Gravitational-wave surrogate models powered by artificial neural networks

Sebastian Khan and Rhys Green
School of Physics and Astronomy, Cardiff University, The Parade, Cardiff, Wales CF24 3AA, United Kingdom

(Received 6 November 2020; accepted 26 January 2021; published 10 March 2021)

Inferring the properties of black holes and neutron stars is a key science goal of gravitational-wave (GW) astronomy. To extract as much information as possible from GW observations, we must develop methods to reduce the cost of Bayesian inference. In this paper, we use artificial neural networks (ANNs) and the parallelization power of graphics processing units (GPUs) to improve the surrogate modeling method, which can produce accelerated versions of existing models. As a first application of our method, the artificial neural networks surrogate model (ANN-Sur), we build a time-domain surrogate model of the spin-aligned binary black hole (BBH) waveform model SEOBNRv4. We achieve median mismatches of approximately $2e-5$ and mismatches no worse than approximately $2e-3$. For a typical BBH waveform generated from 12 Hz with a total mass of 60 $M_\odot$, the original SEOBNRv4 model takes 1794 ms. Existing custom-made code optimizations (SEOBNRv4opt) reduced this to 83.7 ms, and the interpolation-based, frequency-domain surrogate SEOBNRv4ROM can generate this waveform in 3.5 ms. Our ANN-Sur model when run on a CPU takes 1.2 ms and when run on a graphics processing unit (GPU) takes just 0.5 ms. ANN-Sur can also generate large batches of waveforms simultaneously. We find that batches of up to 10^3 waveforms can be evaluated on a GPU in just 1.57 ms, corresponding to a time per waveform of 0.0016 ms. This method is a promising way to utilize the parallelization power of GPUs to drastically increase the computational efficiency of Bayesian parameter estimation.

DOI: 10.1103/PhysRevD.103.064015

I. INTRODUCTION

The swift and accurate computation of the gravitational-wave (GW) signal from merging compact binaries is a crucial part of GW astronomy. Over the past few years, enormous progress has been made in modeling the GW signal [1–20], and recent models have played important roles in the analysis of recent GW events [21,22]. However, as waveform models relax simplifying approximations (such as including subdominant multipoles), the computational cost tends to increase, which ultimately limits their use in GW analyses.

To reduce the computational cost of generating waveforms, the community has developed several custom-made optimizations [23–25]. But these typically require expert knowledge and might not provide general optimizations that other models can incorporate. There are many methods to accelerate Bayesian parameter estimation [26–34], but in general, they each make simplifying assumptions that mean not all waveform models can readily take advantage of the potential speed-up. Another way to accelerate analyses is by parallelization. Typically, this means parallelizing your analysis across multiple CPUs; however, there has been growing interest in the use of graphics processing units (GPUs), see Refs. [35–40] for applications in GW astronomy.

Alternatively, data-driven methods can be employed that are waveform model agnostic and hence are of great interest. One such method is called surrogate modeling [41]. Here, one attempts to build a fast and accurate approximation (a surrogate or emulator) of a slower model. A successful way to build these models typically begins with building a reduced basis representation (e.g., a singular value decomposition or greedy reduced basis) of the model [41–47]. One of the biggest issues in reduced basis surrogate modeling is the approximation of the reduced basis coefficients. This is a multidimension interpolation or regression problem and has recently been investigated in Ref. [48], in which the authors systematically compared different interpolation and regression methods.

In this work, we train artificial neural networks (ANNs), developed with the TensorFlow [49] library, to accurately and efficiently estimate the projection coefficients of a reduced basis representation. ANNs are a versatile tool [50] and have recently been applied to solve reduced-order modeling problems across multiple disciplines using a nonintrusive framework [51–55]. The use of ANNs in GW astronomy is increasing [48,56–73]. In particular, the authors of [74] used ANNs to model the greedy reduced basis coefficients for a frequency domain inspiral post-Newtonian waveforms in the context of massive binary black holes (BBHs) that the space-based GW observatory LISA [75] will be

sensitive to. Here, we look at the projection coefficients of an empirical interpolation basis for time-domain waveforms. We generate the complete inspiral, merger, and ringdown waveform for the dominant ($\ell = |m| = 2$) multipole of spin-aligned BBH coalescences using the SEOBNRv4 model [76].

One advantage of our approach is that our ANN powered surrogate model (ANN-Sur) can be executed on either a CPU or GPU because it is developed with TensorFlow and allows us to explore the possible benefits of utilizing GPUs. We find that by generating waveform on a GPU, we gain a significant improvement in computationally efficiency. On average, waveforms generated with ANN-Sur take 1.2 ms on a CPU, which corresponds to a speed-up factor of 1495 when compared to the SEOBNRv4 and a factor of 70 when compared to SEOBNRv4opt. Moving waveform generation to a GPU provides a further factor of 2.5 improvement taking just 0.5 ms, which corresponds to a speed-up of 3588 when compared to SEOBNRv4 167 when compared to SEOBNRv4opt. These improvements can be readily passed on to standard parameter estimation codes.

Our model can also generate large batches of waveforms simultaneously [77]. We find that batches of waveforms up to sizes of 10^3 take only 1.57 ms on the GPU, a factor of 216 times faster than the CPU. We estimate that the time taken to generate the same waveforms using the SEOBNRv4opt model, on a single CPU, would take $\mathcal{O}(1)$ min, corresponding to a speed-up factor of approximately 38,000. These results are encouraging and suggest a way to drastically reduce waveform generation times using GPUs.

II. METHOD

Let $h(t) = h_+(t) - ih_\times(t)$ be the predicted complex gravitational-wave strain from a fiducial model, where t is the time. We expand this in terms of a spin-weight -2 spherical harmonic basis, which allows us to separate out the intrinsic parameters $\boldsymbol{\lambda}$ (black hole component masses and spin angular momenta) from extrinsic parameters (θ, φ) (direction of propagation)

$$h(t;\boldsymbol{\lambda};\theta,\varphi) = \sum_{\ell\geq 2}\sum_{-\ell\leq m\leq\ell} h_{\ell,m}(t,\boldsymbol{\lambda})\,{}_{-2}Y_{\ell,m}(\theta,\varphi). \quad (1)$$

If we restrict ourselves to noneccentric binary black hole systems with spins either aligned or antialigned with respect to the orbital angular momentum, then the system is completely specified by its mass ratio $q = m_1/m_2$ (m_1 and m_2 are the primary and secondary masses, respectively) and the components of the individual BH spin vectors that are aligned with the orbital angular momentum (χ_1, χ_2). Furthermore, we will model the $(\ell, m) = (2, \pm 2)$ multipoles which are the dominant multipoles for comparable mass BBH systems. The method we use is agnostic to the specific GW multipole and can therefore be applied to the other multipoles in a similar way; however, here we are interested in developing our method and restrict ourselves to just the dominant multipoles. For a final simplification, we note that for aligned-spin binaries the (2, 2) and $(2, -2)$ multipoles are related to each other according to $h_{2,2}(t) = h^*_{2,-2}(t)$, where $*$ denotes the complex conjugation. Therefore, we will only model the $h_{2,2}(t;\boldsymbol{\lambda})$ data where $\boldsymbol{\lambda} = (q, \chi_1, \chi_2)$. Instead of modeling the real and imaginary parts of $h_{2,2}(t;\boldsymbol{\lambda})$ as done in Ref. [74], we decompose the data into an amplitude, $A(t;\boldsymbol{\lambda}) \equiv |h_{2,2}(t;\boldsymbol{\lambda})|$, and phase, $\phi(t;\boldsymbol{\lambda}) \equiv \arg(h_{2,2}(t;\boldsymbol{\lambda}))$, and model these independently [41]. The original complex data are recovered with

$$h_{2,2}(t;\boldsymbol{\lambda}) = A(t;\boldsymbol{\lambda})e^{-i\phi(t;\boldsymbol{\lambda})}. \quad (2)$$

We use the surrogate modeling methods described in Refs. [41,78], borrowing notation and only recounting the basic steps here. We aim to build a surrogate model of the GW signal, denoted $h^S(t;\boldsymbol{\lambda})$, that emulates the fiducial model such that $h^S(t;\boldsymbol{\lambda}) \approx h(t;\boldsymbol{\lambda})$ to within a given error tolerance. The surrogate model is defined for times $t \in [t_{\rm min}, t_{\rm max}]$ and for system parameters $\boldsymbol{\lambda} \in \mathcal{T}$, where $\mathcal{T}$ is the compact parameter space of all possible BBH parameters. We therefore aim to build computationally efficient and accurate representations of the amplitude and phase functions $A^S(t;\boldsymbol{\lambda})$ and $\phi^S(t;\boldsymbol{\lambda})$, respectively, with the final surrogate $h^S_{2,2}$ given by

$$h^S_{2,2}(t;\boldsymbol{\lambda}) = A^s(t;\boldsymbol{\lambda})e^{-i\phi^S(t;\boldsymbol{\lambda})}. \quad (3)$$

In the following discussion, we will use $X(t;\boldsymbol{\lambda})$ as a placeholder variable to describe either the amplitude or the phase and $X^S(t;\boldsymbol{\lambda})$ as the surrogate approximation.

We can build an efficient representation of a function $X(t;\boldsymbol{\lambda})$ by building a reduced basis. A reduced basis is a linear decomposition such that for any value $\boldsymbol{\lambda} \in \mathcal{T}$ we can approximate $X(t;\boldsymbol{\lambda})$ as a linear combination of projection coefficients $\{c_i(\boldsymbol{\lambda})\}_{i=1}^n$ and the n-element basis $B_n = \{e_i(t)\}_{i=1}^n$ given by

$$X(t;\boldsymbol{\lambda}) \approx \sum_{i=1}^{n} c_i(\boldsymbol{\lambda})e_i(t). \quad (4)$$

We define the representation error between the true function and our reduced basis approximation as σ,

$$\sigma = \left\| X(t;\boldsymbol{\lambda}) - \sum_{i=1}^{n} c_i(\boldsymbol{\lambda})e_i(t) \right\|^2, \quad (5)$$

where $\|\cdot\|$ is the L_2 norm. To find the reduced basis representation, we use a greedy algorithm implemented in the RomPy PYTHON package [41,79]. We begin by densely sampling the parameter space and thus creating our training set $\mathcal{T}_{\rm TS}$; we then pick one of the points randomly to seed the greedy algorithm. This seed point is the first *greedy*

point and the first element in the basis B. The greedy algorithm iteratively builds up the basis by computing the current representation error σ against all points in $\mathcal{T}_{\rm TS}$. The sample with the largest representation error is added to the set of greedy points and also added to the basis using the iterative-modified Gram-Schmidt algorithm [80]. The greedy algorithm stops when the sample with the largest representation error is already in the basis or if the largest representation error is below the user-specified tolerance $\sigma_{\rm tol}$. This results in a set of m greedy points and a basis B of size m that covers $\mathcal{T}_{\rm TS}$ to within an accuracy of $\sigma_{\rm tol}$. If the $\mathcal{T}_{\rm TS}$ is sufficiently dense and thus representative of the entire $\mathcal{T}$, then we can use the reduced basis to approximate the function for any point in $\mathcal{T}$.

After we have built a reduced basis, we use the empirical interpolation method (EIM) [81,82] to construct an empirical interpolant of $X(t;\boldsymbol{\lambda})$. This results in a new basis, $\bar{B}_n = \{e_i(t)\}_{i=1}^n$, also of size m, that is constructed such that the coefficients of the basis $\{\alpha_j(\boldsymbol{\lambda})\}_{j=1}^n$ are values of the function X themselves at the empirical time nodes T_j,

$$\alpha_j(\boldsymbol{\lambda}) = X(T_j;\boldsymbol{\lambda}). \tag{6}$$

To evaluate the surrogate model at any point in $\mathcal{T}$, one can approximate the α coefficients by either fitting or interpolating them across $\mathcal{T}$. We denote the fitted coefficients as $\hat{\alpha}$. We use the EIM because typically the variation of the α coefficients is smoother than the c reduced basis coefficients. This makes it easier to fit or interpolate the coefficients and requires a smaller training set to obtain a model of the coefficients.

Up to this point, we only know the α coefficients at the greedy points. This is typically not enough points to sample the α functions to accurately fit or interpolate across the parameter space. In this paper, we build a surrogate model of SEOBNRv4, which permits us to generate large training sets that we can use to sample the $\alpha_j(\boldsymbol{\lambda})$ functions.

Finally, the surrogate model for $X(t;\boldsymbol{\lambda})$ is defined as

$$X^S(t;\boldsymbol{\lambda}) \approx \sum_{i=1}^{n} \hat{\alpha}_i(\boldsymbol{\lambda})\bar{B}_i(t). \tag{7}$$

The problem therefore is reduced to finding a fast and accurate approximation to α. There are many machine learning methods that can be used to interpolate or fit these coefficients, and depending on the accuracy required and the dimensionality, different methods will be more suitable than others. In Ref. [43], the authors interpolate the reduced basis coefficients directly in the three-dimensional (3D) aligned-spin parameter space. Interpolation is a good method for low-dimensional parameter spaces, but in dimensions greater than or approximately equal to 3, interpolation becomes difficult due to the large number of data points typically required. In Refs. [9,78], the authors built a surrogate model for numerical relativity produced precessing BBHs corresponding to a seven-dimensional parameter space with the EIM. Here, because of the relatively small size of their training set and the high-dimensional parameter space, interpolation was not appropriate and instead used a basis of monomials constructed with a greedy algorithm to reduce overfitting. In Ref. [9], the authors modeled the 3D aligned-spin parameter space with numerical relativity simulations using EIM and fit the coefficients using Gaussian process regression.

In Ref. [48], the authors systematically explored several methods and ranked them in terms of accuracy, time to fit, and prediction time. They also experimented with ANNs but restricted to shallow networks with only two hidden layers and training/execution on CPUs only. In Ref. [74], the authors modeled the reduced basis coefficients of post-Newtonian inspiral waveforms using a four-dimensional parameter space, comprising the component masses and the aligned-spin components, using ANNs. In this work, we use a similar approach but instead applied to the empirical interpolation (EI) α coefficients and model the complete inspiral, merger, and ringdown signal.

III. BINARY BLACK HOLE SURROGATE MODEL

A. Parameter space

In this paper, we investigate the possibility to use ANNs in the construction of surrogate waveform models for BBH signals. We build a surrogate model of the model SEOBNRv4 [76].[1] It predicts the GW signal emitted from noneccentric BBH mergers where the black hole spin angular momenta are constrained to be parallel (or antiparallel) with the orbital angular momentum. Extensions of this model to include subdominant multipoles and precession have been done [5,83]; however, we develop our method with the simpler case. This model is based on the effective-one-body (EOB) formalism, extended to predict the merger and ringdown signal by fitting free coefficients to numerical relativity solutions. This is a time-domain model where the inspiral model is calculated by solving the EOB Hamiltonian equations of motion: a set of coupled, ordinary differential equations. This method has proven to provide accurate GW templates, but typical implementations of EOB models tend to be computationally expensive. As such, a lot of work has gone into optimizing the production of EOB templates, either by improving the computational efficiency of the inspiral calculation [23–25] or by developing the frequency domain, reduced-order surrogate model [43]. While there already exists frequency domain surrogate models for both SEOBNRv4 [76] and SEOBNRv4HM [84], it is an excellent model to develop new methodology, and we apply this to the time domain rather than the frequency domain.

[1]Waveforms were generated with version 2.1.0 of the LALSimulation library.

Motivated by past work [9] and to facilitate comparisons, we build a surrogate model of SEOBNRv4 covering mass ratios from 1:1 to 1:8 and allowing each black hole spin to range from −0.99 to 0.99. For each system, we generate the $h_{22}(t) \in \mathbb{C}$ multipole data. This method to construct the reduced basis requires that all data are evaluated on the same time grid. We choose to build a surrogate model that is valid from 15 Hz at a total mass of 60 $M_\odot$ for all mass ratios and spins in the training set. To find the start frequency of the surrogate, $f_{\rm start}$, for a new total mass, $M_{\rm new}$, we can use the formula $f_{\rm start}(M_{\rm new}) = 15 \times (60\ M_\odot/M_{\rm new})$ Hz. We work with geometric units M and perform a time shift such that $t = 0M$ corresponds to the peak of the amplitude. It is important that this procedure is done with high accuracy to avoid an unnecessarily large reduced basis [41].

To ensure that the surrogate is valid for the domain stated above, we generate all waveforms with a lower start frequency of 8 Hz and then truncate all data such that the data start at atleast 15 Hz. For the parameter space we consider, this corresponds to a start time of $-20{,}000\,M$. In addition to performing a time shift to the data, we also perform a phase shift such that the phase is zero at the start time (i.e., $-20{,}000\,M$). We keep $100\,M$ of postpeak ringdown data. Finally, the data are resampled at a resolution of $\Delta t = 0.5\,M$ onto the domain $[t_{\rm min}, t_{\rm max}] = [-20{,}000, 100]M$. When analyzing long signals, then the physical constraints of computer memory become an issue. There are a number of ways to compress the training data which typically involve nonuniformly sampled data [26,85–87]; however, these were unnecessary here.

We generate three different sets of data using uniform random sampling: training, validation, and test sets. The training set is used to (1) build the reduced basis and (2) densely sample the projection coefficients that we will fit. The validation set is also used to sample the projection coefficients but is only used to monitor the accuracy of the fit to diagnose if the model is under- or overfitting and to help tune the hyperparameters of the network. The test set, or hold-out set, is not used in the training of the network but is used to evaluate the final accuracy of the model. The validation and test sets serves as a way to assess the size "generalization gap" of the model, the distance between the performance of the model on the training set and on the hold-out set. The training set contains 2×10^5 samples, and the validation and test sets both contain 2×10^4 samples.

B. Waveform performance metrics

To quantify the level of agreement between two, real-valued, waveforms h_1 and h_2, we use the standard inner product weighted by the noise power spectral density (PSD) of the GW detector $S_n(f)$. It is defined as [88]

$$\langle h_1, h_2 \rangle = 4\mathrm{Re} \int_{f_{\rm min}}^{f_{\rm max}} \frac{\tilde{h}_1(f)\tilde{h}_2^*(f)}{S_n(f)} df. \tag{8}$$

The match between two waveforms is defined as the inner product between normalized waveforms ($\hat{h} \equiv h/\sqrt{\langle h, h\rangle}$) maximized over a relative time (t_0) and phase (ϕ_0) shift between the two waveforms,

$$M(h_1, h_2) = \max_{t_0, \phi_0} \langle \hat{h}_1, \hat{h}_2 \rangle. \tag{9}$$

Finally, we shall quote results in terms of the mismatch, which is the fractional loss in the signal-to-noise ratio due to modeling errors defined as

$$\mathcal{M}(h_1, h_2) = 1 - M(h_1, h_2). \tag{10}$$

C. Reduced basis construction

We choose to monitor the relative greedy error, that is, the error relative to the representation error at the first iteration. To determine what value to use for the tolerance, we varied the tolerance from 10^{-6} to 10^{-16} logarithmically in steps of 2. For each resulting basis, we computed the mismatch [Eq. (10)] between the training data and the basis representation. We also did this for the validation set, and Table I shows the results. We find that the number of basis functions grows much faster for the amplitude than for the phase.

We base our choice of greedy tolerance, and therefore of the number of basis functions to use, on the accuracy of the SEOBNR4 model. In Ref. [76], the accuracy in terms of the mismatch was found to be between 10^{-2}–10^{-4} when

TABLE I. Worst mismatch of the reduced basis and reduced basis size (for amplitude and phase bases) as a function of greedy error tolerance.

Greedy tolerance $\sigma_{\rm tol}$	Training set	Validation set	Number of bases: amplitude	Number of bases: phase
10^{-6}	3.1×10^{-1}	3.1×10^{-1}	9	3
10^{-8}	1.9×10^{-2}	1.8×10^{-2}	13	5
10^{-10}	6.5×10^{-5}	6.2×10^{-5}	19	8
10^{-12}	1.2×10^{-6}	1.1×10^{-6}	39	12
10^{-14}	1.1×10^{-8}	8.2×10^{-9}	91	33
10^{-16}	9.7×10^{-10}	9.7×10^{-10}	102	51

compared to numerical relativity data. Therefore, we use a greedy tolerance of 10^{-10}, which produces a basis with mismatch errors of at worst approximately 6.5×10^{-5} for both the training and validation set. The consistency between the training and validation set implies that we have sampled the space with the training set densely enough that the basis can represent out of sample waveforms with equivalent accuracy. This produces a reduced basis with only 19 basis functions for the amplitude and 8 basis functions for the phase.

IV. NEURAL NETWORK TRAINING STRATEGY

In this section, we investigate how different choices of data preprocessing, neural network architecture, optimizers, and minibatch size impact the networks ability to fit (or learn) the data. We will call the combined set of choices our *training strategy*, and our goal is to find the optimal training strategy to minimize the loss function over different training strategies. We only outline our investigation here and leave details to Appendix.

In general, is it not trivial to know how a particular change to any of these parameters will effect the network or indeed if the choices are independent of each other. To make this problem tractable, we will use a greedy method, making locally optimal choices at each step. We explore each aspect of the training strategy in the following order: (i) data preprocessing, (ii) width and depth of the neural network, (iii) activation functions, and finally (iv) optimizers. At each step, we perform the experiment twice, once using batched gradient decent (using the entire dataset) and again using minibatch gradient decent with a minibatch size of 1000 [89]. At each step, we typically will take the neural network which has the smallest final loss and use those parameters in the next step; however, in some tests, we find there are several network configurations that perform equally well. For those cases, we used the settings that resulted in the fastest trained network. We note that if the ordering of exploration were different then it would be possible that we would end up with a different training strategy.

The independent variables of the data we will fit are the mass ratio (q), the aligned-spin component of the primary (χ_1), and the aligned-spin component of the secondary (χ_2). As done in previous surrogate models [9,90], we first perform a logarithmic transformation on the mass ratio, as we also find that this helps fit the data more accurately. In the following sections, we will refer to the independent variables, i.e., $\log(q)$, χ_1, and χ_2, simply as $\mathbf{X}$ and the dependent variables, i.e., the coefficients of the empirical interpolation basis, as $\mathbf{Y}$. For the amplitude, $\mathbf{Y}$ is a 19-dimensional vector, and for the phase, it is an 8-dimensional vector (see Table I).

We use TensorFlow [49] and KERAS [91] to design and train two independent feed-forward, fully connected neural networks, one for the amplitude and one for the phase, using the mean-squared error loss function. The input layer is given by the dimensionality of the independent variables ($\mathbf{X}$). The rest of the network, the number of hidden layers, number of neurons in each layer, and choice of activation function, will be explored. The output layer uses a linear activation function (suitable for regression problems), and the number of output neurons is given by dimensionality of the dependent variables ($\mathbf{Y}$). We train the networks using the backpropagation algorithm to minimize the loss function with respect to the network's weights and biases.

One of the key decisions to make is how one should choose the learning rate for the stochastic gradient decent algorithm. Some authors suggest that the choice of minibatch size should be linked with the choice of learning rate [92–94]. We explore a range of different optimizers in Appendix but always use a learning rate that decreases with time according to

$$\tau_k = (\tau_{\text{init}} - \tau_{\text{final}})/(1 + R\lfloor k/\Delta k \rfloor) + \tau_{\text{final}}, \quad (11)$$

where τ_k is the learning rate at epoch k, τ_{init} is the initial learning rate (10^{-3}), τ_{final} is the final learning rate (10^{-5}), R is the decay rate (10), and Δk is the interval between decaying (2000); unless otherwise stated, the values we use are given in parentheses. We choose to compute the floor of the ratio $k/\Delta k$, which means the learning rate exhibits stepwise changes. Some optimizers, such as Adam [95], already use an adaptive learning rate; however, by using a learning rate scheduler, we can further control the maximum value of the learning rate as a function of time (epoch).

A. Final neural network model

The final training strategies for the amplitude and phase data are given in Table II. The networks were for trained for 10^5 epochs with a minibatch size of 1000, which took approximately 6–7 hours on a Tesla P100 GPU.

We found that the data preprocessing method had a large impact on the performance of the networks; see Appendix for details. For the amplitude data, the optimal preprocessing

TABLE II. Final training strategy for amplitude and phase data. Data preprocessing, neural network architecture, and hyperparameter choices for amplitude and phase data. A minibatch size of 1000 was used for both. GPU used: Tesla P100.

	Amplitude	Phase
$\mathbf{X}$ preprocessing	Standard scaler	Standard scaler
$\mathbf{Y}$ preprocessing	None	Min-max scaler
N hidden layers	4	4
Units per layer	320	320
Activation function	ReLU	Softplus
Optimizer	Adam	AdaMax
Final loss	4.93×10^{-7}	1.82×10^{-9}
Final validation loss	5.74×10^{-7}	1.90×10^{-9}
Training time	6–7 hr	6–7 hr

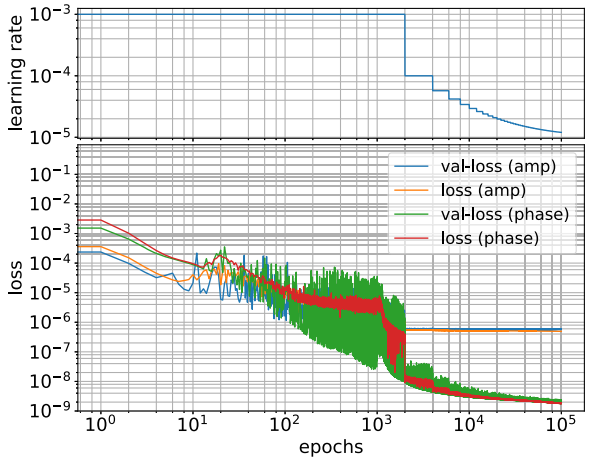

FIG. 1. Top panel: learning rate. Middle panel: the amplitude loss (orange) and validation loss (blue) curves as a function of epochs. Bottom panel: the phase loss (red) and validation loss (green) curves as a function of epochs.

methods are to normalize the **X** data and use the raw **Y** data. For the phase, we normalize the **X** data and scale the **Y** data. For both the amplitude and phase networks, we use four hidden layers, each with a width of 320 units per layer. As detailed in Appendix, we find that deeper networks can achieve lower losses, but not by a significant amount. For the hidden layer activation functions, we find that the rectified linear unit (ReLU) function performed best for the amplitude data and the Softplus function performed best for the phase data. Finally, we used the Adam optimizer for the amplitude data and the AdaMax optimizer for the phase data.

In Fig. 1, we show the loss and validation-loss learning curves for the amplitude and phase data on a log-log scale. The top panel shows the learning rate as a function of epoch, which decreases according to Eq. (11), every 2000 epochs. The sudden drops in the loss curves correspond to the drops in the learning rate.

We find that the amplitude data show some very mild signs of overfitting and the phase data show signs of underfitting; however, as we will see in the next section, these networks produce mismatch errors below our error tolerance.

V. MODEL EVALUATION

With the final neural network models for the EI amplitude and phase coefficients in hand, we can evaluate the performance of ANN-Sur we have built to mimic SEOBNRv4. We scrutinize the surrogate model using a two different tests. The first test (Sec. V A) is to see how accurate the surrogate model is when compared to the original model. The second test (Sec. V B) is to quantify what is the speed improvement we have achieved compared with SEOBNRv4. We also compare to other state-of-the-art models in terms of computational efficiency and the improvement obtained when running the model on a GPU rather than a CPU.

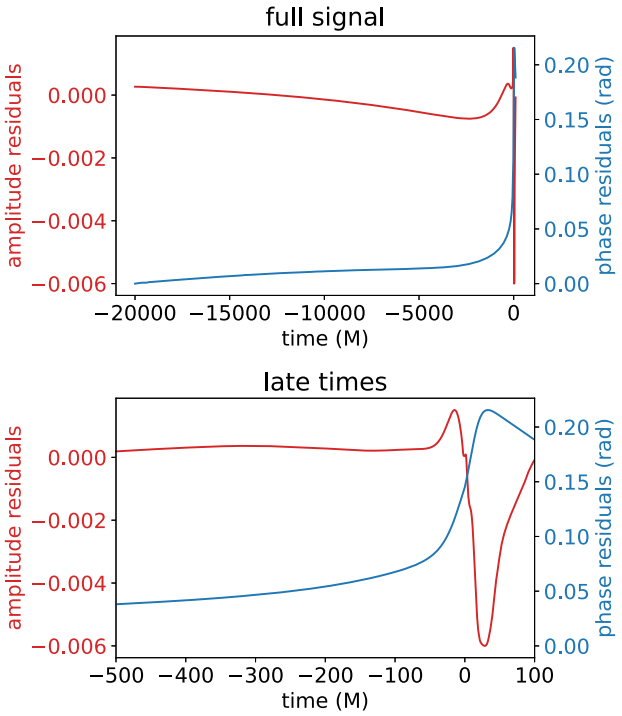

FIG. 2. Amplitude and phase residuals for the case $q = 4.54$, $\chi_1 = 0.99$, $\chi_2 = 0.90$. The top panel shows the complete signal, and the bottom panel shows a zoom-in around the merger (at $t = 0$ M). The left y axis corresponds to the amplitude residuals (red curve), and the right y axis shows the phase residuals (blue curve).

For a first cross-check of the ANN-Sur model, we computed the root mean squared error (RMSE) for the amplitude and phase for all the cases in the validation dataset. The worst RMSE occurred at parameters $q = 4.54$, $\chi_1 = 0.99, \chi_2 = 0.90$ giving a RMSE of 4.4×10^{-4} for the amplitude and 2.1×10^{-2} rad for the phase.

In Fig. 2, we show the residuals for this case. The top panel shows the complete signal, and the bottom panel shows a zoom-in around the merger (at $t = 0$ M). The left y axis corresponds to the amplitude residuals (red curve), and the right y axis shows the phase residuals (blue curve). We find that the majority of the signal has a small, slowly varying error. The main source of error comes from the region of time around the merger. The peak errors in amplitude and phase are approximately 0.006 and approximately 0.2 rad, respectively.

A. Mismatch vs total mass

To quantify the accuracy of the surrogate model, we compute the mismatch, using the expected noise curve for Advanced LIGO operating at design sensitivity [96], between $\mathrm{Re}(h_{2,2})$ generated by ANN-Sur and all the waveforms in the validation dataset, noting that results are similar for the training and test datasets. Because of the shape of the PSD, the smaller (larger) values of M_{tot} tend to accentuate

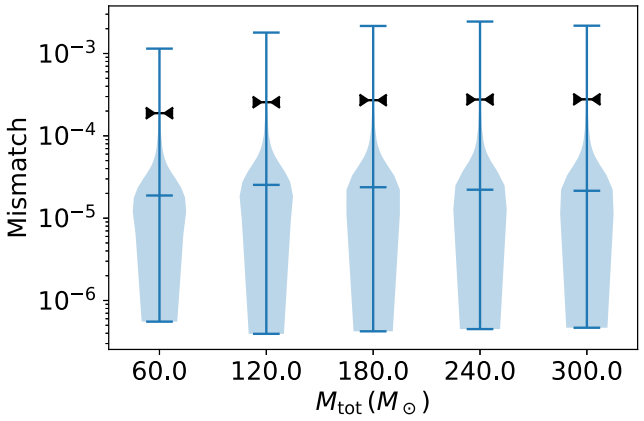

FIG. 3. Mismatches between the ANN-Sur and the SEOBNRv4 validation dataset represented as a violin plot. The median is marked by the middle horizontal line, and the extent of the lines shows the minimum and maximum values. The envelope is proportional to the density of points. The black triangles mark the 95th percentile. We remind the reader that the accuracy of the SEOBNRv4 model is between 10^{-2} and 10^{-4} [76].

modeling errors during the inspiral (merger); therefore, we consider the values $M_{\rm tot} = (60, 120, 180, 240, 300)\ M_\odot$. We used a low-frequency cutoff of 15 Hz and variable high-frequency cutoff given by $1.4 f_{\rm RD}$ Hz, where $f_{\rm RD}$ is an estimate of the final black hole ringdown frequency [97], the results of which are shown in Fig. 3. We find that the mismatch is stable as a function of $M_{\rm tot}$ with a slight rise in the mismatch by 1e−3 for larger values of $M_{\rm tot}$. The vast majority of cases have mismatches below approximately 3×10^{-4} (95th percentile) with a median value of approximately 2×10^{-5}. The lowest mismatch we achieve is approximately 4×10^{-6}. The highest mismatch obtained is approximately 2×10^{-3}, and these cases are distributed primarily in two clusters as shown in Fig. 4. One cluster is toward the upper boundary of χ_1. The other cluster is toward the corner of low χ_1 and low q. If more training points in these regions do not improve performance here, then a domain decomposition strategy can be employed.

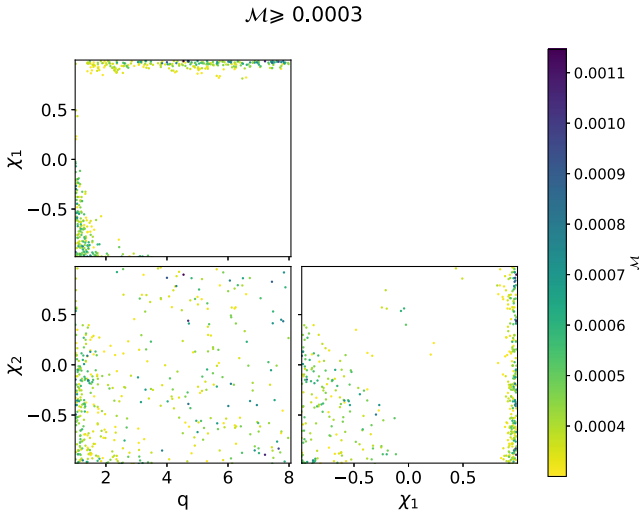

FIG. 4. Mismatches plotted across the (q, χ_1, χ_2) parameter space. Only cases with mismatches larger than the 95th percentile (3e − 4) are shown. This is the result for $M_{\rm tot} = 60\ M_\odot$, but other $M_{\rm tot}$ are similar.

B. Computational speed

Most waveform models are designed to run on CPUs with some recent work on moving waveform generation onto a GPU [37,38,73,77]; however, it is still an open question of how waveform generation can make the most use of GPUs. With TensorFlow, we can generate optimized TensorFlow graphs with accelerated linear algebra [98] compilation that can be executed on either a CPU or GPU. Note that these results will vary depending on the hardware used. Here, we used an Intel 2.40 GHz Xeon CPU E5-2630v3 and a Tesla V100 GPU for our comparisons.

In Table III, we quantify the speed-up we achieve compared with the original SEOBNRv4 model as well as the optimized version of the model SEOBNRv4opt. We generated the GW signal with the parameters $q = 3$, $M_{\rm tot} = 60\ M_\odot$, $\chi_1 = 0.8$, and $\chi_2 = 0.5$ and use a sample rate of $1/2048$ s and an initial frequency of $f_{\rm min} = 12$ Hz (corresponding to a length of approximately 20,000 M). When generating the GW signal with the ANN-Sur model, we use a linear interpolation algorithm, available in the TensorFlow Probability library, to resample the amplitude and phase onto the same time grid as the EOB waveform as well as compute the h_+ polarization.

We find that the SEOBNRv4 model takes 1794 ms to compute this waveform with the SEOBNRv4opt model improving upon this by a factor of approximately 20 to 83.7 ms. The ANN-Sur model on a CPU takes 1.2 ms, giving

TABLE III. Average time (ms) to generate a one waveform averaged over 100 waveforms. Times and speed-ups in parentheses correspond to the SEOBNRv4opt model. $q = 3$, $M_{\rm tot} = 60\ M_\odot$, $\chi_1 = 0.8$, $\chi_2 = 0.5$. $f_{\rm min} = 12$ Hz (corresponding to a length of approximately 20,000 M). For time-domain approximants, we used a sample rate of $1/2048$ s. Models prefixed with a * are frequency-domain models, and we used a sample rate of $1/8$ Hz. When evaluating ANN-Sur, SEOBNRv4ROM, and NRHybSur3dq8, "warm-up" execution is performed to load one-time overhead data. Additionally, for NRHybSur3dq8, we only evaluate the (2, 2) mode.

Model	Time (ms)	Speed-up
SEOBNRv4 (opt)	1794 (83.7)	···
ANN-Sur CPU	1.2	1495 (70)
ANN-Sur GPU	0.5	3588 (167)
*SEOBNRv4ROM	3.5	···
*IMRPhenomD	1.4	···
*IMRPhenomXAS	1.4	···
NRHybSur3dq8	27.3	···

a speed up of 1495 (70 with respect to SEOBNRv4opt). When running the ANN-Sur model on a GPU waveform, generation takes just 0.5 ms, giving a speed-up of 3588 (167 with respect to SEOBNRv4opt).

We generated the same GW signal with other state-of-the-art GW signal models for the dominant (2,2) harmonic for nonprecessing binaries. SEOBNRv4ROM is also a surrogate model for SEOBNRv4; however, it is constructed in the frequency domain and interpolates reduced basis projection coefficients. NRHybSur3dq8 [9] is a time-domain surrogate model for numerical relativity simulations produced with the SpEC code and hybridized with Post-Newtonian/EOB inspiral waveforms. It also uses EIM but models the α projection coefficients using Gaussian process regression. IMRPhenomD [99] and its successor IMRPhenomXAS [16] are frequency domain phenomenological models. Phenomenological models combine results from post-Newtonian theory, black hole perturbation theory, and numerical relativity solutions together with sophisticated modeling techniques to build custom-made models for the GW signal. We note that comparing to other surrogate models should be done with caution. The computational speed-up of a surrogate model comes from (i) the size of the basis and (ii) the efficiency of the method used to estimate the projection coefficients. Both of these are affected by the parameter space (including the duration of the signal) that the surrogate hopes to cover. Therefore, for SEOBNRv4ROM and NRHybSur3dq8, that cover longer-duration signals, the comparisons relate to their specific implementation and not necessarily to the optimal performance of the method used to predict the basis coefficients.

We find that NRHybSur3dq8 takes the longest to generate this waveform taking 27.3 ms. Next is SEOBNRv4ROM taking 3.5 ms. Finally, the fastest models are the IMRPhenomD and IMRPhenomXAS models taking approximately 1.4 ms. ANN-Sur is highly competitive in terms of computational speed, outperforming all but the IMRPhenom models when run on a CPU and outperforming all models when run on a GPU by a factor of approximately 3.

Some calculations can be rapidly accelerated by using a GPU by processing similar calculations in parallel using *batches*. ANN-Sur is built with TensorFlow and can readily take advantage of this. In Table IV, we time how long ANN-Sur takes to generate random batches of (10, 100, 1000) waveforms, averaged over 100 trials, both on a CPU and a GPU. We find that, even on a single CPU, the batched calculation can produce 1e3 waveforms in approximately 0.4 s and the use of a GPU provides a speed-up factor of 216 taking only 1.57 ms. To generate the same number of SEOBNRv4opt waveforms on a single CPU, we estimate it would take approximately 1 min. Therefore, ANN-Sur produces a speed-up factor of approximately 38,000.

The ability to extremely efficiently produce large numbers of template waveforms simultaneously on a single CPU or GPU has the potential to substantially reduce the computational cost of GW analyses such as parameter estimation [100,101] and in the generation of GW template banks [102,103].

VI. CONCLUSION

In the next five years, the size of GW catalogs is expected to grow from $\mathcal{O}(10)$ to $\mathcal{O}(10^3)$ [104,105]. It is therefore imperative that we devise methods that can make use of the most accurate waveform models, which are typically also the most computationally expensive, in the analysis of all GW events.

In this paper, we have presented ANN-Sur, our methodology to construct surrogates for GW signal models powered by artificial neural networks. A similar idea was presented in Ref. [74] with a focus on inspiral-only signal models and masses suitable for the LISA detector. Here, we focus on GW signals for the complete inspiral, merger, and ringdown with a mass range targeted for current ground-based detectors. In a first application of our method, we have built a time-domain surrogate model of the SEOBNRv4 model for spin-aligned binary black hole mergers, which covers the following 3D intrinsic parameter space: $q \in [1, 8]$, $\chi_{1,2} \in [-0.99, 0.99]$. We built the surrogate to be valid from 15 Hz for a total mass of 60 $M_\odot$, which leads to a length of approximately 20,000 M. When compared with the original SEOBNRv4 model, our surrogate model has a worst mismatch of approximately 2e−3 and a median mismatch of approximately 2e−5; see Fig. 3.

ANN-Sur is built with the TensorFlow library and can seamlessly run on either a CPU or GPU. In Sec. V B, we compared the computational efficiency of ANN-Sur with the original SEOBNRv4 model. We find that the average time to compute a single waveform with the optimized SEOBNRv4 model is 83.7 ms; when running ANN-Sur on

TABLE IV. Computational efficiency of ANN-Sur when generating batches of waveforms.

	CPU		GPU		Speed-up
	Total time (ms)	Time per waveform (ms)	Total time (ms)	Time per waveform (ms)	(CPU/GPU)
Single	1.2	1.2	0.5	0.5	2.4
Batched (10)	6.7	0.67	0.6	0.06	11
Batched (10^2)	48	0.48	0.57	0.0057	84
Batched (10^3)	339	0.34	1.57	0.0016	216

a CPU, this is reduced to 1.2 ms, and when run on a GPU, this takes just 0.5 ms, a factor of 167 improvement. When comparing with the frequency-domain surrogate model SEOBNRv4ROM, we find that ANN-Sur is a factor of 3 (7) times faster when run on a CPU (GPU). We expect that frequency-domain surrogate models built using this method would be significantly improved, which may further increase the performance of likelihood acceleration techniques such as the reduced-order quadrature rule [26–28].

ANN-Sur also permits us to generate large numbers of waveforms simultaneously in batches on a single CPU or GPU. In Table IV, we find that we can generate batches of up to 10^3 waveforms in approximately 340 ms on a CPU and in just approximately 1.57 ms on a GPU, corresponding to a per waveform generation time of just 0.0016 ms. This new kind of parallelization allows for the generation of large training sets to train deep learning methods to perform Bayesian inference [63,77,106] or to rapidly generate waveforms for grid-based methods such as [39,100,101]. The increased computational efficiency gained here should also be obtained for binary neutron star systems [6,14] and neutron star black hole binaries [107,108], increasing the likelihood that we will find multimessenger events [109].

While our surrogate meets current accuracy requirements, with only 19 and 8 basis functions for the amplitude and phase, respectively, higher-accuracy surrogate models will be required in the future as detectors become more sensitive. Higher-accuracy surrogates can be built by including more basis functions (for example, see Table I); however, we found that the ANNs we used were unable to model the projection coefficients accurately enough. This issue should be solved by using larger training sets and improving our training strategy.

One of the next steps will be to incorporate the full BBH parameter space, i.e., build a surrogate model that includes spin-precession and higher harmonics [3,11,19,83,110]. Extending our method to work effectively in higher dimensions is also possible by increasing the size of the training set and network capacity.

A final and unique advantage of our method is to be able compute waveform derivatives using automatic differentiation [111]. This is a key ingredient for the Bayesian inference sampling method Hamiltonian Monte Carlo [112–114]. This has rarely been used in the GW astronomy community [115,116] as the computational cost of computing the required likelihood derivatives quickly offsets any performance gained from using Hamiltonian Monte Carlo. We are currently exploring the benefits of combining Hamiltonian Monte Carlo with ANN-Sur, which will be presented in the future.

ACKNOWLEDGMENTS

We thank Alvin Chua, Edward Fauchon-Jones, Vasileios Skliris, Michael Norman, Luke Berry, David Sullivan, Vivien Raymond, and Mark Hannam for useful discussions. S. K. was supported by European Research Council Consolidator Grant No. 647839. R. G. was supported by Science and Technology Facilities Council (STFC) Grant No. ST/L000962/1. This work was performed using the Cambridge Service for Data Driven Discovery (CSD3), part of which is operated by the University of Cambridge Research Computing on behalf of the STFC DiRAC HPC Facility ([117]). The DiRAC component of CSD3 was funded by BEIS capital funding via STFC capital Grants No. ST/P002307/1 and No. ST/R002452/1 and STFC operations Grant No. ST/R00689X/1. DiRAC is part of the National e-Infrastructure. We acknowledge the support of the Supercomputing Wales project, which is partly funded by the European Regional Development Fund (ERDF) via the Welsh government. We are grateful for computational resources provided by the LIGO Laboratory, supported by National Science Foundation Grants No. PHY-0757058 and No. PHY-0823459 and by Cardiff University supported by STFC Grant No. ST/I006285/1.

APPENDIX: NEURAL NETWORK EXPLORATION

In this Appendix, we show additional material to justify our choice for the final networks. As mentioned in Sec. IV, we run each experiment twice, first using batched gradient decent and second using minibatch gradient decent with a minibatch size of 1000. We find that the minibatch results always outperform the batched results, and so we only present the minibatch results for most cases.

Data preprocessing.—Data preprocessing refers to actions we do to modify the $\mathbf{X}$ and $\mathbf{Y}$ data. We considered three different options: (i) do nothing, (ii) *normalize* the data such that it has zero mean and unit variance, or (iii) *scale* the data to lie between 0 and 1. To normalize and scale the data, we use the StandardScaler and MinMaxScaler functions in the SCIKIT-LEARN PYTHON package.

Before performing a more exhaustive search to find the optimal number of hidden layers and artificial neurons, we use an initial network to explore the effects of data preprocessing. This initial network, found through manual prototyping, makes use of several common choices in neural network design. It has six hidden layers with 256 neurons in each layer, and each neuron uses the ReLU activation function.

For each dimension or *feature* of $\mathbf{X}$ and $\mathbf{Y}$, we apply the three preprocessing methods and train a neural network for each pair of preprocessing methods. We also consider the effect of the minibatch size on the training by repeating each experiment twice, once with a batch size equal to the entire training set (2×10^5) and again with a minibatch size downsampled by 200, giving a minibatch size of 1000. We trained the networks for 10^3 epochs, which took approximately 20 mins for the batched gradient decent case and

approximately 40 mins for the minibatch case on a Tesla P100 GPU.

We find that the phase $\mathbf{Y}$ data are influenced the strongest by the choice of preprocessing and the $\mathbf{X}$ data preprocessing has a smaller impact, although is noticeable. For the amplitude data, we find that preprocessing can influence the results but not as strongly as the phase data. The reason for this is because the amplitude data are, for the most part, of the same order of magnitude and O(1). The phase, on the other hand, as it is accumulated as the binary system evolves, can span many orders of magnitude depending on the duration of the signal. Therefore, by applying a preprocessing step such as normalizing or scaling the data brings all the EI coefficients into a similar range, which can help make it easier to train a neural network. We believe that the preprocessing step applied to the $\mathbf{X}$ data is less important because, for our dataset, the data are between 0 and 1 for the $\log(q)$ and between -1 and 1 for the spin dimensions.

We find that, for the phase data, the optimal preprocessing methods are to normalize the $\mathbf{X}$ data and scale the $\mathbf{Y}$ data. For the amplitude, we will normalize the $\mathbf{X}$ data and use the raw $\mathbf{Y}$ data. We will use these as the optimal choices for preprocessing the data moving forward and investigate how the network architecture, choice of optimizer, and minibatch size can affect the training these neural networks.

Width and depth.—The number of possible configurations a feed-forward, fully connected artificial neural network could take presents a near limitless number of possible network architectures. While the number of neurons in each hidden layer does not have to be the same, we restrict ourselves to neural networks of a constant width (i.e., number of neurons in each hidden layer) but allow this number and the number of hidden layers (the depth) to vary. Following the parametrization in Ref. [118], we perform a systematic search for the optimal number of hidden layers (depth) L and number of neurons in each hidden layer (width) N. We form the ratio $\beta = L/N$ and consider values $\beta < 1$ which correspond to networks that are wider than their depth. We consider networks with a maximum number of hidden layers $L_{\max} = 10$ and three values of $\beta \in \{0.0125, 0.025, 0.05\}$.

We find that for the phase data $\beta = 0.0125$ predominantly perform best, followed by $\beta = 0.025$ and $\beta = 0.05$, respectively. A similar pattern is observed for the amplitude data. The amplitude data favor deeper networks with seven to ten layers, whereas the phase data prefer networks with three to nine layers.

For the phase, the top two networks both have $\beta = 0.0125$. The best network has $L = 5$ hidden layers and $N = 400$ units per layer, and the second best network has $L = 4$ hidden layers and $N = 320$ units per layer. As the difference in final loss is insignificant, we chose the network with four hidden layers as it was significantly faster to train.

For the amplitude data, the best performing networks were typically deeper and wider than the phase networks. However, these differences did not present a significant increase in accuracy, so we opted to use the same network chosen for the phase data as it also performed well for the amplitude data.

Activation function.—We found that the performance of ANNs on the phase data was strongly influenced by the choice of activation function but the amplitude data were fairly insensitive to this choice. For the amplitude, the best-performing activation functions were PReLU, ReLU, and the Leaky_ReLU. As the PReLU and the Leaky_ReLU add addition parameters to the training strategy, we decided to use the ReLU activation function for the amplitude. For the phase, we find that the PReLU, ReLU, and the Leaky_ReLU also perform well but the Softplus outperforms them both in accuracy and training time.

Optimizer.—We find that SGD, Adadelta, and Adagrad consistently underperform for both the amplitude and phase data, producing loss values approximately 3 orders of magnitude worse than the other optimizers tested. For the amplitude data, we find that the Adam optimizer performs equally as well as the Nadam optimizer and results in a network that is significantly faster to train. For the phase data, we find that AdaMax outperforms Adam, Nadam, and RMSprop.

[1] A. K. Mehta, C. K. Mishra, V. Varma, and P. Ajith, Phys. Rev. D **96**, 124010 (2017).
[2] A. K. Mehta, P. Tiwari, N. K. Johnson-McDaniel, C. K. Mishra, V. Varma, and P. Ajith, Phys. Rev. D **100**, 024032 (2019).
[3] S. Khan, F. Ohme, K. Chatziioannou, and M. Hannam, Phys. Rev. D **101**, 024056 (2020).
[4] H. Estell'es, A. Ramos-Buades, S. Husa, C. Garc'ia-Quir'os, M. Colleoni, L. Haegel, and R. Jaume, arXiv: 2004.08302.
[5] R. Cotesta, A. Buonanno, A. Bohé, A. Taracchini, I. Hinder, and S. Ossokine, Phys. Rev. D **98**, 084028 (2018).
[6] A. Nagar, S. Bernuzzi, W. del Pozzo, G. Riemenschneider, S. Akcay, G. Carullo, P. Fleig, S. Babak, K. W. Tsang, M. Colleoni, F. Messina, G. Pratten, D. Radice, P. Rettegno, M. Agathos, E. Fauchon-Jones, M. Hannam, S. Husa, T. Dietrich, P. Cerdá-Durán, J. A. Font, F. Pannarale, P. L. Schmidt, and T. Damour, Phys. Rev. D **98**, 104052 (2018).

[7] A. Nagar, G. Pratten, G. Riemenschneider, and R. Gamba, Phys. Rev. D **101**, 024041 (2020).
[8] N. E. M. Rifat, S. E. Field, G. Khanna, and V. Varma, Phys. Rev. D **101**, 081502 (2020).
[9] V. Varma, S. E. Field, M. A. Scheel, J. Blackman, L. E. Kidder, and H. P. Pfeiffer, Phys. Rev. D **99**, 064045 (2019).
[10] S. Khan, K. Chatziioannou, M. Hannam, and F. Ohme, Phys. Rev. D **100**, 024059 (2019).
[11] V. Varma, S. E. Field, M. A. Scheel, J. Blackman, D. Gerosa, L. C. Stein, L. E. Kidder, and H. P. Pfeiffer, Phys. Rev. Research **1**, 033015 (2019).
[12] D. Williams, I. S. Heng, J. R. Gair, J. A. Clark, and B. Khamesra, Phys. Rev. D **101**, 063011 (2020).
[13] S. Ossokine, A. Buonanno, S. Marsat, R. Cotesta, S. Babak, T. Dietrich, R. Haas, I. Hinder, H. P. Pfeiffer, M. Purrer, C. J. Woodford, M. Boyle, L. E. Kidder, M. A. Scheel, and B. Szilágyi, Phys. Rev. D **102**, 044055 (2020).
[14] T. Dietrich, S. Khan, R. Dudi, S. J. Kapadia, P. Kumar, A. Nagar, F. Ohme, F. Pannarale, A. Samajdar, S. Bernuzzi, G. Carullo, W. del Pozzo, M. Haney, C. Markakis, M. Puerrer, G. Riemenschneider, Y. E. Setyawati, K. W. Tsang, and C. van Den Broeck, Phys. Rev. D **99**, 024029 (2019).
[15] T. Dietrich, A. Samajdar, S. Khan, N. K. Johnson-McDaniel, R. Dudi, and W. Tichy, Phys. Rev. D **100**, 044003 (2019).
[16] G. Pratten, S. Husa, C. Garc'ia-Quir'os, M. Colleoni, A. Ramos-Buades, H. Estell'es, and R. Jaume, Phys. Rev. D **102**, 064001 (2020).
[17] L. London, S. Khan, E. Fauchon-Jones, C. García, M. D. Hannam, S. Husa, X. Jiménez-Forteza, C. Kalaghatgi, F. Ohme, and F. Pannarale, Phys. Rev. Lett. **120**, 161102 (2018).
[18] A. Nagar, G. Riemenschneider, G. Pratten, P. Rettegno, and F. Messina, Phys. Rev. D **102**, 024077 (2020).
[19] G. Pratten, C. Garc'ia-Quir'os, M. Colleoni, A. Ramos-Buades, H. Estell'es, M. Mateu-Lucena, R. Jaume, M. Haney, D. Keitel, J. E. Thompson, and S. Husa, arXiv:2004.06503.
[20] C. Garc'ia-Quir'os, M. Colleoni, S. Husa, H. Estell'es, G. Pratten, A. Ramos-Buades, M. Mateu-Lucena, and R. Jaume, Phys. Rev. D **102**, 064002 (2020).
[21] R. Abbott *et al.* (LIGO Scientific and Virgo Collaborations), Phys. Rev. D **102**, 043015 (2020).
[22] R. Abbott *et al.* (LIGO Scientific and Virgo Collaborations), Astrophys. J. Lett. **896**, L44 (2020).
[23] C. Devine, Z. B. Etienne, and S. Mcwilliams, Classical Quantum Gravity **33**, 125025 (2016).
[24] T. D. Knowles, C. Devine, D. A. Buch, S. A. Bilgili, T. R. Adams, Z. B. Etienne, and S. T. Mcwilliams, Classical Quantum Gravity **35**, 155003 (2018).
[25] A. Nagar and P. Rettegno, Phys. Rev. D **99**, 021501 (2019).
[26] R. Smith, S. E. Field, K. Blackburn, C.-J. Haster, M. Pürrer, V. Raymond, and P. Schmidt, Phys. Rev. D **94**, 044031 (2016).
[27] S. Morisaki and V. Raymond, Phys. Rev. D **102**, 104020 (2020).
[28] P. Canizares, S. E. Field, J. R. Gair, V. Raymond, R. C. Smith, and M. Tiglio, Phys. Rev. Lett. **114**, 071104 (2015).
[29] P. Canizares, S. E. Field, and J. R. Gair, M. T. I. of Astronomy, Cambridge, U. of Maryland, C. Park, C. I. of Technology, and Pasadena, Phys. Rev. D **87**, 124005 (2013).
[30] B. Zackay, L. Dai, and T. Venumadhav, arXiv:1806.08792.
[31] N. J. Cornish and K. E. Shuman, Phys. Rev. D **101**, 124008 (2020).
[32] S. Vinciguerra, J. Veitch, and I. Mandel, Classical Quantum Gravity **34**, 115006 (2017).
[33] C. Garc'ia-Quir'os, S. Husa, M. Mateu-Lucena, and A. Borchers, arXiv:2001.10897.
[34] R. C. Smith, C. Hanna, I. Mandel, and A. Vecchio, Phys. Rev. D **90**, 044074 (2014).
[35] X. Guo, Q. Chu, S. K. Chung, Z. Du, and L. Wen, arXiv:abs/1702.02256.
[36] D. Keitel and G. M. Ashton, Classical Quantum Gravity **35**, 205003 (2020).
[37] M. L. Katz, S. Marsat, A. J. K. Chua, S. Babak, and S. L. Larson, Phys. Rev. D **102**, 023033 (2020).
[38] C. Talbot, R. S. Smith, E. Thrane, and G. B. Poole, Phys. Rev. D **100**, 043030 (2019).
[39] D. Wysocki, R. O'Shaughnessy, and Y.-L. L. Fang, J. L. C. for Computational Relativity, Gravitation, R. I. of Technology, C. S. Initiative, and B. N. Laboratory, Phys. Rev. D **99**, 084026 (2019).
[40] S. A. Usman, A. H. Nitz, I. Harry, C. M. Biwer, D. A. Brown, M. Cabero, C. D. Capano, T. dal Canton, T. Dent, S. Fairhurst, M. S. Kehl, D. G. Keppel, B. Krishnan, A. Lenon, A. B. Lundgren, A. B. Nielsen, L. Pekowsky, H. P. Pfeiffer, P. R. Saulson, M. West, and J. L. Willis, Classical Quantum Gravity **33**, 215004 (2016).
[41] S. E. Field, C. R. Galley, J. S. Hesthaven, J. Kaye, and M. Tiglio, Phys. Rev. X **4**, 031006 (2014).
[42] R. J. Smith, K. Cannon, C. S. Hanna, D. G. Keppel, and I. Mandel, Phys. Rev. D **87**, 122002 (2013).
[43] M. Pürrer, Classical Quantum Gravity **31**, 195010 (2014).
[44] K. Cannon, C. S. Hanna, and D. G. Keppel, Phys. Rev. D **85**, 081504 (2012).
[45] D. Barta and M. Vas'uth, Phys. Rev. D **97**, 124011 (2018).
[46] Y. E. Setyawati, F. Ohme, and S. Khan, Phys. Rev. D **99**, 024010 (2019).
[47] K. Cannon, J. Emberson, C. S. Hanna, D. G. Keppel, and H. P. Pfeiffer, Phys. Rev. D **87**, 044008 (2013).
[48] Y. Setyawati, M. Pürrer, and F. Ohme, Classical Quantum Gravity **37**, 075012 (2020).
[49] M. Abadi *et al.*, TensorFlow: Large-scale machine learning on heterogeneous systems, (2015), software available from tensorflow.org.
[50] C. M. Bishop, Rev. Sci. Instrum. **65**, 1803 (1994).
[51] J. Hesthaven and S. Ubbiali, J. Comput. Phys. **363**, 55 (2018).
[52] P. Jacquier, A. Abdedou, V. Delmas, and A. Soulaimani, J. Comput. Phys. **424**, 109854 (2021).
[53] H. Gao, J.-X. Wang, and M. J. Zahr, Physica (Amsterdam) **412D**, 132614 (2020).
[54] N. T. Mucke, L. Hjuler Christiansen, A. P. Engsig-Karup, and J. Bagterp Jorgensen, in *2019 IEEE 58th Conference on Decision and Control (CDC)*, (IEEE, 2019), Vol. 2019-December, pp. 4267–4272, arXiv:1904.06965.

[55] O. San, R. Maulik, and M. Ahmed, Commun. Nonlinear Sci. Numer. Simul. **77**, 271 (2019).
[56] K. Wong, G. Contardo, and S. Ho, Phys. Rev. D **101**, 123005 (2020).
[57] K. W. K. Wong, K. K. Y. Ng, and E. Berti, arXiv: 2007.10350.
[58] D. Gerosa, G. Pratten, and A. Vecchio, Phys. Rev. D **102**, 103020 (2020).
[59] J. P. Marulanda, C. A. Santa, and A. E. Romano, Phys. Lett. B **810**, 135790 (2020).
[60] M. B. Schäfer, F. Ohme, and A. H. Nitz, Phys. Rev. D **102**, 063015 (2020).
[61] C. Dreissigacker, R. Sharma, C. Messenger, R. Zhao, and R. Prix, Phys. Rev. D **100**, 044009 (2019).
[62] B. Beheshtipour and M. A. Papa, Phys. Rev. D **101**, 064009 (2020).
[63] H. Gabbard, C. Messenger, I. S. Heng, F. Tonolini, and R. Murray-Smith, arXiv:abs/1909.06296.
[64] L. Haegel and S. Husa, Classical Quantum Gravity **37**, 135005 (2020).
[65] S. R. Green, C. M. Simpson, and J. R. Gair, arXiv:abs/2002.07656.
[66] E. Cuoco, J. Powell, M. Cavaglia, K. Ackley, M. Bejger, C. Chatterjee, M. W. Coughlin, S. Coughlin, P. J. Easter, R. Essick, H. Gabbard, T. Gebhard, S. Ghosh, L. Haegel, A. Iess, D. Keitel, Z. Marka, S. M'arka, F. Morawski, T. M. Nguyen, R. Ormiston, M. Puerrer, M. Razzano, K. Staats, G. Vajente, and D. R. Williams, arXiv:2005.03745.
[67] B. Lin, X. Li, and W. Yu, Front. Phys. **15**, 24602 (2020).
[68] H. Shen, E. A. Huerta, and Z. Zhao, arXiv:abs/1903.01998.
[69] H. Shen, D. George, E. A. Huerta, and Z. Zhao, in *ICASSP 2019—2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2019), p. 3237, https://ieeexplore.ieee.org/document/8683061.
[70] M. Carrillo, J. A. González, M. Gracia-Linares, and F. S. Guzm'an, J. Phys. Conf. Ser. **654**, 012001 (2015).
[71] Y.-C. Lin and J.-H. Proty Wu, arXiv:2007.04176.
[72] P. Graff, F. Feroz, M. P. Hobson, and A. N. Lasenby, arXiv: abs/1309.0790.
[73] A. J. K. Chua, M. L. Katz, N. Warburton, and S. A. Hughes, arXiv:2008.06071.
[74] A. J. K. Chua, C. R. Galley, and M. Vallisneri, Phys. Rev. Lett. **122**, 211101 (2019).
[75] P. Amaro-Seoane *et al.*, arXiv:1702.00786.
[76] A. Bohé, L. Shao, A. Taracchini, A. Buonanno, S. Babak, I. W. Harry, I. Hinder, S. Ossokine, M. Pürrer, V. Raymond, T. Chu, H. Fong, P. Kumar, H. P. Pfeiffer, M. Boyle, D. A. Hemberger, L. E. Kidder, G. Lovelace, M. A. Scheel, and B. Szilágyi, Phys. Rev. D **95**, 044028 (2017).
[77] A. J. K. Chua and M. Vallisneri, Phys. Rev. Lett. **124**, 041102 (2020).
[78] J. Blackman, S. E. Field, M. A. Scheel, C. R. Galley, D. A. Hemberger, P. Schmidt, and R. Smith, Phys. Rev. D **95**, 104023 (2017).
[79] C. R. Galley, RomPy, https://bitbucket.org/chadgalley/rompy (2020).
[80] W. Hoffmann, Computing **41**, 335 (2015).
[81] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera, C. R. Math. **339**, 667 (2004).
[82] Y. Maday, N. Nguyen, A. Patera, and G. S. H. Pau, Commun. Pure Appl. Anal. **8**, 383 (2009).
[83] S. Ossokine, A. Buonanno, S. Marsat, R. Cotesta, S. Babak, T. Dietrich, R. Haas, I. Hinder, H. P. Pfeiffer, M. Pürrer, C. J. Woodford, M. Boyle, L. E. Kidder, M. A. Scheel, and B. Szilágyi, Phys. Rev. D **102**, 044055 (2020).
[84] R. Cotesta, S. Marsat, and M. Pürrer, Phys. Rev. D **101**, 124040 (2020).
[85] C. R. Galley and P. Schmidt, arXiv:1611.07529.
[86] S. Vinciguerra, J. Veitch, and I. Mandel, Classical Quantum Gravity **34**, 115006 (2017).
[87] C. García-Quirós, S. Husa, M. Mateu-Lucena, and A. Borchers, Classical Quantum Gravity **38**, 015006 (2021).
[88] C. Cutler and E. E. Flanagan, Phys. Rev. D **49**, 2658 (1994).
[89] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (The MIT Press, Cambridge, MA, 2016).
[90] N. E. M. Rifat, S. E. Field, G. Khanna, and V. Varma, Phys. Rev. D **101**, 081502 (2020).
[91] F. Chollet *et al.*, "Keras," https://keras.io (2015).
[92] E. Hoffer, I. Hubara, and D. Soudry, Adv. Neural Inf. Process. Syst. **30**, 1729 (2017), arXiv:1705.08741.
[93] D. Granziol, arXiv:abs/2006.09092 (2020).
[94] D. S. Deighan, S. E. Field, C. D. Capano, and G. Khanna, arXiv:2010.04340.
[95] D. P. Kingma and J. Ba, arXiv:abs/1412.6980.
[96] https://dcc.ligo.org/LIGO-T1800044/public.
[97] S. Husa, S. Khan, M. Hannam, M. Purrer, F. Ohme, X. J. Forteza, and A. Boh'e, Phys. Rev. D **93**, 044006 (2016).
[98] https://www.tensorflow.org/xla.
[99] S. Khan, S. Husa, M. Hannam, F. Ohme, M. Pürrer, X. J. Forteza, and A. Bohé, Phys. Rev. D **93**, 044007 (2016).
[100] C. Pankow, P. Brady, E. Ochsner, and R. O'Shaughnessy, Phys. Rev. D **92**, 023002 (2015).
[101] J. M. Lange, R. O'Shaughnessy, and M. Rizzo, arXiv: 1805.10457.
[102] I. Harry, S. Privitera, A. Bohé, and A. Buonanno, Phys. Rev. D **94**, 024012 (2016).
[103] S. Roy, A. S. Sengupta, and P. Ajith, Phys. Rev. D **99**, 024048 (2019).
[104] V. Baibhav, E. Berti, D. Gerosa, M. Mapelli, N. Giacobbo, Y. Bouffanais, and U. N. di Carlo, Phys. Rev. D **100**, 064060 (2019).
[105] B. Abbott *et al.* (KAGRA, LIGO Scientific, and VIRGO Collaborations), Living Rev. Relativity **21**, 3 (2018).
[106] S. R. Green and J. Gair, arXiv:2008.03312.
[107] J. Thompson, E. Fauchon-Jones, S. Khan, E. Nitoglia, F. Pannarale, T. Dietrich, and M. Hannam, Phys. Rev. D **101**, 124059 (2020).
[108] A. Matas, T. Dietrich, A. Buonanno, T. Hinderer, M. Purrer, F. Foucart, M. Boyle, M. Duez, L. Kidder, H. P. Pfeiffer, and M. Scheel, Phys. Rev. D **102**, 043023 (2020).
[109] B. Abbott *et al.* (LIGO Scientific, Virgo, Fermi GBM, INTEGRAL, IceCube, AstroSat Cadmium Zinc Telluride Imager Team, IPN, Insight-Hxmt, ANTARES, Swift, AGILE Team, 1M2H Team, Dark Energy Camera GW-EM, DES, DLT40, GRAWITA, Fermi-LAT, ATCA, ASKAP, Las Cumbres Observatory Group, OzGrav, DWF (Deeper Wider Faster Program), AST3, CAASTRO,

VINROUGE, MASTER, J-GEM, GROWTH, JAGWAR, CaltechNRAO, TTU-NRAO, NuSTAR, Pan-STARRS, MAXI Team, TZAC Consortium, KU, Nordic Optical Telescope, ePESSTO, GROND, Texas Tech University, SALT Group, TOROS, BOOTES, MWA, CALET, IKI-GW Follow-up, H.E.S.S., LOFAR, LWA, HAWC, Pierre Auger, ALMA, Euro VLBI Team, Pi of Sky, Chandra Team at McGill University, DFN, ATLAS Telescopes, High Time Resolution Universe Survey, RIMAS, RATIR, SKA South Africa/MeerKAT), Astrophys. J. Lett. **848**, L12 (2017).

[110] J. Blackman, S. E. Field, M. A. Scheel, C. R. Galley, C. D. Ott, M. Boyle, L. E. Kidder, H. P. Pfeiffer, and B. Szilágyi, Phys. Rev. D **96**, 024058 (2017).

[111] C. C. Margossian, arXiv:1811.05031.

[112] S. Duane, A. Kennedy, B. J. Pendleton, and D. Roweth, Phys. Lett. B **195**, 216 (1987).

[113] R. Neal, Handbook of markov chain monte carlo **2**, 2 (2011), arXiv:1206.1901.

[114] M. Betancourt, arXiv:1701.02434.

[115] E. K. Porter and J. Carr'e, Classical Quantum Gravity **31**, 145004 (2014).

[116] Y. Bouffanais and E. K. Porter, Phys. Rev. D **100**, 104023 (2019).

[117] www.dirac.ac.uk.

[118] B. Adcock and N. Dexter, arXiv:2001.07523.