$ compared to the DU17 model's complexity of $O(n)$. Thus the machine learning model scales significantly easier than the original model.

In spite of the success during training we can see in *Figure 4.4* that the machine has failed to predict sensible lightcurves when confronted with data from the 170817 neutron star merger. The machine predicted a lightcurve with maximum absolute magnitudes of order -5×10^{11} which is unreasonable.

There are several reasons why the machine learning model has not produced a reasonable result withe priors from the 170817 neutron star merger. Firstly the $Lambda_{1,2}$ values had no noticeable noise in the training data while the priors from the 170817 merger had noise resulting from the statistical analysis of the gravitational wave signal. It is not likely that the prediction is due to over-fitting as the model performs well on data it has not seen before.

Acknowledgements
Special thanks to Prof. Ik Siong Heng for his help throughout the project and for providing consistent feedback, help, and insight.

Also thanks to Jordan McGinn for providing the combinations of neutron star masses and tidal deformabilities from his own research.

Finally, I would like to thank Michael Williams and Federico Stachurski for their help during the training stage of the AI model's development.

References

[1] B. P. Abbott et al. In: *The Astrophysical Journal* 850.2 (2017), p. L39. URL: http://dx.doi.org/10.3847/2041-8213/aa9478.

[2] B. P. Abbott et al. In: *Phys. Rev. Lett.* 119 (2017), p. 161101. URL: https://link.aps.org/doi/10.1103/PhysRevLett.119.161101.

[3] B. P. Abbott et al. In: 826.1 (2016), p. L13. URL: https://doi.org/10.3847/2041-8205/826/1/l13.

[4] Pablo Cerda-Duran and Nancy Elias-Rosa. In: *Astrophysics and Space Science Library* (2018), pp. 1–56. URL: http://dx.doi.org/10.1007/978-3-319-97616-7_1.

[5] Jordan McGinn (personal communications). URL: https://github.com/jmcginn.

[6] Michael Coughlin et al. In: *The Astrophysical Journal* 849.1 (2017), p. 12. URL: http://dx.doi.org/10.3847/1538-4357/aa9114.

[7] Scott Coughlin. URL: https://gwemlightcurves.github.io/.

[8] Tim Dietrich and Maximiliano Ujevic. In: 34.10 (2017), p. 105014. URL: https://doi.org/10.1088/1361-6382/aa6bb0.

[9] Igr-Ml. URL: https://github.com/igr-ml/glasflow.

[10] Débora Peres Menezes. In: *Universe* 7.8 (2021), p. 267. URL: http://dx.doi.org/10.3390/universe7080267.

[11] Feryal Özel et al. In: *The Astrophysical Journal* 757.1 (2012), p. 55. URL: http://dx.doi.org/10.1088/0004-637X/757/1/55.

[12] G. Papamakarios et al. In: *Journal of Machine Learning Research* 22.57 (2021), pp. 1–64. URL: https://arxiv.org/abs/1912.02762.

[13] Marco Peixeiro. 2020. URL: https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e.

[14] Luciano Rezzolla, Elias R. Most, and Lukas R. Weih. In: *The Astrophysical Journal* 852.2 (2018), p. L25. URL: http://dx.doi.org/10.3847/2041-8213/aaa401.

[15] Masaomi Tanaka and Kenta Hotokezaka. In: 775.2 (2013), p. 113. URL: https://doi.org/10.1088/0004-637x/775/2/113.

[16] F.-K. Thielemann et al. In: *Progress in Particle and Nuclear Physics* 66.2 (2011). Particle and Nuclear Astrophysics, pp. 346–353. URL: https://www.sciencedirect.com/science/article/pii/S0146641011000330.

[17] Lilian Weng. 2018. URL: https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html.

[18] Kent Yagi and Nicolás Yunes. In: *Physics Reports* 681 (2017), pp. 1–72. URL: http://dx.doi.org/10.1016/j.physrep.2017.03.002.

Appendix

A. Training Hyperparameters
The following hyper parameters were used when training the machine learning model in this paper.:

Hyperparameter	*Value*
Total Epochs	1500
Initial Learning Rate	1×10^{-3}
f	$\frac{1}{3}$
γ	0.01
Test/Validation Split	$\frac{1}{3}$
Batch Size	1000
Patience	0.01

Table 2. Hyper parameters used when training the final machine learning model

The Total Epochs is the total number of iterations through the entire set of training data that the machine will run over during training assuming the process is not stopped early. The Initial learning rate is the initial learning rate used in the learning rate scheduler (As discussed in *Section 3.3.2*). 'f' refers to the fraction of epochs at which the learning rate will be multiplied by 'γ' in order to reduce it over time. The 'Test/Validation Split' refers to the fraction of data reserved for the validation data set used for analysing the effectiveness of our model on data it has not seen before. The Batch Size is the number of samples processed before the model updates. Finally, the 'patience' is a threshold parameter that if the change in loss (ΔLoss) is below for 50 consecutive epochs the training will stop. In practice this given value was so small it rarely stopped the program.

This configuration was the optimal combination that I found. Shorter Total Epochs did not show as promising training results however there was little difference. A learning rate of $1 \times 10^{-}5$ was the most effective however without starting at a relatively high learning rate it would not suffice alone. Similarly the machine learning model showed huge spikes in loss when γ was 0.1 though theoretically it shouldn't have as large an impact.

B. Further Results

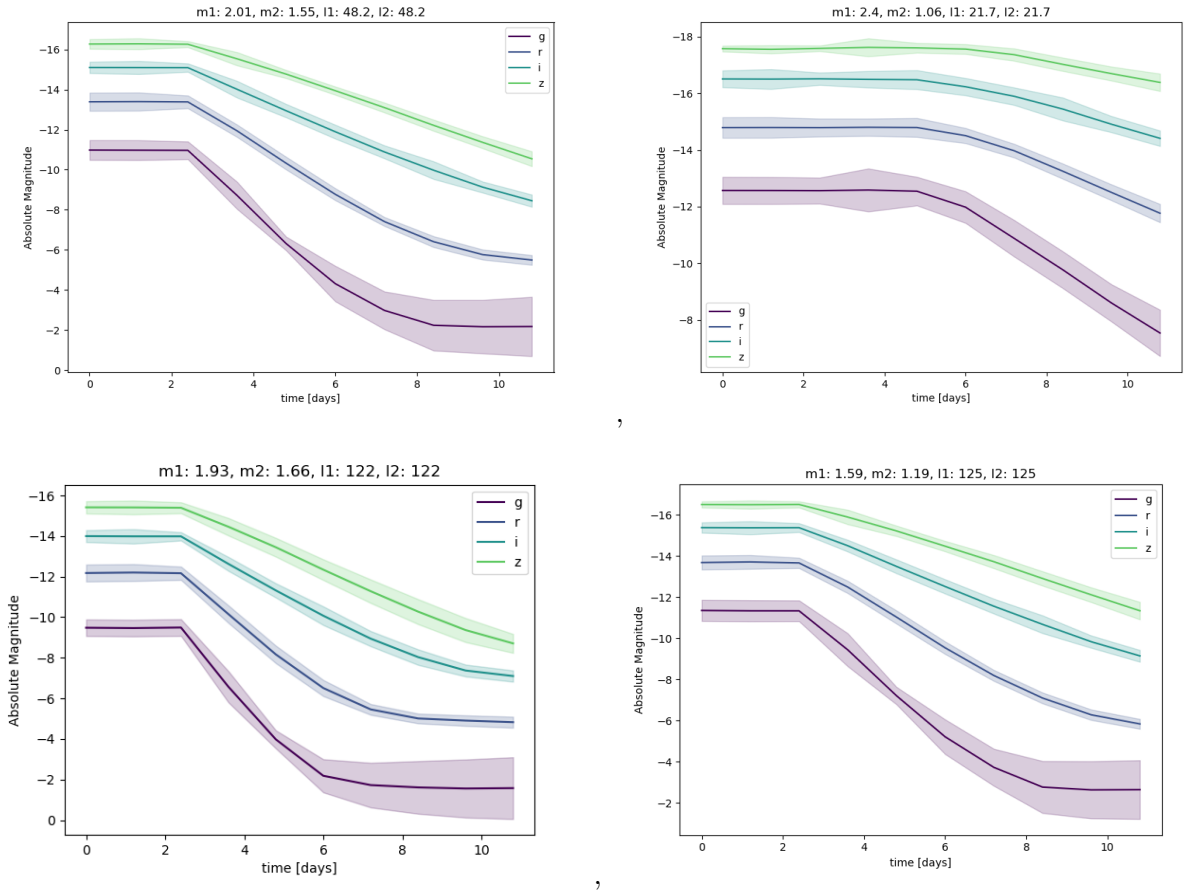


Figure B.1. Further examples of DU17 model predictions in the g, r, i, and z bands. The shaded region represents an area of 3σ confidence with input parameters given in the titles.

C. Additional Analysis of Inputs

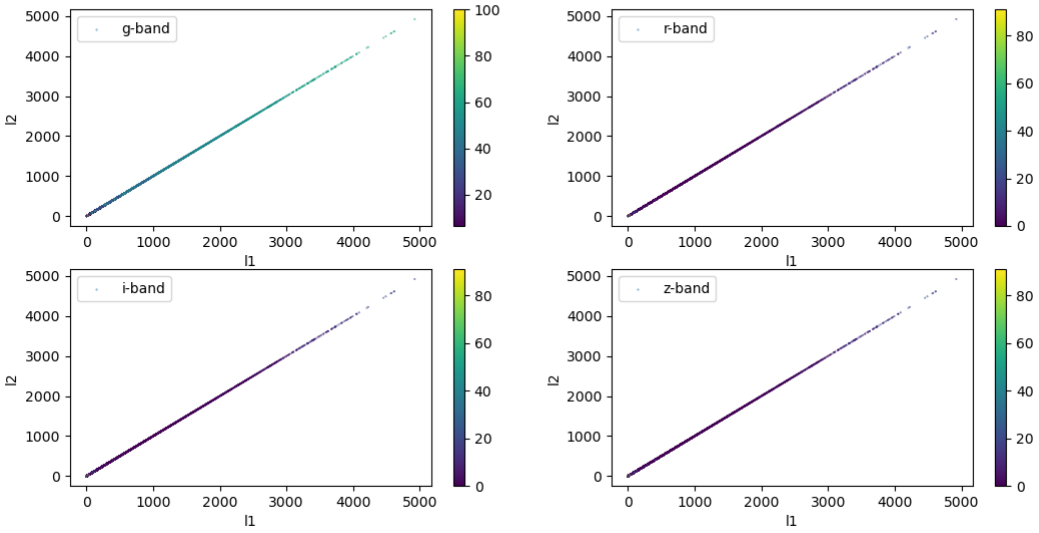


Figure C.1. Plotting the percentage of lightcurves that are made of '*nan*' in Λ space. showing that there is some correlation between Λ_1 and Λ_2 in the g-band but not so much in the r, i, and z bands.