Deep neural networks to enable real-time multimessenger astrophysics

Daniel George[1,2,*] and E. A. Huerta[1,2]

[1]Department of Astronomy, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA
[2]NCSA, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA

(Received 18 October 2017; published 26 February 2018)

Gravitational wave astronomy has set in motion a scientific revolution. To further enhance the science reach of this emergent field of research, there is a pressing need to increase the depth and speed of the algorithms used to enable these ground-breaking discoveries. We introduce `Deep Filtering`—a new scalable machine learning method for end-to-end time-series signal processing. `Deep Filtering` is based on deep learning with two deep convolutional neural networks, which are designed for *classification* and *regression*, to detect gravitational wave signals in highly noisy time-series data streams and also estimate the parameters of their sources in real time. Acknowledging that some of the most sensitive algorithms for the detection of gravitational waves are based on implementations of matched filtering, and that a matched filter is the optimal linear filter in Gaussian noise, the application of `Deep Filtering` using whitened signals in Gaussian noise is investigated in this foundational article. The results indicate that `Deep Filtering` outperforms conventional machine learning techniques, achieves similar performance compared to matched filtering, while being several orders of magnitude faster, allowing real-time signal processing with minimal resources. Furthermore, we demonstrate that `Deep Filtering` can detect and characterize waveform signals emitted from new classes of eccentric or spin-precessing binary black holes, even when trained with data sets of only quasicircular binary black hole waveforms. The results presented in this article, and the recent use of deep neural networks for the identification of optical transients in telescope data, suggests that deep learning can facilitate real-time searches of gravitational wave sources and their electromagnetic and astroparticle counterparts. In the subsequent article, the framework introduced herein is directly applied to identify and characterize gravitational wave events in real LIGO data.

DOI: 10.1103/PhysRevD.97.044039

I. INTRODUCTION

Gravitational wave (GW) astrophysics is a well-established field of research. To date, the Advanced Laser Interferometer Gravitational Wave Observatory (aLIGO) detectors [1,2] have detected five GW events from binary black hole (BBH) mergers that are consistent with Einstein's general relativity predictions [3–7].

By the end of aLIGO's second discovery campaign (O2), the European advanced Virgo (aVirgo) detector [8] joined aLIGO, establishing the first, three-detector search for GW sources in the advanced detector era. This international network was critical for the detection of the fifth BBH merger with improved sky localization, and also provided the means to carry out new phenomenological tests of gravity [7].

The international aLIGO-aVirgo detector network was used for the first detection of GWs from two colliding neutron stars (NSs), GW170817 [9–11], which was followed up with broadband electromagnetic observations after several hours [12]. These multimessenger observations led to the first direct confirmation that NS mergers are the progenitors of gamma-ray bursts, GRB170817A [13–19], and the cosmic factories where about half of all elements heavier than iron are produced [12]. These major scientific breakthroughs, worthy of the 2017 Nobel Prize in Physics, have initiated a new era in contemporary astrophysics.

Ongoing improvements in the sensitivity of aLIGO and aVirgo, will enable future multimessenger observations with astronomical facilities [20–25], increasing the number and types of GW sources, and providing new and detailed information about the astrophysical origin, and cosmic evolution of compact objects.

Multimessenger astrophysics is an interdisciplinary program that brings together experimental and theoretical physics, cosmology, fundamental physics, high-performance computing (HPC) and high-throughput computing (HTC). For instance, at the interface of HPC and theoretical physics, numerical relativity (NR) simulations of Einstein's field equations are extensively used to validate the astrophysical

*Corresponding author.
dgeorge5@illinois.edu

nature of GW sources [26,27]. Furthermore, NR simulations of NS mergers, neutron star–black hole mergers, core collapse supernovae and other massive, relativistic systems provide key physical insights into the physics of GW sources that are expected to generate electromagnetic (EM) and astroparticle counterparts [18,28–32].

On the other hand, large-scale GW data analysis has traditionally relied on HTC resources. Flagship GW searches have been very successful at exploiting these resources to identify and characterize GW sources [33–36]. Within the next few years GW discovery campaigns will bring together an international network of GW interferometers, that will gather data for extended periods of time. As the sensitivity of this detector network reaches design sensitivity, the detection rate will continue to increase in successive detection campaigns. Furthermore, existing low-latency (online) matched-filtering-based algorithms currently target a four-dimensional parameter space, which describes spin-aligned compact binary systems.[1]

Accelerating the offline Bayesian parameter estimation algorithms, which typically last from several hours to a few days, is no trivial task since they have to sample a 15-dimensional parameter space [37–40]. In light of these challenges, there are ongoing efforts to reduce the size of template banks used for matched-filtering-based GW searches [41]. Based on these considerations, and realizing that to maximize the science one can extract from GW observations, it is essential to rapidly cover a deeper parameter space of astrophysically motivated sources, the GW community has been exploiting state-of-the-art HPC facilities to increase the pool of computational resources to carry out for large scale GW data analysis [42,43].

To further contribute to fully realize the multimessenger astrophysics program, this article introduces a new machine (deep) learning algorithm, `Deep Filtering`, which is based on deep neural networks (DNNs) [44] to directly process highly noisy time-series data for both classification and regression in real time. In particular, this algorithm consists of two deep convolutional neural networks [45] that take time-series inputs, and are capable of detecting and estimating parameters of GW signals whose peak power is weaker than that of the background noise.

The main objective in developing `Deep Filtering` is to complement and enhance the existing, *low-latency* GW detection algorithms, such as `PyCBC` [33] and `gstLAL` [46], to enable deeper and faster GW searches. `Deep Filtering` may be applied to identify and rapidly constrain the astrophysical parameters of GW transients. This real-time analysis would then be followed up by existing *offline* Bayesian parameter estimation pipelines [37,38]. A targeted search of this nature can significantly reduce the size of multidimensional template banks, enabling the use of established matched-filtering searches at a fraction of their computational cost to quantify the significance of new GW detections. This approach would combine the best of two approaches: the scalable nature of DNNs with the sophistication of LIGO-Virgo detection pipelines.

In this foundational article, we describe the key features of `Deep Filtering` and carry out a systematic study of DNNs trained using a data set of inspiral-merger-ringdown BBH waveforms [47,48] to cover the BBH parameter space where ground-based GW detectors are expected to have the highest detection rate [49]. This analysis is carried out using GW signals whitened with aLIGO's design sensitivity [50] injected into Gaussian noise. This simplified scenario is studied in this first article to illustrate the key ideas and new deep learning methods in a transparent manner, and also to compare these results to a matched filter, the optimal linear filter in Gaussian noise, which is at the core of some of the most sensitive GW detection pipelines [33,34]. In the subsequent article, the methods presented here are successfully applied for the detection and characterization of GW signals in real LIGO data [51].

The results in this article suggest that DNNs may be ideal tools for enhancing GW analysis. In particular, DNNs are able to *interpolate* between waveform templates, in a similar manner to Gaussian process regression (GPR),[2] and to *generalize* to some new classes of signals beyond the templates used for training. An important advantage of `Deep Filtering` is its scalability, i.e., all the intensive computation is diverted to the one-time training stage, after which the data sets can be discarded, i.e., the size of the template banks presents no limitation when using deep learning. With existing computational resources on supercomputers, such as Blue Waters, it will be feasible to train DNNs that target a nine-dimensional parameter space within a few weeks. Furthermore, once trained these DNNs can be evaluated in real time with a single CPU, and more intensive searches over longer time periods covering a broader range of signals can be carried out with a dedicated GPU.

The analysis presented here, contextualized with recent work to understand and characterize aLIGO non-Gaussian noise transients [55,56], and new deep learning applications for transient identification in large sky surveys [57] suggests that it is feasible to create an efficient deep learning pipeline to perform all tasks—identifying the presence or absence of GW signals, classifying noise transients, reconstructing the astrophysical properties of detected GW sources, and identification of EM counterparts of GW events, thus paving a natural path to

[1]Astrophysically motivated sources describe a nine-dimensional parameter space: two component masses, eccentricity, and two three-dimensional vectors describing the spin of each binary component.

[2]GPR [52–54] is a statistical tool that can serve as a probabilistic interpolation algorithm providing information about the training set of NR simulations needed to accurately describe a given parameter space and generates interpolated waveforms that match NR counterparts above any given level of accuracy.

realizing real-time multimessenger astrophysics with a unified framework.

This article is organized as follows. Section II provides a comprehensive overview of artificial neural networks and deep learning, particularly focusing on convolutional neural networks in the context of time-series signal processing. Section III describes the assumptions, data sets, and procedure to construct the DNN-based GW analysis pipeline. The results are reported in Sec. IV. In Sec. V, the immediate implications for GW astrophysics missions are discussed. We summarize the findings and outline their broader applications in Sec. VI.

II. NEURAL NETWORKS AND DEEP LEARNING

This section presents a brief overview of the main concepts of deep learning, including machine learning, artificial neural networks, and convolutional neural networks in the context of time-series signal processing.

The vast majority of algorithms are designed with a specific task in mind. They require extensive modifications before they can be reused for any other task. The term machine learning refers to a special class of algorithms that can *learn* from examples to solve new problems without being explicitly reprogrammed. This enables cross-domain applications of the same algorithm by training it with different data [58]. More importantly, some of these algorithms are able to tackle problems which humans can solve intuitively but find difficult to explain using well-defined rules; hence they are often called "artificial intelligence" [58].

The two main categories of machine learning are supervised and unsupervised learning. In supervised learning, the algorithm learns from some data that is correctly labeled, while unsupervised learning algorithms have to make sense of unstructured and unlabeled data [59]. This work focuses on an application of supervised learning, where labeled data obtained from physics simulations is used to train an algorithm to detect signals embedded in noise and also estimate multiple parameters of the source.

Although traditional machine learning algorithms have been successful in several applications, they are limited in their ability to deal directly with raw data. Often the data has to be simplified manually into a representation suitable for each problem. Determining the right representation is extremely difficult and time consuming, often requiring decades of effort even for domain experts, which severely limits the applicability of these algorithms [58]. Representation learning is a field of machine learning which aims to resolve this issue by creating algorithms that can learn by themselves to find useful representations of the raw data and extract relevant features from it automatically for each problem [60].

Deep learning is one of the most rapidly growing subfields of machine learning, which resolves this difficulty of feature engineering with algorithms that can find useful representations of the raw data by extracting multiple levels of relevant features automatically for each problem. This is achieved by combining a computational architecture containing long interconnected layers of "artificial neurons" with powerful learning (optimization) algorithms [44,58]. These DNNs are able to capture complex nonlinear relationships in the data by composing hierarchical internal representations, all of which are learned automatically during the training stage. The deepest layers are able to learn highly abstract concepts, based on the simpler outputs of the previous layers, to solve problems that previously required human-level intelligence thus achieving state-of-the-art performance for many tasks [59].

A. Artificial neural networks

Artificial neural networks (ANNs), the building blocks of DNNs, are biologically inspired computational models that have the capability to learn from observational data [61]. The fundamental units of neural networks are artificial neurons (loosely modeled after real neurons [62]), which are based on perceptrons introduced by Rosenblatt in 1957 [63]. A perceptron takes a vector of inputs ($\vec{x}$) and computes a weighted output with an offset known as bias. This can be modeled by the equation $f(\vec{x}) = \vec{w} \cdot \vec{x} + b$, where the weights ($\vec{w}$) and bias ($b$) are learned through training.

Minsky and Papert showed that a single perceptron has many limitations [64]. However, it was later found that these limitations can be overcome by using multiple layers of interconnected perceptrons to create ANNs [59]. The universality theorem [65] proves that ANNs with just three layers (one hidden layer) can model any function up to any desired level of accuracy.

Multilayer perceptrons are also known as feed-forward neural networks because information is propagated forward from the input layer to the output layer without internal cycles (i.e. no feedback loops) [58]. While potentially more powerful cyclic architectures can be constructed, such as recurrent neural networks [58], they are often more computationally expensive to train. Therefore, only feed-forward neural networks are considered in this article.

An ANN usually has an input layer, one or more hidden layers, and an output layer (shown in Fig. 1). A nonlinear "activation" function is applied to the output of each of the hidden layers. Without this nonlinearity, using multiple layers would become redundant, as the network will only be able to express linear combinations of the input. The most commonly used nonlinear activation functions are the logistic sigmoid, hyperbolic tan, and the rectified linear unit (also called ReLU or ramp). It has been empirically observed that the ramp produces the best results for most applications [66]. This function is mathematically expressed as $\max(0, x)$.

The key ingredient that makes ANNs useful is the learning algorithm. Almost all neural networks used today are trained with variants of the back-propagation algorithm in conjunction with the gradient descent methods [59].

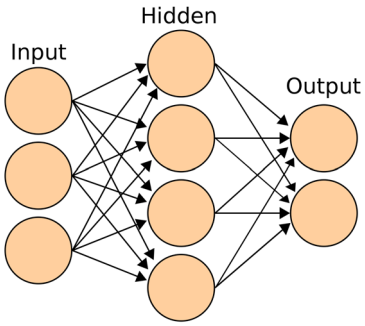

FIG. 1. *Diagram of a neural network.* An ANN or multilayer perceptron with one hidden layer is depicted [67]. The circles represent neurons and arrows represent connections (weights) between neurons. Note that each neuron has only a *single* output, which branches out to connect with neurons in the next layer.

The idea is to propagate errors backward from the output layer to the input layer after each evaluation of a neural network, in order to adjust the weights of each neuron so that the overall error is reduced in a supervised learning problem [68]. The weights of an ANN are usually initialized randomly to small values and then back propagation is performed over multiple rounds, known as epochs, until the errors are minimized. Stochastic gradient descent with mini-batches [69] has been the traditional method used for learning. This technique uses an estimate of the gradient of the error over subsets of the training data in each iteration to change the weights of the ANN. The magnitude of these changes is determined by the "learning rate." New methods with variable learning rates such as adaptive momentum estimation (ADAM) are becoming more popular and have been shown empirically to achieve better results more quickly for a wide range of problems [70].

B. Convolutional neural networks

A convolutional neural network [45] (CNN), whose structure is inspired by studies of the visual cortex in mammals [58], is a type of feed-forward neural network. CNNs have been found to approach or even surpass human-level accuracy[3] at a variety of image and video processing tasks [44,71].

The introduction of a "convolution layer," containing a set of neurons that share their weights, is the critical component of these networks. Multiple convolution layers are commonly found in DNNs, with each having a separate set of shared weights that are learned during training. The name comes from the fact that an output equivalent to a convolution, or sometimes cross-correlation [58], operation is computed with a kernel of fixed size. A convolutional layer can also be viewed as a layer of identical neurons that each "look" at small overlapping sections of the input, defined as the receptive field.

The main advantage of using these layers is the ability to reduce computational costs by having shared weights and small kernels, thus allowing deeper networks and faster training and evaluation speeds. Because of the shared weights, CNNs are also able to automatically deal with spatially translated as well as (with a few modifications [44]) rotated and scaled signals. In practice, multiple modules each consisting of a sequence of convolution and pooling (subsampling) layers, followed by a nonlinearity, are used. The pooling layers further reduce computational costs by constraining the size of the DNN, while also making the networks more resilient to noise and translations, thus enhancing their ability to handle new inputs [44]. Dilated convolutions [72] are a recent development which enables rapid aggregation of information over larger regions by having gaps within each of the receptive fields. In this study, we focus on CNNs as they are the most efficient DNNs on modern hardware, allowing fast training and evaluation (inference).

C. Time-series analysis with convolutional neural networks

Conventional methods for digital signal processing such as matched filtering (cross-correlation or convolution against a set of templates) [73] in time-domain or frequency space are limited in their ability to scale to a large parameter space of signal templates, as discussed in Refs. [39,41], while being too computationally intensive for real-time parameter estimation analyses [37]. Signal processing using machine learning in the context of GW astrophysics is an emerging field of research [55,56,74–79]. These traditional machine learning techniques, including shallow ANNs, require "handcrafted" features extracted from the data as inputs rather than the raw noisy data itself. DNNs, on the other hand, are capable of extracting these features automatically.

Deep learning has been previously applied for the classification of glitches with spectrogram images as inputs to CNNs [56,78,80] and unsupervised clustering of transients [81], in the context of aLIGO. Using images as inputs is advantageous for two reasons: (i) there are well-established architectures of two-dimensional CNNs which have been shown to work (GoogLeNet [82], VGG [83], ResNet [84]); and (ii) pretrained weights are available for them, which can speed up the training process via transfer learning while also providing higher accuracy even for small data sets [56]. However, experiments showed that this approach would not be optimal for detection or parameter estimation since many signals having low signal-to-noise ratio (SNR[4]) are not visible in spectrograms, as shown in Fig. 2.

[3]In the context of classification, accuracy is defined as the ratio of inputs whose labels were predicted correctly with respect to the total number of inputs.

[4]Note that the standard definition of optimal matched-filtering SNR is used in this article, as described in Ref. [85]. This SNR is on average proportional to 12.9 ± 1.4 times the ratio of the amplitude of the signal to the standard deviation of the noise for the test set.

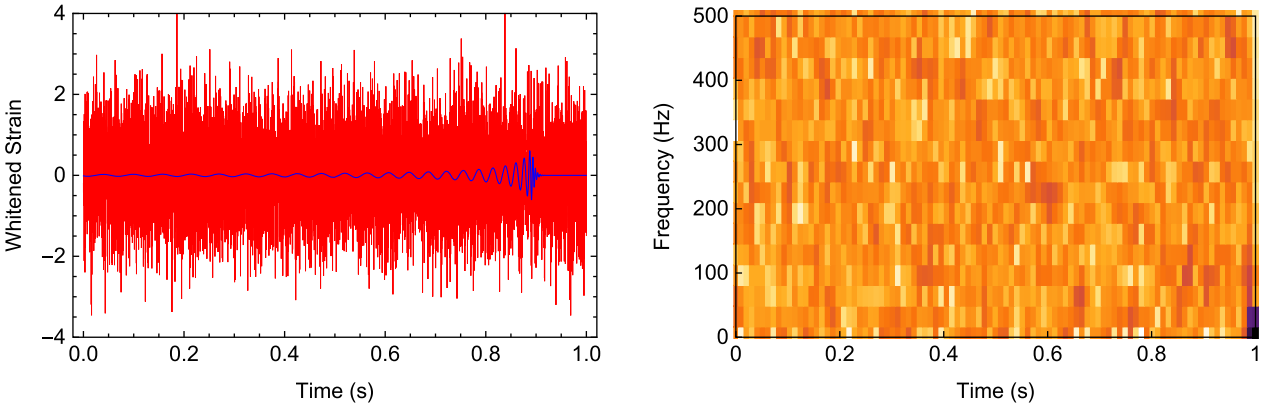

FIG. 2. *Sample of input data.* The red time series is an example of the input to the deep neural network algorithm. It contains a binary black hole agravitational waveform signal (blue), which was whitened with aLIGO's design sensitivity and superimposed in noisy data with SNR = 7.5 (the peak power of this signal is 0.36 times the power of background noise). The component masses of the merging black holes are 57 $M_\odot$ and 33 $M_\odot$, respectively. The corresponding spectrogram on the right shows that the gravitational wave signal on the left is not visible, and thus cannot be detected by an algorithm trained for image recognition. Nevertheless, the deep neural network detects the presence of this signal directly from the (red) time-series input with over 99% sensitivity, and reconstructs the source's parameters with a mean relative error of about 10%.

Theoretically, all the information about the signal is encoded within the time series, whereas spectrograms are lossy noninvertible representations of the original data. Although two-dimensional CNNs are commonly used, especially for image-related tasks, by directly feeding the time-series data as inputs to one-dimensional CNNs, one can obtain higher sensitivities of detection (defined as the fraction of signals detected with respect to the total number of signals present in the inputs) at low SNR, lower error[5] rates in parameter estimation, and faster analysis speeds. This automated feature learning allows the algorithm to develop more optimal strategies of signal processing than when given hand-extracted information such as spectrograms. There has been a few attempts at signal processing using CNNs with raw noisy time-series data in other domains which considered estimation of a single parameter [86,87].

This article demonstrates that DNNs can be used for both signal detection and multiple-parameter estimation directly from highly noisy time-series data, once trained with templates of the expected signals, and that deep CNNs outperform many traditional machine learning algorithms shown in Fig. 14, and reach accuracies comparable to matched-filtering methods. The results show that deep learning is more computationally efficient than matched filtering for GW analysis. Instead of repeatedly performing overlap computations against all templates of known signals, the CNN builds a deep *nonlinear* hierarchical structure of nested convolutions, with small kernels, that determines the parameters in a single evaluation. Moreover, the DNNs act as an efficient compression mechanism by learning patterns and encoding all the relevant information in their weights, analogous to a reduced-order model [88], which is significantly smaller than the size of the training templates. Therefore, the DNNs automatically perform an internal optimization of the search algorithm and can also interpolate, or even extrapolate, to new signals not included in the template bank (unlike matched filtering).

Note that matched filtering performs the convolution of the input data against a set of templates; therefore, it is equivalent to a single convolution layer in a neural network, with very long kernels corresponding to each signal in the template bank. Therefore, `Deep Filtering` can be viewed as a more efficient extension of matched filtering, which performs template matching against a small set of short-duration templates, which are learned automatically, and aggregates this information in the deeper layers to effectively model the full range of long-duration signals.

III. METHOD

As a proof of concept in this first article, we focus on GWs from BBH mergers, which are expected to dominate the number of GW detections with ground-based GW detectors [49,89,90]. Note that this method can be extended to GW signals produced by other types of events by adding more neurons in the final layer corresponding to the number of classes/parameters, changing the size of the input layer depending on the length of the templates, and training with template banks of these GW signals injected into simulated or real noise.

[5]The error on the test set is defined as the mean of the magnitudes (absolute values) of the relative error in estimating each parameter averaged over all inputs in the test set and over each parameter.

We have divided the problem into two separate parts, each assigned to a different DNN, so that they may be used independently. The first network, henceforth known as the "classifier," will detect the presence of a signal in the input, and will provide a confidence level for the detection. The classes chosen for now are "True" or "False" depending on whether or not a signal from a BBH merger is present in the input. The second network, referred to as the "predictor," will estimate the parameters of the source of the signal (in this case, the component masses of the BBH). The predictor is triggered when the classifier identifies a signal with a high probability.

The system is partitioned in this manner so that, in the future, more classes of GW transients [28,29,91], may be added to the classifier, and separate specialized predictors can be made for each type of signal. Moreover, categories for various types of anomalous sources of noise, like glitches and blips [36,80], can also be incorporated in the classifier [56].

A. Assumptions

For this initial study, the signals are assumed to be optimally oriented with respect to the detectors, and that the individual spins and orbital eccentricities are zero. This reduces the parameter space to two dimensions, namely, the individual masses of the BBH systems, which are restricted to lie between 5 $M_\odot$ and 75 $M_\odot$. Furthermore, the inputs are constrained to have a duration of 1 second, and a sampling rate of 8192 Hz throughout this analysis, which was an arbitrary choice made initially, which was found to perform well for the type of events that are considered here. Note that the classifier will be applied to the continuous data stream by using a sliding window of width 1 second. However, it is straightforward to use inputs of any duration by changing a hyperparameter corresponding to the input size of the CNNs, which will result in the computational cost scaling linearly with the length of the input.

Throughout this analysis, the signals are whitened using aLIGO's power spectral density (PSD) at the "zero-detuned high-power" design sensitivity [50], shown in Fig. 3, to approximate the sensitivity of LIGO at different frequencies. Consideration of transient sources of detector noise are deferred to the subsequent article. This is in line with previous studies, which have first showcased a machine learning algorithm for LIGO data analysis using simulated noise [38,77,92], and then followed up with an independent study where the algorithm is tested using real aLIGO noise [76]. In this article, we follow a similar approach by describing the key concepts and methods for the construction of DNNs for GW data analysis in the context of Gaussian noise, and then show in the following article how this `Deep Filtering` algorithm can be directly applied to detect and characterize GW events in real LIGO data, which has non-Gaussian and nonstationary noise including glitches [51].

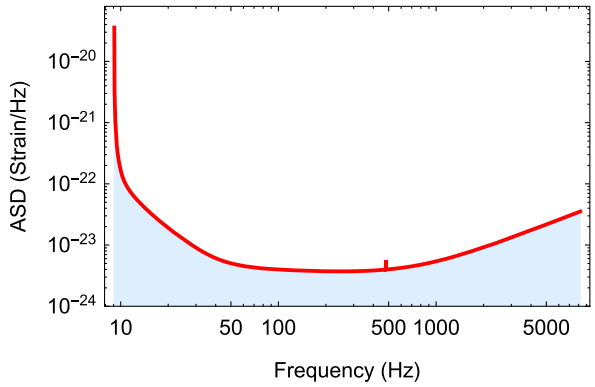

FIG. 3. *Sensitivity curve of aLIGO*. Throughout this analysis, the zero-detuned high-power sensitivity configuration for aLIGO [50] is used to simulate the colored noise in the detectors by whitening the GW signals. The amplitude spectral density (ASD) of the noise vs frequency for this configuration is shown in the figure.

B. Obtaining data

Supervised deep learning algorithms are more effective when trained with large data sets [58]. Obtaining high-quality training data has been a difficult and cumbersome task in most applications of DNNs, such as object recognition in images, speech and text processing, etc. Fortunately, this issue is not faced here since one can take advantage of scientific simulations to produce the necessary data for training.

Over the last decade, sophisticated techniques have been developed to perform accurate three-dimensional NR simulations of merging BHs [91,93]. For the analysis at hand, effective-one-body (EOB) [47,48] waveforms that describe GWs emitted by quasicircular, nonspinning BBHs are used. The final 1 second window of each template is extracted for this analysis.

Following the standard practice in machine learning, the data is split into separate sets for training and testing. For the training data set, the BBHs component masses are in the range 5 $M_\odot$ to 75 $M_\odot$ in steps of 1 $M_\odot$. The testing data set has intermediate component masses, i.e., masses separated from values in the training data set by 0.5 $M_\odot$. By not having overlapping values in the training and testing sets, one can ensure that the network is not overfitting, i.e., memorizing only the inputs shown to it without learning to generalize to new inputs. The distribution of component masses, and a template from the training and testing sets, is shown in Fig. 4.

Subsequently, the location of the peak of each signal is shifted randomly within an interval of 0.2 seconds in both the training and testing sets to make the DNNs more robust with respect to time translations. Next, different realizations of Gaussian white noise are superimposed on top of the signals over multiple iterations, thus amplifying the size of the data sets. The power of the noise was adjusted according to the desired SNR for each training session. As

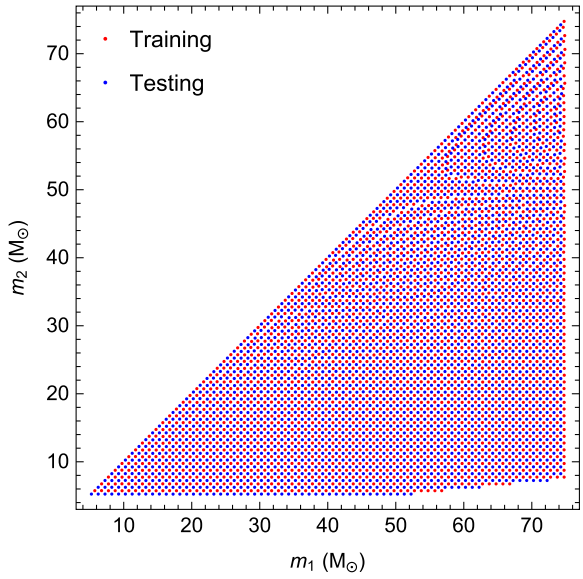

FIG. 4. *Distribution of data*. The figure shows the distribution of component masses of BBHs for the training and testing data sets. The mass ratios are confined between 1 and 10, which accounts for the missing points in the lower right corner. This mass-ratio range is chosen because the state-of-the-art EOB model used to create the data sets has only been validated for these mass-ratio values. Each point represents a quasicircular, nonspinning GW signal of 1 second duration, sampled at 8192 Hz, which is whitened with aLIGO's expected noise spectrum at design sensitivity. These waveforms are normalized and translated randomly in time. Thereafter, multiple batches of noise at each SNR are added to produce training and testing data sets.

	Input	vector (size: 8192)
1	Reshape	matrix (size: 1 × 8192)
2	Convolution	matrix (size: 16 × 8177)
3	Pooling	matrix (size: 16 × 2044)
4	ReLU	matrix (size: 16 × 2044)
5	Convolution	matrix (size: 32 × 2016)
6	Pooling	matrix (size: 32 × 504)
7	ReLU	matrix (size: 32 × 504)
8	Convolution	matrix (size: 64 × 476)
9	Pooling	matrix (size: 64 × 119)
10	ReLU	matrix (size: 64 × 119)
11	Flatten	vector (size: 7616)
12	Linear Layer	vector (size: 64)
13	ReLU	vector (size: 64)
14	Linear Layer	vector (size: 2)
	Output	vector (size: 2)

FIG. 5. *Architecture of the deep neural network*. This is the deep dilated one-dimensional CNN, modified to take time-series inputs, designed for prediction, which outputs two real-valued numbers for the two component masses of the BBH system. For classification, a *softmax* layer was added after the 14th layer to obtain the probabilities for two classes, i.e., "True" or "False." The input is the time series sampled at 8192 Hz and the output is either the probability of each class or the value of each parameter. Note that the number of neurons in layer 14 can be increased to add more categories for classification or more parameters for prediction. The size of this CNN is about 2 MB.

usual, the inputs are standardized to have zero mean and unit variance to make the training process easier [94].

The final training sets at each SNR are produced from ∼2500 templates of GWs from BBH mergers by adding multiple batches of noise and shifting in time. It is also a standard practice to use a validation set to monitor the performance on unseen data during training in order to prevent overfitting. The validation and testing sets at each SNR are generated from a different set of ∼2500 templates by superimposing different noise realizations.

C. Designing neural networks

Similar DNN architectures are used for both the classifier and predictor, which demonstrates the versatility of this method. The only difference is the addition of a *softmax* layer to the classifier to obtain probability estimates as the outputs. The strategy is to first train the predictor on the data sets labeled with the BBH masses, and then transfer the weights of this pretrained network to initialize the classifier and then train it on data sets in which half of the inputs contained an injected signal. This transfer learning process reduces the training time required for the classifier, while also slightly improving its accuracy at low SNR.

Overall, we designed and tested around 80 configurations of DNNs ranging from one to four convolutional layers and one to three fully connected layers (also called linear layers) similar to Ref. [95], but modified for time-series inputs. Among these, a design for the classifier with three convolutional layers followed by two fully connected layers yields good results with the fastest inference speed. We tried adding a few recent developments such as batch normalization [96] and dropout [97] layers. However, they are not used in the final design as they did not provide improvements for the simple problem that is considered here. Note that the addition of noise to the signals during the training process serves as a form of regularization in itself. Many of the layers have parameters, commonly known as hyperparameters, which are tuned manually via a randomized trial-and-error procedure.

Depth is a hyperparameter which determines the number of filters in each convolutional layer. The choices for depth in the consecutive layers are 16, 32, and 64 respectively. Kernel sizes of 16, 8, and 8 are used for the convolutional layers and 4 for all the (max) pooling layers. Stride, which specifies the shift between the receptive fields of adjacent neurons, is chosen to be 1 for all the convolution layers and 4 for all the pooling layers. Dilation determines the overall size of each receptive field, which could be larger than the kernel size by having gaps in between. Here, it is a measure of the temporal extend of the convolutions. Using a dilation of 4 in the final two convolution layers improves the

	Input	vector (size: 8192)
1	Reshape	matrix (size: 1×8192)
2	Convolution	matrix (size: 64×8177)
3	Pooling	matrix (size: 64×2044)
4	ReLU	matrix (size: 64×2044)
5	Convolution	matrix (size: 128×2014)
6	Pooling	matrix (size: 128×503)
7	ReLU	matrix (size: 128×503)
8	Convolution	matrix (size: 256×473)
9	Pooling	matrix (size: 256×118)
10	ReLU	matrix (size: 256×118)
11	Convolution	matrix (size: 512×56)
12	Pooling	matrix (size: 512×14)
13	ReLU	matrix (size: 512×14)
14	Flatten	vector (size: 7168)
15	Linear Layer	vector (size: 128)
16	ReLU	vector (size: 128)
17	Linear Layer	vector (size: 64)
18	ReLU	vector (size: 64)
19	Linear Layer	vector (size: 2)
	Output	vector (size: 2)

FIG. 6. *Architecture of the deeper neural network*. This is the deeper version of the CNN, modified to take time-series inputs, designed for parameter estimation. The input is the time series sampled at 8192 Hz and the output is the predicted value of each parameter. This can be converted to a classifier by adding a *softmax* layer after layer 19 to obtain the probability for a detection. Note that the number of neurons in layer 19 can be increased to add more categories for classification or more parameters for prediction. The two neurons in the final layer output the two parameters corresponding to the individual masses of BBHs. The size of this CNN is approximately 23 MB.

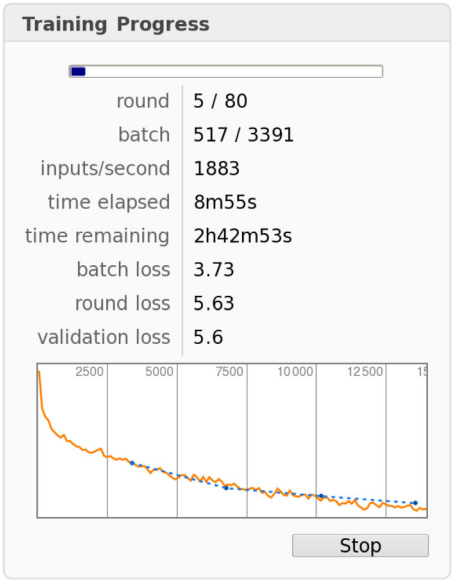

FIG. 7. *Visualization of training*. This is a snapshot of one of the training sessions for parameter estimation. The mean squared error on the training set is plotted in orange and the blue curve measures the error on the validation set.

performance. The final layout of the classifier DNN is shown in Fig. 5.

Deeper networks are expected to provide further improvements in accuracy although at the cost of slower evaluation speed. To show this, we also designed a deeper net, shown in Fig. 6, with four convolution layers and three fully connected layers that had improved sensitivity for detection and significantly better performance for parameter estimation. Although this design performs slightly better, it is a factor of 5 slower on a GPU for evaluation. This CNN has convolution layers whose kernel sizes were 16, 16, 16, and 32 with dilations 1, 2, 2, and 2 respectively. The pooling layers all have kernel size 4 and stride 4.

A loss function (cost function) is required to compute the error after each iteration by measuring how close the outputs are with respect to the target values. A new loss function, i.e., the mean absolute relative error loss, is applied for training the predictor. For classification, the standard cross-entropy loss function [58] is used.

D. Training strategy

Hyperparameter optimization is performed by trial and error to design architectures of the CNNs that achieve the best performance in terms of speed and accuracy. First, we use Gaussian white noise without whitening the signals i.e., a flat PSD, to determine the optimal architectures of the DNNs. This design was also found to be optimal for signals whitened with the zero-detuned PSD of aLIGO. This indicates that the same architecture will perform well on a wide variety of PSDs. Once the best performing DNNs are chosen, they are trained for a total of approximately 10 hours. The DNNs are designed and trained using the neural network functionality in MATHEMATICA, based internally on the open-source MXNet framework [98], which utilizes the CUDA deep learning library (cuDNN) [99] for acceleration using GPUs. The ADAM [70] method is used as the learning algorithm. A snapshot of the training process is shown in Fig. 7.

A new strategy was devised to reduce the training time of DNNs, while also ensuring an optimal performance, by starting off training the predictor on inputs having high SNR ($\geq$100) and then gradually increasing the noise in each subsequent training session until a final SNR distribution randomly sampled in the range $5 \leq \mathrm{SNR} \leq 15$. This process ensures that the performance can be quickly maximized for low SNR, while remaining accurate for signals with high SNR. For instance, 11% error (defined as the mean of the absolute values of the relative error averaged over all the test set elements and over each parameter) is obtained when trained using this scheme, with gradually decreasing SNR, and only about 21% mean error at parameter estimation is obtained on the test set when directly trained on the same range of SNRs without this scheme.

Furthermore, the classifier performs better (with an increase from 96 to 99% accuracy on the test set) when its initial weights are transferred from the fully trained predictor, i.e., the classifier is created by simply adding a *softmax* layer to the trained predictor and then trained on the data set of signals and noise. These techniques are also useful when applying `Deep Filtering` for GW detection and characterization in real LIGO data [51]. Therefore, they may also be useful for training neural networks, in general, with noisy time-series data.

IV. RESULTS

A. Detection

Defining sensitivity as the ratio of the number of correct detections made to the total number of inputs containing signals, at a fixed false-alarm rate, the classifier achieved 100% sensitivity throughout the parameter space for signals with SNR ≥ 10, and a single-detector false-alarm rate less than 0.6%. The false-alarm rate of `Deep Filtering` can be further decreased by combining the classifications on multiple detector inputs and by computing the overlap of the template predicted by `Deep Filtering` with the input data to confirm each detection.

The left panel of Fig. 8 presents the sensitivity of detection using the shallower DNN architecture shown in Fig. 5. After training over the entire range of SNRs, and tuning the single detector false-alarm rate to 0.6%, we found that the sensitivity of detection saturates at 100% for SNR ≥ 10, i.e., GWs with SNRs in this range are always detected. Under the same set of assumptions, i.e., training strategy and single-detector false-alarm rate, but now using the deeper DNN in Fig. 6, the right panel of Fig. 8 indicates that the sensitivity of detection saturates at 100% for SNR ≥ 9, and performs similarly to matched filtering throughout the SNR range used for comparison. These results indicate that `Deep Filtering` can extract GW signals weaker than the background noise.

Note that Fig. 8 showed results averaged over the BBH parameter space under consideration. To further investigate the performance in different regions of the parameter space, Fig. 9 presents the sensitivity of detection, using the deeper DNN shown in Fig. 6, for *each template* in the test set assuming a fixed SNR $= 6$. It is worth pointing out that the sensitivity of detection for *each template* in the test set is 100% for SNR ≥ 10 in each region of parameter space. For very-low-mass BBH systems, at the limit of sensitivity of independent implementations of matched filtering, i.e., SNR ~ 6 [33], the sensitivity of the classifier is relatively lower. This is because for low-mass BBH GWs, the last second of the signal is contained in the high-frequency regime of the aLIGO band (~4.4 kHz/M) beyond aLIGO's range of optimal sensitivity. Therefore, to attain better sensitivity of detection for low-mass systems, the DNNs can be trained using data sets with longer waveform templates, which may be explored in a subsequent article. On the other hand, the DNNs are capable of correctly identifying high-mass BBH events. This is a promising result, because high-mass BBH templates are short lived and they are difficult to accurately extract and characterize in LIGO data, as shown in Ref. [100]. In summary, `Deep Filtering` performs well throughout the BBH parameter space for GW events with SNR ≥ 10, excelling in the detection of high-mass systems even at lower SNR.

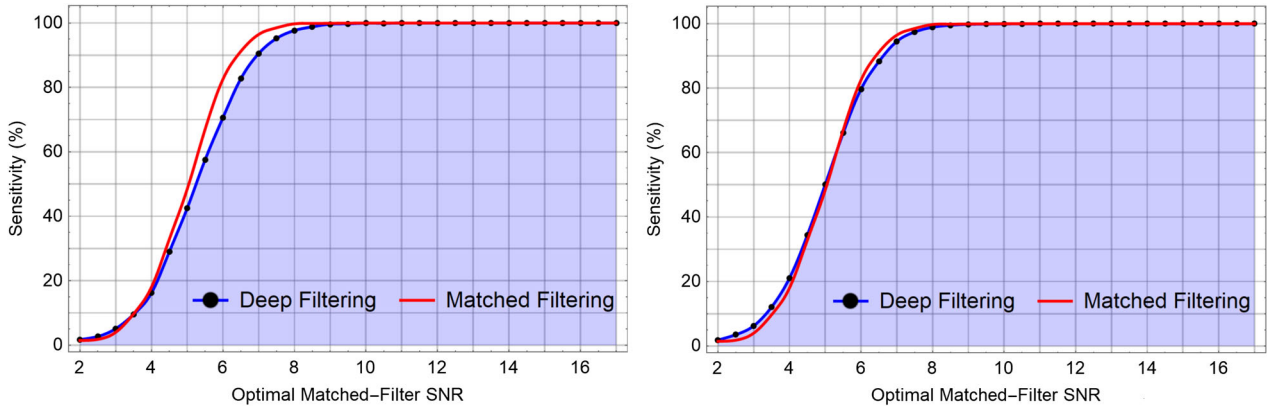

FIG. 8. *Left panel: Sensitivity of detection with smaller CNN*. This is the sensitivity of the shallower classifier, shown in Fig. 5, as a function of SNR on the test set at a fixed false-alarm rate. Note that the sensitivity was measured with the same classifier after training once over the entire range of SNR, i.e., without specifically retraining it for each SNR. This curve saturates at a sensitivity of 100% for SNR ≥ 10, i.e, signals with (SNR ≥ 10) are always detected. The single-detector false-alarm rate was tuned to be about 0.5% for this classifier. Note that the optimal matched-filter SNR is on average proportional to 12.9 ± 1.4 times the ratio of the amplitude of the signal to the standard deviation of the noise for the test set. *Right panel: Sensitivity of detection with deeper CNN*. The same as in the left panel, but now using the deeper classifier, shown in Fig. 6. This deeper DNN now leads to a slightly increased sensitivity of detection, which saturates at 100% for SNR ≥ 9, i.e., signals with SNR ≥ 9 are always detected. These results imply that `Deep Filtering` is capable of detecting signals weaker than the background noise.

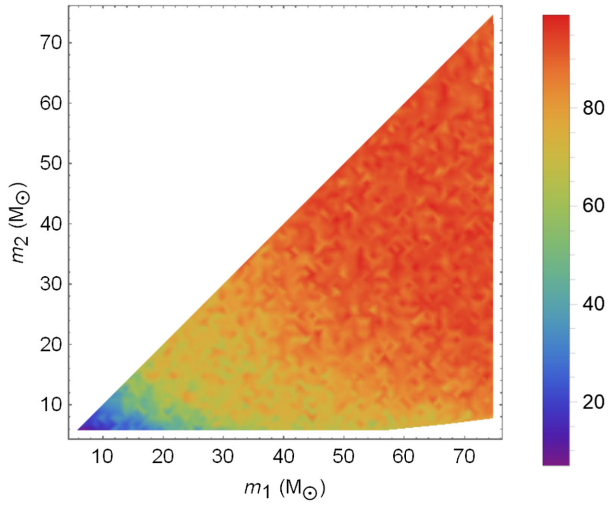

FIG. 9. *Sensitivity at* SNR $= 6$. The color indicates the sensitivity (%) of detection at each region of parameter space in the test set at a fixed SNR $= 6$ using the deeper CNN shown in Fig. 6. This indicates that for a low BBH total mass, 1 s templates may not be sufficiently long. Note that for SNR ≥ 10, however, the classifier achieved 100% sensitivity throughout the parameter space.

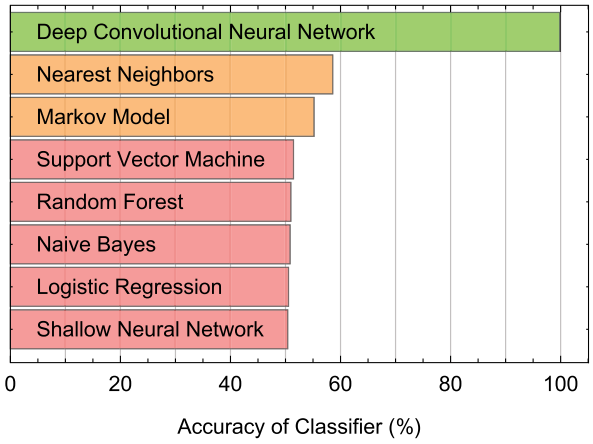

FIG. 10. *Comparison with machine learning methods for detection*. The figure compares the accuracy of different machine learning methods for detection after training each with roughly 8000 elements, half of which contained noisy whitened signals with a fixed peak power, less than the background noise, and constant total mass, with the other half being pure Gaussian noise with unit standard deviation; see Sec. A 3 for a detailed description of this comparison. An accuracy of 50% can be obtained by randomly guessing.

To provide a baseline for comparing the classification results, we trained standard implementations of all commonly used machine learning classifiers—random forest, support vector machine, k-nearest neighbors, hidden Markov model, shallow neural networks, naive Bayes, and logistic regression—along with the DNNs on a simpler training set of 8000 elements for fixed total mass and peak signal amplitude. It can be seen that unlike DNNs, none of these algorithms were able to directly handle raw noisy data even for this simple problem as shown in Fig. 10.

B. Parameter estimation

Figure 11 shows the variation in relative error against SNR for predicting the component masses of BBH GW signals from the test set, embedded in Gaussian noise, for each architecture of the DNNs shown in Figs. 5 and 6. This indicates that the predictor can measure the component masses with an error of the same order as the spacing between templates for SNR ≥ 13. These results show that the deeper predictor shown in Fig. 6 consistently outperformed matched filtering at each SNR, as shown in the right panel of Fig. 11. For SNR ≥ 50 both predictors could be trained to have a relative error less than 5%, whereas the error with matched filtering using the same templates was always greater than 11% with the given template bank. This means that, unlike matched filtering, the deep learning algorithm is able to automatically perform interpolation between the known templates to predict intermediate values. Furthermore, the largest relative errors were concentrated at lower masses, because a small variation in predicted masses leads to larger relative errors in this region.

The distribution of errors and uncertainties were estimated *empirically* in each region of the parameter space, and it was observed that the errors closely follow Gaussian normal distributions for each input for SNR (≥ 9), thus allowing easier characterization of uncertainties. Figure 12 presents a sample of the distribution of errors incurred in predicting the component masses of a BBH system with component masses (57 $M_\odot$, 33 $M_\odot$). The dependence of the error with which the component masses of each template of the test data set are recovered in each region of the parameter space is presented in Fig. 13 using the deeper CNN shown in Fig. 6 assuming a fixed SNR $= 10$).

Finally, we tested the baseline performance of a variety of common machine learning techniques including linear regression, k-nearest neighbors, shallow neural networks, Gaussian process regression, and random forest on the simpler problem of predicting the mass ratio after fixing the total mass. The results shown in Fig. 14 indicate that, unlike DNNs, they could not predict even a single parameter accurately when trained directly on time-series data.

Having quantified the performance of `Deep Filtering` for GW signals emitted by nonspinning, quasicircular BBH mergers, in the following section, the ability of the DNN-based algorithm to automatically identify new classes of signals beyond the parameter space employed for the original training and testing procedure, without retraining, is explored.

C. New classes of gravitational wave sources

In this section, we test the ability of `Deep Filtering` to detect two distinct types of signals that were *not* considered during the training stage, namely (i) moderately

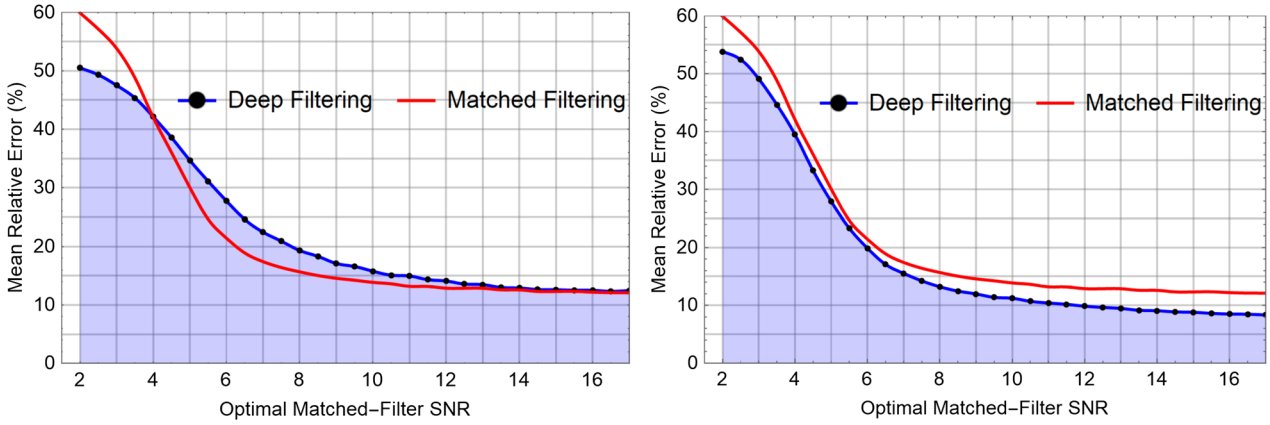

FIG. 11. *Left panel: Error in parameter estimation with smaller net*. This shows the mean percentage error of estimated masses on the test sets at each SNR using the predictor DNN with three convolution layers shown in Fig. 5. Note that the DNN was trained only once over the range of SNR and was then tested at different SNR, without retraining. A mean relative error less than 20% was obtained for SNR ≥ 8. At high SNR, the mean error saturates at around 11%. *Right panel: Error in parameter estimation with deeper net*. This shows the mean percentage error of estimated masses on the test sets at each SNR using the deeper CNN with four convolution layers shown in Fig. 6. A mean relative error less than 15% was obtained for SNR ≥ 7. At high SNR, the mean error saturates at around 7%. Note that we were able to optimize this predictor to have less than 3% error for very high SNR ($\geq$50), which demonstrates the ability of `Deep Filtering` to learn patterns connecting the templates and effectively interpolate to intermediate points in parameter space.

eccentric NR simulations (approximate eccentricity ($e_0 \lesssim 0.2$ when entering the aLIGO frequency band), that we recently generated with Einstein Toolkit—an open-source, NR software [91]—using the Blue Waters petascale supercomputer, and (ii) NR waveforms from the SXS catalog [101] that describe spin-precessing, quasicircular BBHs, with each BH having spin $\geq$0.5 oriented in random directions [101]. Sample waveforms of these GW classes are shown in Fig. 15. Since these NR simulations scale trivially with mass, the data was enlarged by rescaling the signals to have different total masses. Thereafter, the templates were whitened and added to different realizations of noise, in the same manner as before, to produce test sets.

The DNN classifiers detected all these signals with nearly the same sensitivity as the original test set, with 100% sensitivity for SNR ≥ 10. Remarkably, the predictor quantified the component masses of the eccentric simulations for SNR ≥ 12 with a mean relative error less than 20% for mass ratios $q = \{1, 2, 3, 4\}$, and less than 30% for ($q = 5.5$) respectively. For the spin-precessing systems that were tested, with SNR ≥ 12, the mean error in predicting the masses was less than 20% for $q = \{1, 3\}$, respectively.

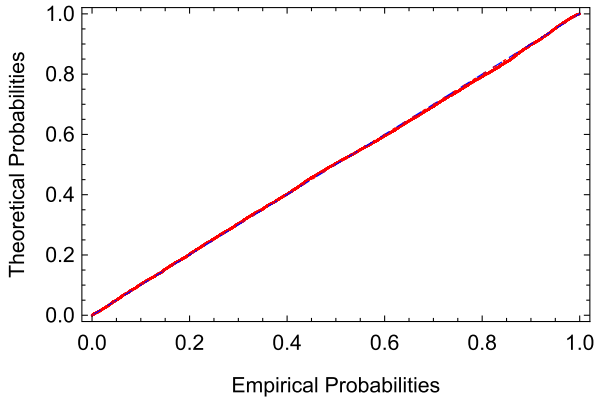

FIG. 12. *P-P plot of errors in parameter estimation* This is a P-P (probability) plot of the distribution of errors in predicting m_1 for test parameters $m_1 = 57\ M_\odot$ and $m_2 = 33\ M_\odot$, superimposed with different realizations of noise at SNR = 9. The best fit is a Gaussian normal distribution with mean $= 1.5\,M_\odot$ and standard deviation $= 4.1\,M_\odot$. The errors followed similar Gaussian distributions in other regions of the parameter space as well.

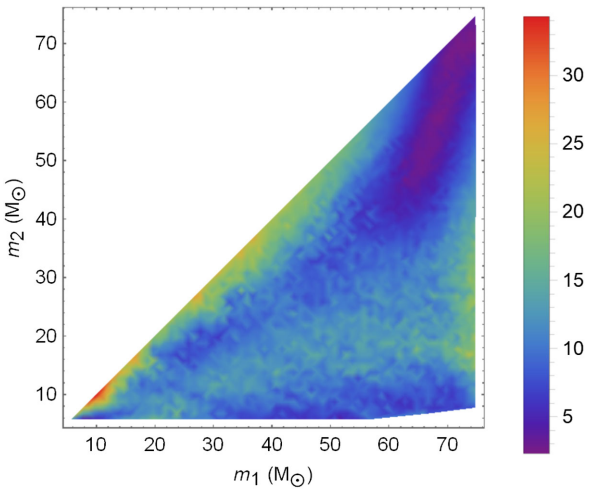

FIG. 13. *Error in parameter estimation at* SNR = 10. This figure shows the mean relative error (%) in predicting the component masses for each template in the test set at a fixed SNR = 10 using the deeper CNN shown in Fig. 6.

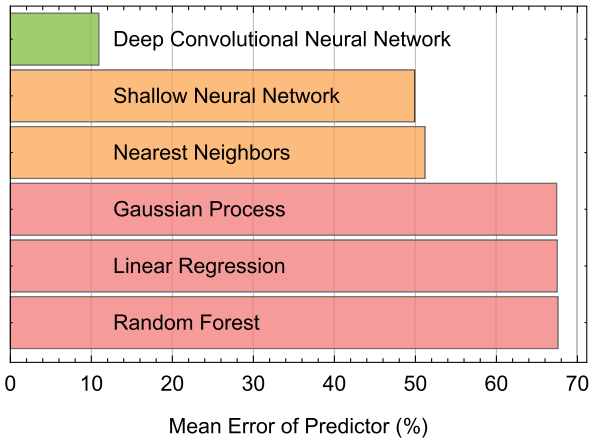

FIG. 14. *Comparison of machine learning methods for parameter estimation*. The figure shows the mean relative error obtained by various machine learning algorithms for predicting a single parameter, i.e., mass ratio, using a training set containing about 8000 signals with fixed amplitude $= 0.6$ added to white noise with unit standard deviation. Note that scaling the alternate methods to high-dimensional parameter spaces to predict multiple parameters is often difficult, unlike deep learning, which is more scalable, where neurons can be added to the final layer of neural networks to predict each parameter.

It is worth emphasizing that there exist GW algorithms that search for a wide range of *high-SNR*, *short-duration* (burst) GW signals with minimal assumptions [35], i.e., without resorting to the use of waveform templates to identify GW events. Indeed, these "burst" pipelines were used to carry out the first direct detection of GWs [3,35]. These searches do not, however, attain the same sensitivity as template-based searches for low-SNR and long-duration GW signals. Other recent advances in GW data analysis have explored the detection of spin-precessing BBH mergers using matched-filtering-based algorithms [33,37,39,102].

In view of the aforementioned considerations, let us discuss the importance of these findings. First of all, previous studies have reported that no matched-filtering algorithm has been developed to extract continuous GW signals from compact binaries on orbits with low to moderate values of eccentricity, and available algorithms to detect binaries on quasicircular orbits are suboptimal to recover these events [103]. Recent analyses have also made evident that existing GW detection algorithms are not capable of accurately detecting or reconstructing the parameters of eccentric signals [104–108].

However, when we scale the GW waveform from the NR simulation used in the left panel of Fig. 15 to describe BBH mergers with a mass ratio ($q = 5.5$) and total mass ($M \in [50\,M_\odot, 90\,M_\odot]$), and with initial eccentricity $e_0 = 0.2$ when they enter the aLIGO band, `Deep Filtering` was able to identify these signals with 100% sensitivity (for SNR ≥ 10), and recover the masses of the system with a mean relative error $\leq 30\%$ for SNR ≥ 12. To put these results in context, the right panel of Fig. 2 in Ref. [104], shows that a signal of this nature will be poorly recovered with a matched-filtering quasicircular search.

If we now consider eccentric GW signals that are relativity weak, i.e., SNR ≥ 10, this means that these events do not fall into the category of loud, short-duration, events that GW "burst" pipelines are able to recover without the use of templates. For reference, these low-latency GW pipelines, that use minimal assumptions, recovered short-duration high-SNR GW events such as GW150914, but missed long-duration low-SNR events, such as GW151226, which was identified by the matched-filtering-based GW pipeline

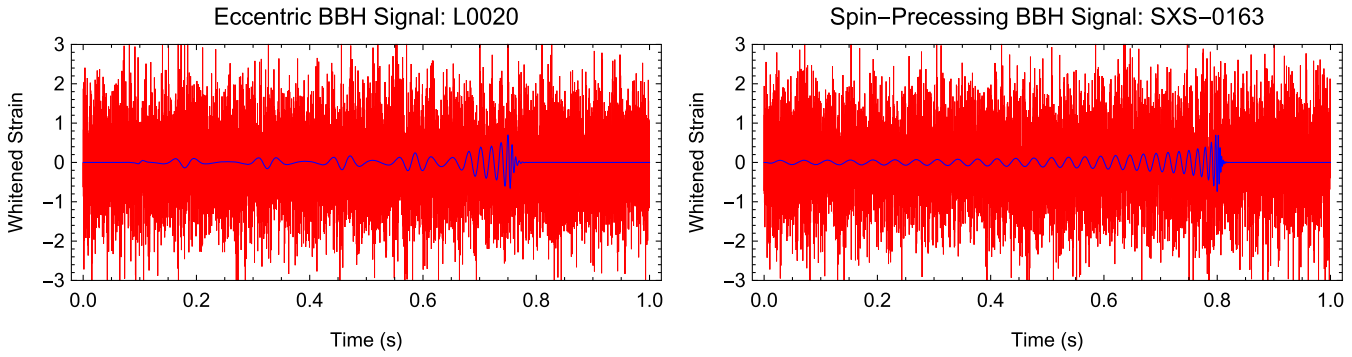

FIG. 15. *New types of signals*. Left panel: This waveform was obtained from one of our NR simulations of eccentric BBH merger that has a mass ratio of 5.5, a total mass of about 90 $M_\odot$, and an initial eccentricity $e_0 = 0.2$ when it enters the aLIGO band. The `Deep Filtering` pipeline successfully detected this signal, even when the total mass was scaled between 50 $M_\odot$ and 90 $M_\odot$, with 100% sensitivity (for SNR ≥ 10) and predicted the component masses with a mean relative error $\leq 30\%$ for SNR ≥ 12. See also Fig. 18 for more types of eccentric waveforms that were used. Right panel: One of the spin-precessing waveforms obtained from the NR simulations in the SXS catalog with component masses equal to 25 $M_\odot$ each. The individual spins are each 0.6 and oriented in unaligned directions. The DNNs also successfully detected this signal, even when the total mass was scaled between 40 $M_\odot$ and 100 $M_\odot$, with 100% sensitivity for SNR ≥ 10 and predicted the component masses with a mean relative error $\leq 20\%$ for SNR ≥ 12. See also Fig. 19 for more examples of spin-precessing waveforms which were tested.

gstLAL [4]. If we now consider that we have found similar results for a larger set of eccentric BBH signals with mass ratios ($q \leq 5.5$) and ($e_0 \leq 0.2$) ten orbits before merger, then these results imply that, in the context of stationary Gaussian noise, Deep Filtering can detect and characterize eccentric BBH mergers that are poorly recovered by matched-filtering-based quasicircular searches, and whose SNRs are low enough to not be optimally recovered by GW detection pipelines with minimal assumptions. Results in the subsequent article [51], show that this is also the case when real LIGO noise is used.

This ability to generalize to new categories of signals, without being shown any such examples, means that DNN-based pipelines may be able to increase the depth of existing GW detection algorithms without incurring additional computational expense. These results provide an incentive to develop DNNs that are also trained with data sets of eccentric and spin-precessing GWs to further improve the accuracy with which the Deep Filtering algorithms can detect and characterize these events in low latency.

D. Speed and computational cost

Furthermore, the simple classifier and predictor (in Fig. 5) are only 2 MB in size each, yet they achieve excellent results. The average time taken for evaluating them per input of 1 second duration is approximately 6.7 milliseconds, and 106 microseconds using a single CPU and GPU respectively. The deeper predictor CNN (in Fig. 6), which is about 23 MB, achieves slightly better accuracy at parameter estimation but takes about 85 milliseconds for evaluation on the CPU and 535 microseconds on the GPU, which is still orders of magnitude faster than real time. Note that the current deep learning frameworks are not well optimized for CPU evaluation. For comparison, we estimated an evaluation time of 1.1 seconds for time-domain matched filtering [46] on the same CPU (using two cores) with the same template bank of clean signals used for training; the results are shown in Fig. 16. This fast inference rate indicates that real-time analysis can be carried out with a single CPU or GPU, even with DNNs that are significantly larger and trained with template banks of millions of signals.[6] Note that CNNs can be trained on millions of inputs in a few hours using distributed training on parallel GPUs [111]. Furthermore, the input layer of the CNNs can be modified to consider inputs/templates of any duration, which will result in the computational cost scaling linearly with the input size. Therefore, even with inputs that are 1000s long, the analysis can still be carried out in real time.

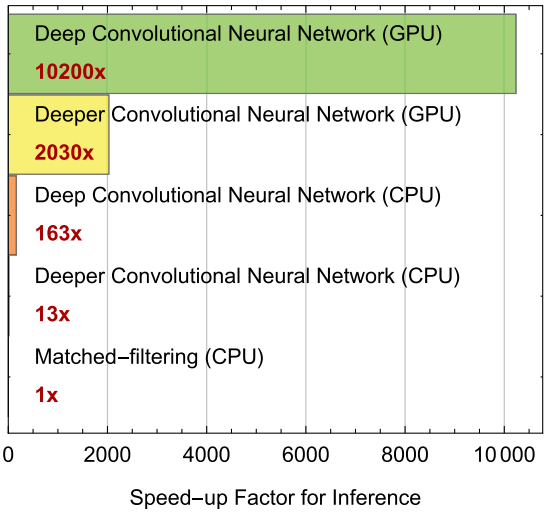

FIG. 16. *Speed-up of analysis*. The DNN-based pipeline is many orders of magnitude faster compared to matched filtering (cross-correlation or convolution) against the same template bank of waveforms (tested on batches of inputs using both cores of an Intel Core i7-6500U CPU and an inexpensive NVIDIA GeForce GTX 1080 GPU for a fairer comparison). Note that the evaluation time of a DNN is constant regardless of the size of training data, whereas the time taken for matched filtering is proportional to the number of templates being considered, i.e., exponentially proportional to the number of parameters. Therefore, the speed-up of Deep Filtering would be higher in practice, especially when considering larger template banks over a higher-dimensional parameter space.

For applying the Deep Filtering method to a multidetector scenario, one can directly apply the DNNs pretrained for single-detector inference separately to each detector and check for coincident detections with similar parameter estimates. Enforcing coincident detections would decrease the false-alarm probability, from about 0.59% to about 0.003%. Once the Deep Filtering pipeline detects a signal then traditional matched filtering may be applied with a select few templates around the estimated parameters to cross-validate the event and estimate the confidence measure. Since only a few templates need to be used with this strategy, existing challenges to extend matched filtering for higher-dimensional GW searches may thus be overcome, allowing real-time analysis with minimal computational resources.

V. DISCUSSION

It was found that the DNN architecture is resilient to the nature of the detectors' PSD. The best-performing architecture was the same when using Gaussian noise without whitening the signals i.e., a flat PSD, and when using signals whitened with aLIGO's design sensitivity. By incorporating examples of transient detector noise in the

[6]For example, a state-of-the-art CNN for image recognition [109,110] has hundreds of layers (61 MB in size) and is trained with over millions of examples to recognize thousands of different categories of objects. This CNN can process very large inputs, each having dimensions $224 \times 224 \times 3$, using a single GPU with a mean time of 6.5 milliseconds per input.

training set, the DNNs can also be taught to automatically ignore or classify glitches. While only simple DNNs were explored in this first study, our results show that deeper DNNs improve the accuracy of interpolation between GW templates for prediction as well as the sensitivity at low SNR, while retaining real-time performance. Even though the analysis presented in this article was carried out using Gaussian noise, the following article [51] shows that the key features of this method remain the same when using real LIGO data, and that `Deep Filtering` is able to learn from and adapt to the characteristics of LIGO noise, without changing the architecture of the DNNs.

Deep learning is known to be highly scalable, overcoming what is known as the curse of dimensionality [58,112]. This intrinsic ability of DNNs to take advantage of large data sets is a unique feature to enable simultaneous GW searches over a higher-dimensional parameter space that is beyond the reach of existing algorithms. Furthermore, DNNs are excellent at generalizing or extrapolating to new data. Initially, we had trained a DNN to predict only the mass ratios at a fixed total mass. Extending this to predict two-component masses only required the addition of an extra neuron to the output layer. The preliminary results in this article with simulated data indicates that the DNNs may be able to detect and reconstruct the parameters of eccentric and spin-precessing compact sources that may go unnoticed with existing aLIGO detection algorithms [103,105–107]. The extendability of this approach to predict additional parameters such as spins, eccentricities, etc., may also be explored. Note that there are also emerging techniques to estimate and quantify uncertainties in the parameter predictions of DNNs [113], which may be applied to enhance this method.

This DNN algorithm requires minimal preprocessing. In principle, aLIGO's colored noise can be superimposed into the training set of GW templates, along with observed glitches. It has been recently found that deep CNNs are capable of automatically learning to perform bandpass filtering on raw time-series inputs [114], and that they are excellent at suppressing highly nonstationary colored noise [115] especially when incorporating real-time noise characteristics [116]. This suggests that manually devised preprocessing and whitening steps may be eliminated and raw aLIGO data can be fed to DNNs. This would be particularly advantageous since it is known that Fourier transforms are the bottlenecks of aLIGO pipelines [33].

Once DNNs are trained with a given aLIGO PSD, they can be more quickly retrained, via transfer learning, during a detection campaign for recalibration in real time based on the latest characteristics of each detector's noise. Deep learning methods can also be immediately applied through distributed computing via citizen science campaigns such as Einstein@Home [117] as several open-source deep learning libraries, including MXNet, allow scalable distributed training and evaluation of neural networks simultaneously on heterogeneous devices, including smartphones and tablets. Low-power devices such as FPGAs and GPU chips dedicated for deep learning inference [118–120] may even be placed on the GW detectors to reduce data transfer issues and latency in analysis.

DNNs automatically extract and compress information by finding patterns within the training data, creating a dimensionally reduced model [121]. The fully trained DNNs are each only 2 MB (or 23 MB for the deeper model) in size yet encode all the relevant information from about 2500 GW templates (about 200 MB, before the addition of noise) used to generate the training data. Once trained, analyzing a second of data takes only milliseconds with a single CPU and microseconds with a GPU. This means that real-time GW searches could be carried out by anyone with an average laptop computer or even a smartphone, while big data sets can be processed rapidly in bulk with inexpensive hardware and software optimized for inference. The speed, power efficiency, and portability of DNNs could allow for rapid analysis of the continuous stream of data from GW detectors [51] or other astronomical facilities [57].

VI. CONCLUSION

The framework for signal processing presented in this article may be applied to enhance existing low-latency (online) GW data analysis techniques in terms of both performance and scalability and could help in enabling real-time multimessenger astrophysics observations in the future. Deep CNNs were exposed to time-series template banks of GWs, and allowed to develop their own strategies to detect and predict source parameters for a variety of GW signals embedded in highly noisy simulated data. The DNN-based method introduced in this article has been applied in Ref. [51] to build a `Deep Filtering` pipeline, trained with *real* LIGO noise, including glitches, which detect true GWs in real LIGO data and accurately estimate their parameters. These results, provide an incentive to further improve and extend `Deep Filtering` to target a larger class of GW sources, incorporating glitch classification and clustering [81], and GW denoising [79] algorithms to accelerate and broaden the scope of GW searches with aLIGO and future GW missions.

It was found that even though the DNNs were trained using a data set of GWs that describe only quasicircular, nonspinning BBH mergers, `Deep Filtering` is capable of detecting and characterizing low-SNR GW signals that describe nonspinning, eccentric BBH mergers, and quasicircular, spin-precessing BBH mergers. This provides motivation to enhance the `Deep Filtering` algorithm introduced herein to predict more parameters by including millions of spin-precessing and eccentric templates for training potentially using distributed computing methods in HPC facilities.

Employing DNNs for multimessenger astrophysics offers opportunities to harness AI computing with rapidly emerging hardware architectures and software optimized for deep learning. In addition, the use of state-of-the-art HPC facilities will continue to be used to numerically model GW sources, getting insights into the physical processes that lead to EM signatures, while also providing the means to continue using distributed computing to train DNNs.

This new approach may help in enabling real-time multimessenger observations by providing immediate alerts for follow-up after GW events. Since deep CNNs also excel at image processing, they have been applied for transient identification in large sky surveys and high-cadence surveys, respectively [57,122]. These results, combined with the analysis presented here and in Ref. [51] suggest an extensive scope for deep learning techniques to develop a new framework to further the multimessenger astrophysics program.

ACKNOWLEDGMENTS

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation, Awards No. OCI-0725070 and No. ACI-1238993 and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. The eccentric numerical relativity simulations used in this article were generated on Blue Waters with the open-source Einstein Toolkit software. We express our gratitude to Gabrielle Allen, Ed Seidel, Roland Haas, Miguel Holgado, Haris Markakis, Justin Schive, Zhizhen Zhao, other members of the NCSA Gravity Group, and Prannoy Mupparaju for their comments and interactions and to the many others who provided feedback. We thank Vlad Kindratenko for granting us unrestricted access to numerous GPUs and HPC resources in the Innovative Systems Lab at NCSA. We are grateful to NVIDIA for their generous donation of several Tesla P100 GPUs, which were used in this analysis. We also acknowledge Wolfram Research for technical assistance and for developing the software stack used to carry out this study and draft this publication.

APPENDIX: SUPPLEMENTARY MATERIALS

MATHEMATICA [123] was used for training and testing the DNNs and comparator methods as well as for data processing and visualization. Detailed documentation for all the functions mentioned in this section can be found at https://reference.wolfram.com.

1. Preparing training and testing data

We generated inspiral-merger-ringdown GW templates that describe BBH systems, with zero component spins, on quasicircular orbits with the open-source EOB model [47,48], that is implemented in LIGO's Algorithm Library [124], which is also used currently to generate template banks for aLIGO analysis pipelines. For the training set, we chose component masses from 5.75 $M_\odot$ to 75 $M_\odot$ in steps of 1 $M_\odot$ such that $m_1 > m_2$. The test set contained intermediate masses, i.e., masses from 5.25 $M_\odot$ to 75 $M_\odot$ in steps of 1 $M_\odot$. The validation set contained intermediate masses, i.e., masses from 5 $M_\odot$ to 75 $M_\odot$ in steps of 1 $M_\odot$. We deleted points with a mass ratio greater than 10. Each of these masses were rounded so that the mass ratio was a multiple of 0.1. This gave the distribution shown in Fig. 4. The EOB waveforms were generated from an initial GW frequency of 15 Hz using a sampling rate of 8192 Hz. For this study, we used the dominant waveform mode $(\ell, m) = (2, 2)$. The detectors' strain is given by [125] $h(t) = h_+(t)F_+ + h_\times(t)F_\times$, where $F_{+,\times}$ represent the antenna pattern of the detectors. As we had assumed optimally oriented systems, which satisfy $F_+ = 1, F_\times = 0$, only the h_+ component was extracted from these templates.

We selected the final 1 second of data from each of these waveforms and resampled them at 8192 Hz. Next, they were all whitened by dividing with aLIGO's design sensitivity amplitude spectral density of noise, in Fourier space. We used the "zero-detuned high-power" sensitivity of aLIGO, shown in Fig. 3, which can be downloaded at Ref. [126]. Each template is a vector of 8192 real numbers labeled with the component masses. Random examples in the final training template bank are shown in Fig. 17. Before every training session, each template was independently translated to the left by up to 0.2 seconds randomly, and padded with zeros on the right to keep the total length invariant, to produce multiple time series with the same parameters so that the positions of the peaks are not always at a fixed location. The mean position of the signal peak after translations was at 0.8 s. Batches of different realizations of Gaussian noise with standard deviations set according to the desired SNR were added to each of these templates, and the resulting time series were scaled to have zero mean and unit standard deviation, before each session. Note that the addition of noise may instead be incorporated into the training at run time and changed automatically in each round, to make the process more efficient.

2. Designing and training neural networks

For training both of our DNNs, the back-propagation algorithm was performed over multiple rounds, known as epochs, until the errors were minimized. Stochastic gradient descent with mini-batches [69] has been the traditional method used for back propagation. This technique uses an estimate of the gradient of the error over subsets of the training data in each iteration to change the weights of the DNN. The magnitude of these changes is determined by the "learning rate." Variations of this with adaptive learning rates such as ADAM have been shown to achieve better results more quickly [70], and therefore we chose this method as our learning algorithm.

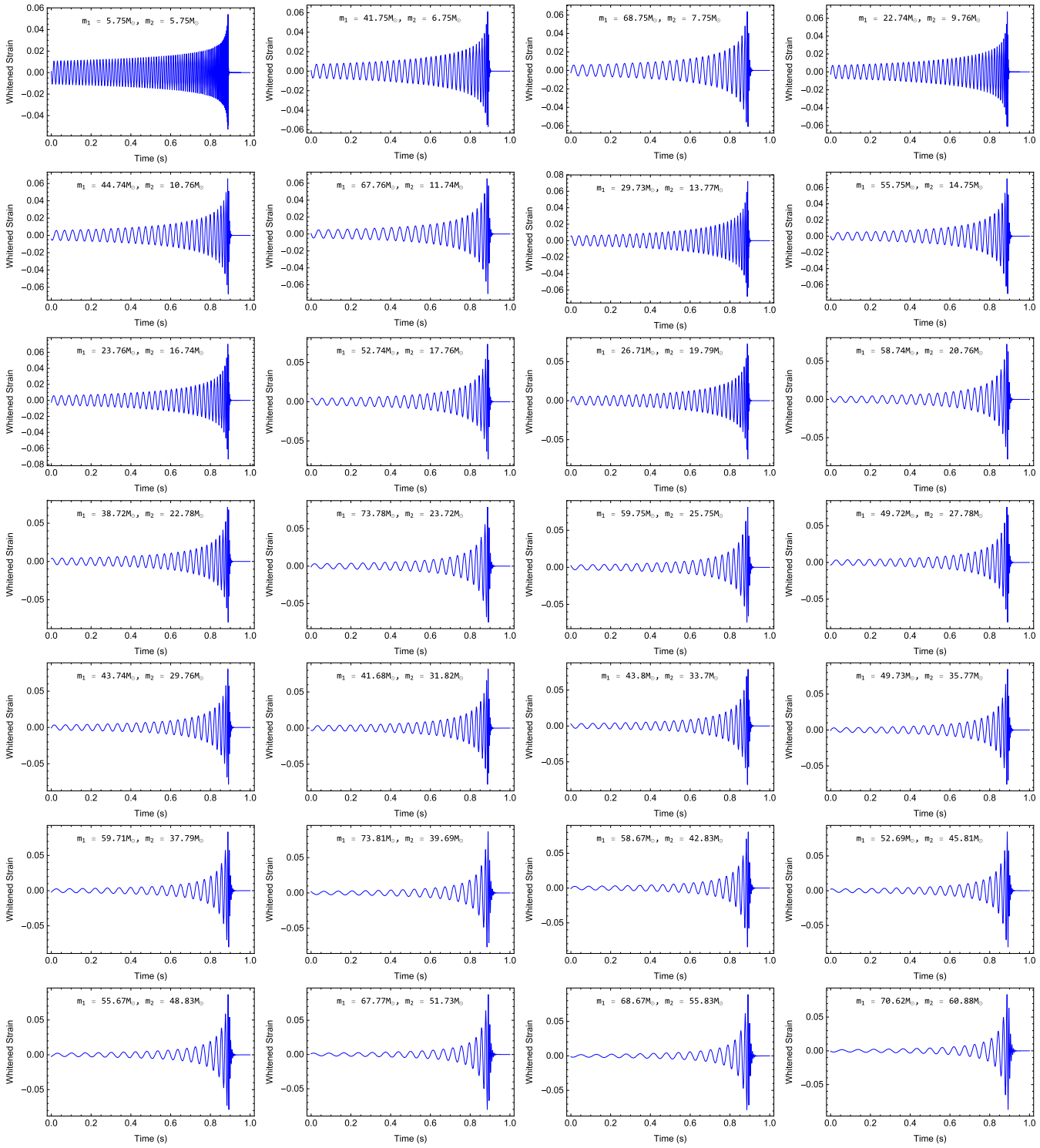

FIG. 17. *Examples of training templates*. This shows 28 randomly chosen examples of clean signal templates in our training data set, obtained with the EOB code, after whitening with the aLIGO PSD but prior to the addition of noise. The original test sets contained the same type of signals with different component masses. These signals are all produced by mergers of nonspinning, noneccentric BBHs.

We employed a random trial-and-error procedure for optimizing the hyperparameters, in which different values of hyperparameters such as stride, depth, kernel size, and dilation, were manually tuned for each layer and the performance of each DNN was interactively monitored during training as shown in Fig. 7. We did not use any zero padding for the convolution and pooling layers, since the sampling rate was high enough so that points near the edges

were irrelevant. We experimented with the ramp (ReLU) and tanh functions for the nonlinear activation layers and found that the ReLU performed the best, which is typically the case for convolutional networks [71]. A reshape layer was added at the input in order to convert vector inputs into a matrix with a single row which can be processed by the convolution layers designed for image processing. These DNNs were designed with the `NetChain` function and trained with the `NetTrain` function in the Wolfram Language. This neural network functionality was internally implemented via the open-source MXNet deep learning library [98] written in C++, which uses standard well-established methods for training. The source code is available at https://github.com/dmlc/mxnet.

When training, the TargetDevice was set to "GPU." The initial learning rate was set to 0.001 and the weights of each neuron were automatically initialized randomly according to the Xavier (Glorot) method [127] (and manually reset when needed with `NetInitialize`). The standard ADAM method was used, with the parameters $\beta_1 = 0.93$ and $\beta_2 = 0.999$ which are the exponential decay rates for the first and second moment estimates respectively. L2 regularization was set to zero. The size of the mini-batches was chosen automatically depending on the specifications of the GPU and data sets. The loss functions were selected to be the mean squared error for prediction and cross-entropy loss for the detection/classification task. The maximum number of overall batches was set to 100 000; however, the training was often stopped earlier manually when over-fitting was found to occur, i.e., the error on the validation set stopped decreasing. Most of the intensive training was done on NVIDIA Tesla P100 GPUs with version 11 of the Wolfram Language; however, a few test sessions were performed with NVIDIA Tesla K40, GTX 1080, and GT 940M GPUs.

For all sessions, the SNR for each time series was randomly sampled from the range 5–10 and multiplied with a constant factor. Initially this constant factor was set to 20, which implies that the first session of training had SNR ≥ 100. Then this constant factor in subsequent training rounds was lowered in decreasing step sizes until it was 1, i.e., the final SNR range was uniformly sampled between 5 and 15. For prediction, each time series was labeled with the component masses of the BBH system that generated the original waveforms.

For classification, we initially added batches of noise having half this size, and the desired SNR, to the clean templates and appended pure noise to get the same number of elements in total for each session. The labels were changed to "True" or "False," depending on whether a signal was present, for training the classifier. The weights of the trained predictor were extracted and used to initialize the same layers in the classifier. We initialized this classifier with the pretrained weights of the predictor and added a *softmax* layer to produce probabilities of different classes as output. The `NetDecoder` function was used within the classifier to convert the numeric vectors of probabilities to classes with labels "True" or False." Then we trained this network using the same procedure with SNR ≥ 100 and slowly decreased the SNR every round until we obtained a final SNR distribution uniformly sampled in the range 5 to 10. The fraction of noise in the training set was tuned by trial and error to about 87.5% to lower the false-alarm rate to the desired value.

Considering that all the DNNs we tested are tiny by modern standards and only a small space of hyperparameters was explored by us, we expect that higher accuracies over a wider range of parameters and types of signals can be obtained by exploring more complex configurations of DNNs, choosing more optimal hyperparameters, and using a larger set of carefully placed training templates covering the full range of GW signals.

3. Comparisons with other methods

Comparisons with other machine learning methods used built-in standard implementations of these common algorithms as documented in the Wolfram Language [123]. Open-source versions of these methods are also available in libraries such as *scikit-learn*. Optimal parameters for each model were chosen automatically by the `Classify` and `Predict` functions. The time series of 1 second duration sampled at 8192 Hz were directly used as inputs to all methods. The mean of the absolute values of the relative error on the test set was measured for prediction. For classification, the accuracy on the test set was measured using the `ClassifierMeasurements` function. The steps followed are described below. To provide a fair comparison, each method was directly given the same raw time series as inputs. Note that it may be possible to improve the performance of any machine learning method by providing hand-extracted "expert" features instead or a DNN may be used as a feature extractor for each of the alternative methods.

For comparison with different methods for prediction (parameter estimation), we used the same EOB waveforms as before but fixed the total mass to be 60 $M_\odot$ for training and testing, to predict only the mass ratio. Thus, we used 91 templates covering mass ratios from 1 to 10 in steps of 0.1 for training and 15 templates with intermediate mass ratios for testing. The size of this training data was enhanced by adding different realizations of Gaussian noise, scaled by the same total mass of 60 $M_\odot$ and labeled with the mass ratio. Then 88 different realizations of noise were added to each of the training templates to produce a total of 8008 time series for training and 264 different realizations of noise were added to each of the testing templates to obtain 3960 time series for testing. A validation set of 2640 elements was also produced by adding another 176 different realizations of noise to each of the testing templates. The noise was chosen to have a Gaussian distribution and a

unit standard deviation. The amplitudes of all the signals were set to 0.6 and added to the noise to create the inputs for training and testing. The inputs were then normalized to have unit standard deviation and zero mean. Smaller data sets were used because the other methods are not implemented efficiently on GPUs, unlike DNNs, and therefore the training procedure was done on a high-performance CPU machine over several days. The predictor DNN was initialized randomly and retrained with this new data set for comparison. The mean relative errors we obtained with the different methods, shown in Fig. 14, are as follows: DNN—10.92%; shallow neural networks—49.93%; Gaussian process regression—67.43%; linear regression—67.50%; random forest—67.59%; k-nearest neighbors—51.18%.

For comparing the classifiers, we trained all the methods from scratch with the same set of templates, labeled "True," appended with 50% pure Gaussian noise labeled "False," comprised of 7662 time series. The testing and validation sets contained 3516 elements, each having 50% noise. The ratio of the amplitude of the signals to the standard deviation of the noise was fixed at 0.6. The classifier DNN was also initialized randomly and retrained with this data set. Note that this implies that DNNs can be successfully trained with much smaller data sets for the detection task alone at fixed high SNR. A larger number of templates were used in our analysis in order to perform parameter estimation, which is a harder problem than classification since the parameter space is continuous as opposed to a finite discrete set of classes, and to improve the performance at low SNR. The accuracy of these methods obtained on the test set, shown in Fig. 14, are as follows: DNN—99.81%; shallow neural networks—50.40; support vector machine—51.45%; logistic regression—50.55; random forest—50.97%; k-nearest neighbors—58.58%; naive Bayes—50.84%; hidden Markov model—55.19%. For both classification and prediction, shallow neural networks refer to fully connected neural networks with less than three hidden layers.

To measure the speed of evaluation (inference) of the DNNs on new inputs, the `AbsoluteTiming` function was used to measure the total time for the evaluation of each method over batches of 1000 inputs and the average time per input was computed. The benchmarks were all run with MATHEMATICA 11, which uses the Intel MKL library, on a Windows 10 64-bit machine with an Intel Skylake Core i7-6500U CPU. A desktop-grade NVIDIA GTX 1080 GPU was used for measuring the speed-up of analysis with DNNs, instead of the expensive high-performance GPUs that were used for training, since this has a price closer to a desktop CPU and thus provides a fair comparison against the performance of the CPU at similar costs. The measured times averaged over batches of inputs were 6.67 milliseconds and 106 microseconds per input (vector of length 8192) with the CPU and GPU respectively.

We used a standard implementation of time-domain matched filtering (similar to Ref. [46]) with the same template bank of clean signals by computing the cross-correlation (which was the same as the convolution with time-reversed templates) of an input of 1 second duration from the test set against the same templates in the training set using the same sampling rate. The parameters were estimated to be those of the best matching template. The threshold of single-detector matched-filter SNR required for detection was tuned to be about 5.3 to have a false-alarm rate similar to the classifier's. Since the peak of the signal was shifted within 0.2 seconds while training the DNNs, we also assumed the same window for the location of the peak for matched filtering by truncating the templates to have a 0.8 second duration (removing the part near the edges, which does not contain the signal). The `ListCorrelate` function, which uses the Intel MKL library, was used to perform this computation and the mean time per input on the same CPU (optimized to use both cores) over 1000 parallelized runs was measured to be about 1.1 seconds.

For timing the larger DNN deployed for the Image Identification Project [109], we used the `NetModel` function to obtain the pretrained model in version 11 of the Wolfram Language. This DNN is based on the Inception V2 model originally proposed in Ref. [110]. The timing of inference was done using an NVIDIA Tesla P100 GPU on a batch of 1000 inputs, each being an RGB image having a resolution of 224×224 pixels. The average time per input for batches of 1000 images was measured to be 6.54 milliseconds. This DNN is publicly available for download via Ref. [128].

4. Measuring accuracy and errors

For computing the sensitivity at each SNR, we applied the DNN classifier to time-series inputs containing the true signals, produced by adding 10 different realizations of noise to each of the clean templates (about 2500) in the test set and computed the ratio of detected signals to the total number of inputs. The SNR was varied from 2 to 17 in steps of 0.5. Therefore, about 0.8 million seconds of data, in total, sampled at 8192 Hz was used for constructing the sensitivity plots. The false-alarm rate was measure by applying the classifier to 100 000 realizations of Gaussian noise with duration 1 s and sampling rate 8192 Hz.

For measuring the mean relative errors in prediction at each SNR, we applied the predictor on time-series inputs produced by adding 10 different realizations of noise to each of the clean templates in the test set and averaged this at each SNR. The absolute value of the relative errors in predicting each component mass was averaged. The SNR was varied from 2 to 17 in steps of 0.5. Thus, about 0.8 million seconds of data in total, sampled at 8192 Hz, was also used for preparing each of these plots.

The distribution of errors for randomly chosen templates in the test set was measured after the addition of 1000 different realizations of noise to each of them at fixed SNR. We verified that the errors closely match

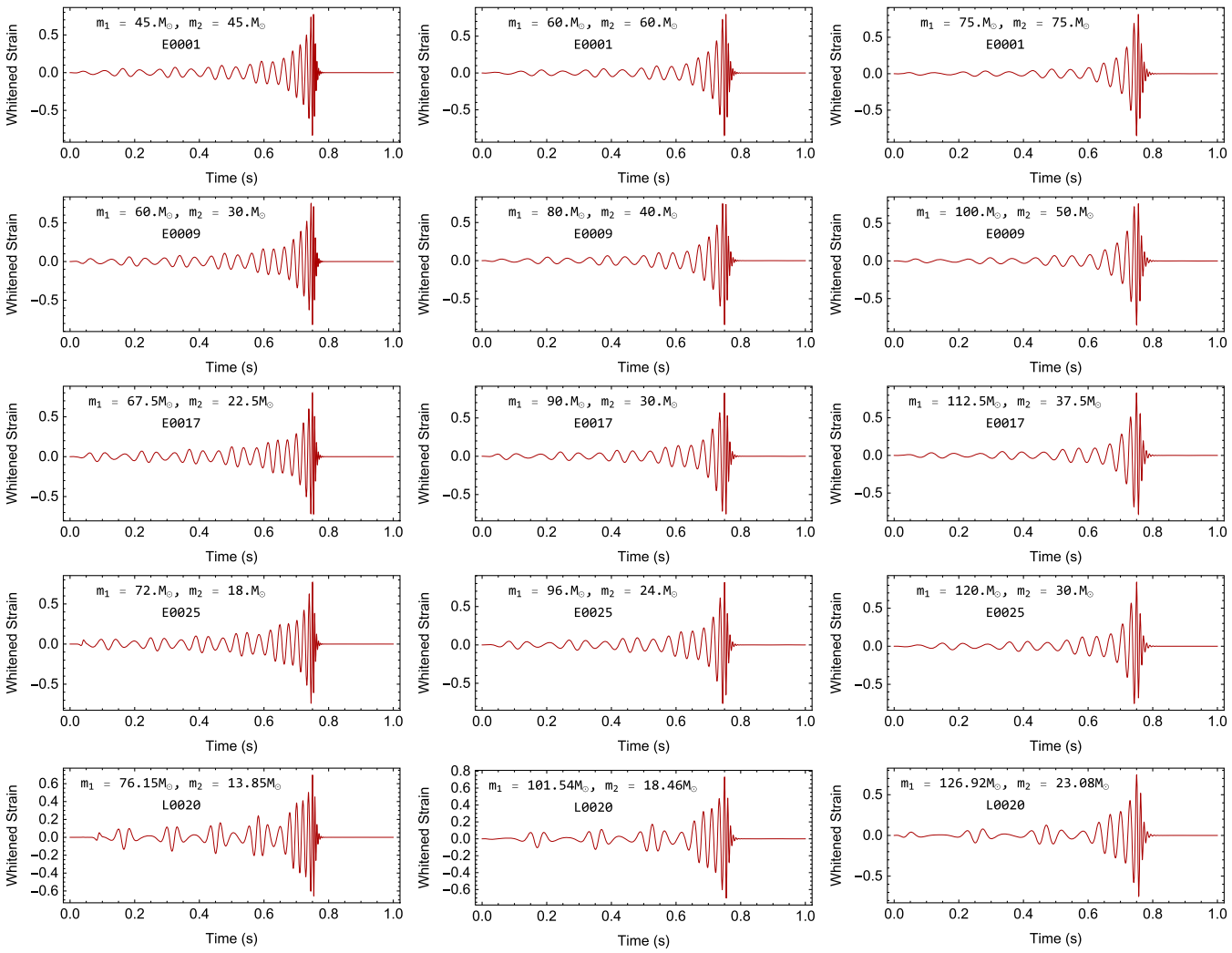

FIG. 18. *Examples of eccentric signals*. These are the five simulations of eccentric BBH systems with different mass ratios, which we used to test the DNNs. Each of these signals were scaled to have different total masses by stretching in time to enlarge the size of the test set. They were produced with the open-source Einstein Toolkit software on the Blue Waters supercomputer. The initial conditions were chosen such that the eccentricity was 0.1 for the first four simulations and 0.2 for the final simulation for each system as it enters the aLIGO band.

Gaussian distributions using standard probability-probability (P-P) plots at randomly chosen points in the parameter space for SNR ≥ 9. For lower SNR, the distribution was slightly skewed. The best-fitting parameters of the normal distribution were automatically chosen by the `ProbabilityPlot` function and a random sample is shown in Fig. 12.

Although this analysis was originally intended for quasicircular, nonspinning binaries, we tested the performance of the DNNs on new classes of signals without extra training. The eccentric NR signals used in this study were generated using the open-source Einstein Toolkit software [129], on the Blue Waters supercomputer. For reproducibility purposes, we are including the metadata information of the simulations we used as auxiliary supplementary material. A large catalog of eccentric NR simulations will be presented in a subsequent publication. The waveforms extracted from the Einstein Toolkit data are rendered in natural units of M, and describe BBH systems with a total mass of 1 $M_\odot$. All four waveforms we used for this article are also attached and have the identifiers E0001, E0009, E0017, E0025, and L0020 for mass ratios 1, 2, 3, 4, and 5.5 respectively. The first four simulations had an eccentricity of 0.1 and the last had 0.2 when entering the LIGO band. The parameter files that we used for our eccentric simulations were modified versions of the open-source parameter file [130]. We had used resolutions of 32, 36, and 40 grid points across each BH matching resolution used in typical production simulations. We verified that these exhibited strong convergent behavior. Full simulation data will be provided upon request.

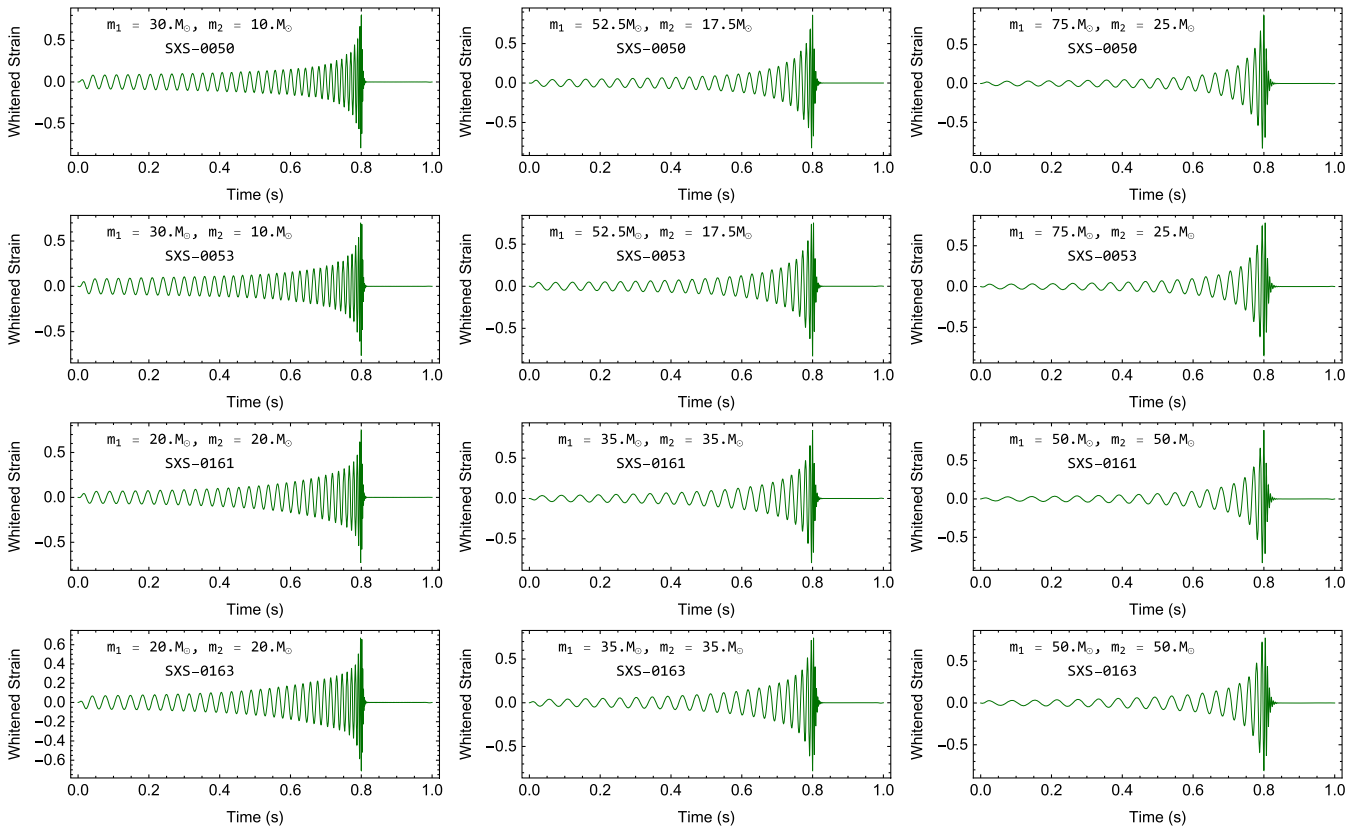

FIG. 19. *Examples of spin-precessing signals*. These are the four GW simulations of spin-precessing BBHs from the SXS catalog, which we used to test the DNNs. Each of these signals was also scaled to have different total masses by stretching in time to enlarge the size of the test set. The individual spins of each system were higher than 0.5, and the orientation was in arbitrary directions, i.e., the spins were not aligned or antialigned. Full details of these simulations are available at https://www.black-holes.org/waveforms/catalog.php.

As discussed before, since GW templates scale trivially with mass, more templates were produced by scaling the eccentric NR waveforms to have total masses between 85 $M_\odot$ and a maximum mass depending on the mass ratio, to ensure that the simulations provided enough data for about 1 second and the component masses lie between 5 $M_\odot$ and 75 $M_\odot$, which is the range of component masses that the predictor was originally trained for. The maximum mass for each mass ratio was set so that the largest component mass was 75 $M_\odot$. We again used the real (+) component of the dominant $(\ell, m) = (2, 2)$ mode. A random template at each mass ratio is shown in Fig. 18.

The mean relative errors were predicted on a test set obtained after adding 10 000 different realizations of noise with SNR $= 10$ and 12 for every value of total mass for each mass ratio. The mean of the absolute values of the relative errors was calculated. We separately analyzed the prediction rates for each signal, since they differed by a large rate for different mass ratios. A step size of 0.5 $M_\odot$ was used to vary the total mass by stretching E0001, E0009, E0017, E0025, and L0020. The ranges used for total masses were different for different mass ratios according to the constraint that individual masses should lie between 5 $M_\odot$ and 75 $M_\odot$. For measuring the sensitivity of detection, we used the combined data set of all these templates used for prediction, at a fixed SNR of 10, each added to 10 000 realizations of noise to create a single test set.

For testing with spin-precessing systems, we used waveforms extracted from four NR simulations that describe quasicircular, spin-precessing BBH systems obtained from the publicly available catalog of simulations performed by the SXS Collaboration [101], hosted at https://www.black-holes.org/waveforms/catalog.php. The full data and parameters for each simulation can be found at this website. The BBH configurations we selected, labeled SXS:BBH:0050, SXS:BBH:0053, SXS:BBH:0161, SXS:BBH:0163, represent compact binaries with the largest values of spin (larger than 0.5 each) oriented in arbitrary directions, so as to exacerbate the effect of spin precession, and serve as strong tests of the robustness of the detection and parameter reconstruction algorithms. Their mass ratios were 3, 3, 1, and 1 respectively.

The spin-precessing NR waveforms we selected correspond to the highest quality waveforms for each simulation. This was found in the highest-resolution runs (labeled with highest "Lev"). The second-order extrapolation to infinite radius (N2-Extrapolated file) within the "rhOverM_Asymptotic_GeometricUnits.h5" files was

selected. Since we assumed optimally oriented systems for this study, we chose the + component of the dominant waveform mode, $(\ell, m) = (2, 2)$, which captures the signatures of spin precession. The total mass was again scaled in the same manner, with the constraints on the range of component masses and that the signal should last 1 second. These NR simulations were longer than the eccentric ones; therefore, the lower limit of total mass was set to 60 $M_\odot$. The upper limit was chosen such that, for each mass ratio, the largest component mass was 75 $M_\odot$. A randomly chosen template for each system is shown in Fig. 19.

The mean of the absolute value of the relative errors in predicting component masses was computed separately for the different mass ratios (1 and 3), in the same manner as for spin-pressing systems. For each signal, 10 000 sets of different noise realizations were added at each value of total mass, which was varied in steps of 0.5 $M_\odot$. The sensitivity of detection was measured on the combined set of signals obtained in the same manner as before.

[1] LIGO Scientific and Virgo Collaborations, Phys. Rev. Lett. **116**, 131103 (2016).
[2] J. Aasi *et al.* (LIGO Scientific Collaboration), Classical Quantum Gravity **32**, 7 (2015).
[3] B. P. Abbott *et al.*, Phys. Rev. Lett. **116**, 061102 (2016).
[4] B. P. Abbott *et al.*, Phys. Rev. Lett. **116**, 241103 (2016).
[5] B. P. Abbott *et al.*, Phys. Rev. Lett. **118**, 221101 (2017).
[6] B. P. Abbott *et al.* (LIGO Scientific and Virgo Collaborations), Astrophys. J. **851**, L35 (2017).
[7] B. P. Abbott *et al.*, Phys. Rev. Lett. **119**, 141101 (2017).
[8] F. Acernese *et al.*, Classical Quantum Gravity **32**, 024001 (2015).
[9] B. P. Abbott *et al.*, Phys. Rev. Lett. **119**, 161101 (2017).
[10] B. P. Abbott *et al.*, Living Rev. Relativity **19**, 1 (2016).
[11] L. P. Singer, L. R. Price, B. Farr, A. L. Urban, C. Pankow, S. Vitale, J. Veitch, W. M. Farr, C. Hanna, K. Cannon, T. Downes, P. Graff, C.-J. Haster, I. Mandel, T. Sidery, and A. Vecchio, Astrophys. J. **795**, 105 (2014).
[12] B. P. Abbott *et al.*, Astrophys. J. **848**, L12 (2017).
[13] B. P. Abbott *et al.*, Astrophys. J. Lett. **848**, L13 (2017).
[14] D. Eichler, M. Livio, T. Piran, and D. N. Schramm, Nature (London) **340**, 126 (1989).
[15] B. Paczynski, Astrophys. J. Lett. **308**, L43 (1986).
[16] R. Narayan, B. Paczynski, and T. Piran, Astrophys. J. Lett. **395**, L83 (1992).
[17] C. S. Kochanek and T. Piran, Astrophys. J. **417**, L17 (1993).
[18] C. D. Ott, Classical Quantum Gravity **26**, 063001 (2009).
[19] E. S. Phinney, arXiv:0903.0098.
[20] T. Abbott *et al.* (Dark Energy Survey Collaboration), Mon. Not. R. Astron. Soc. **460**, 1270 (2016).
[21] A. A. Abdo *et al.*, Astrophys. J. Suppl. Ser. **208**, 17 (2013).
[22] J. A. Tyson, Proc. SPIE Int. Soc. Opt. Eng. **4836**, 10 (2002).
[23] L. Amendola *et al.*, Living Rev. Relativity **16**, 6 (2013).
[24] N. Gehrels and D. Spergel (on behalf of the WFIRST SDT and Project), J. Phys. Conf. Ser. **610**, 012007 (2015).
[25] S. Adrián-Martínez *et al.* (Antares Collaboration, IceCube Collaboration, LIGO Scientific Collaboration, and Virgo Collaboration), Phys. Rev. D **93**, 122010 (2016).
[26] B. P. Abbott *et al.*, Phys. Rev. D **94**, 064035 (2016).
[27] J. Healy *et al.*, arXiv:1712.05836.
[28] P. Mösta, B. C. Mundim, J. A. Faber, R. Haas, S. C. Noble, T. Bode, F. Löffler, C. D. Ott, C. Reisswig, and E. Schnetter, Classical Quantum Gravity **31**, 015005 (2014).
[29] R. Haas, C. D. Ott, B. Szilagyi, J. D. Kaplan, J. Lippuner, M. A. Scheel, K. Barkett, C. D. Muhlberger, T. Dietrich, M. D. Duez, F. Foucart, H. P. Pfeiffer, L. E. Kidder, and S. A. Teukolsky, Phys. Rev. D **93**, 124062 (2016).
[30] E. Abdikamalov, S. Gossan, A. M. DeMaio, and C. D. Ott, Phys. Rev. D **90**, 044001 (2014).
[31] L. E. Kidder, S. E. Field, F. Foucart, E. Schnetter, S. A. Teukolsky, A. Bohn, N. Deppe, P. Diener, F. Hébert, J. Lippuner, J. Miller, C. D. Ott, M. A. Scheel, and T. Vincent, J. Comput. Phys. **335**, 84 (2017).
[32] S. Nissanke, M. Kasliwal, and A. Georgieva, Astrophys. J. **767**, 124 (2013).
[33] S. A. Usman *et al.*, Classical Quantum Gravity **33**, 215004 (2016).
[34] K. Cannon, R. Cariou, A. Chapman, M. Crispin-Ortuzar, N. Fotopoulos, M. Frei, C. Hanna, E. Kara, D. Keppel, L. Liao, S. Privitera, A. Searle, L. Singer, and A. Weinstein, Astrophys. J. **748**, 136 (2012).
[35] B. P. Abbott *et al.*, Phys. Rev. D **93**, 122004 (2016).
[36] N. J. Cornish and T. B. Littenberg, Classical Quantum Gravity **32**, 135012 (2015).
[37] R. Smith, S. E. Field, K. Blackburn, C.-J. Haster, M. Pürrer, V. Raymond, and P. Schmidt, Phys. Rev. D **94**, 044031 (2016).
[38] J. Veitch *et al.*, Phys. Rev. D **91**, 042003 (2015).
[39] I. Harry, S. Privitera, A. Bohé, and A. Buonanno, Phys. Rev. D **94**, 024012 (2016).
[40] T. B. Littenberg, B. Farr, S. Coughlin, and V. Kalogera, Astrophys. J. **820**, 7 (2016).
[41] N. Indik, H. Fehrmann, F. Harke, B. Krishnan, and A. B. Nielsen, arXiv:1712.07869.
[42] E. A. Huerta, R. Haas, E. Fajardo, D. S. Katz, S. Anderson, P. Couvares, J. Willis, T. Bouvet, J. Enos, W. T. C. Kramer, H. W. Leong, and D. Wheeler, arXiv:1709.08767.
[43] D. Weitzel, B. Bockelman, D. A. Brown, P. Couvares, F. Würthwein, and E. Fajardo Hernandez, arXiv:1705.06202.
[44] Y. Lecun, Y. Bengio, and G. Hinton, Nature (London) **521**, 436 (2015).

[45] Y. LeCun and Y. Bengio, in *The Handbook of Brain Theory and Neural Networks*, edited by M. A. Arbib (MIT, Cambridge, MA, 1998), p. 255.
[46] C. Messick *et al.*, Phys. Rev. D **95**, 042001 (2017).
[47] A. Taracchini, A. Buonanno, Y. Pan, T. Hinderer, M. Boyle, D. A. Hemberger, L. E. Kidder, G. Lovelace, A. H. Mroué, H. P. Pfeiffer, M. A. Scheel, B. Szilágyi, N. W. Taylor, and A. Zenginoglu, Phys. Rev. D **89**, 061502 (2014).
[48] A. Bohé *et al.*, Phys. Rev. D **95**, 044028 (2017).
[49] K. Belczynski, D. E. Holz, T. Bulik, and R. O'Shaughnessy, Nature (London) **534**, 512 (2016).
[50] D. Shoemaker, Advanced LIGO anticipated sensitivity curves, LIGO Document T0900288-v3, https://dcc.ligo.org/LIGO-T0900288/public.
[51] D. George and E. A. Huerta, Phys. Lett. B **778**, 64 (2018).
[52] D. J. C. Mackay, *Information Theory, Inference and Learning Algorithms* (Cambridge University Press, Cambridge, England, 2003).
[53] C. J. Moore, C. P. L. Berry, A. J. K. Chua, and J. R. Gair, Phys. Rev. D **93**, 064001 (2016).
[54] C. J. Moore and J. R. Gair, Phys. Rev. Lett. **113**, 251101 (2014).
[55] M. Zevin, S. Coughlin, S. Bahaadini, E. Besler, N. Rohani, S. Allen, M. Cabero, K. Crowston, A. Katsaggelos, S. Larson, T. K. Lee, C. Lintott, T. Littenberg, A. Lundgren, C. Oesterlund, J. Smith, L. Trouille, and V. Kalogera, arXiv:1611.04596.
[56] D. George, H. Shen, and E. A. Huerta, arXiv:1706.07446.
[57] N. Sedaghat and A. Mahabal, arXiv:1710.01422.
[58] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT, Cambridge, MA, 2016).
[59] J. Schmidhuber, Neural Netw. **61**, 85 (2015).
[60] Y. Bengio, A. Courville, and P. Vincent, IEEE Trans. Pattern Anal. Mach. Intell. **35**, 1798 (2013).
[61] M. Nielsen, Neural Networks and Deep Learning (2016), http://neuralnetworksanddeeplearning.com/.
[62] D. Graupe, *Principles of Artificial Neural Networks*, 3rd ed. (World Scientific, Singapore, 2013).
[63] F. Rosenblatt, Psychol. Rev. **65**, 386 (1958).
[64] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry* (MIT, Cambridge, MA, 1969).
[65] K. Hornik, M. Stinchcombe, and H. White, Neural Netw. **2**, 359 (1989).
[66] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. Lecun, What is the best multi-stage architecture for object recognition?, in *2009 IEEE 12th International Conference on Computer Vision, Kyoto, 2009* (IEEE, New York, 2009), p. 2146.
[67] Wikimedia Commons: Artifical Neural Network, https://upload.wikimedia.org/wikipedia/commons/thumb/e/e4/Artificial_neural_network.svg/2000px-Artificial_neural_network.svg.png.
[68] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, in *Neural Networks: Tricks of the Trade* (Springer-Verlag, Berlin, 1998), p. 9.
[69] S. Ruder, arXiv:1609.04747.
[70] D. P. Kingma and J. Ba, arXiv:1412.6980.
[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Advances in Neural Information Processing Systems 25*, edited by P. Bartlett (Curran Associates, Inc., Red Hook, NY, 2012), p. 1097.
[72] F. Yu and V. Koltun, in *ICLR* (2016).
[73] B. J. Owen and B. S. Sathyaprakash, Phys. Rev. D **60**, 022002 (1999).
[74] P. Graff, F. Feroz, M. P. Hobson, and A. Lasenby, Mon. Not. R. Astron. Soc. **421**, 169 (2012).
[75] N. Mukund, S. Abraham, S. Kandhasamy, S. Mitra, and N. S. Philip, Phys. Rev. D **95**, 104059 (2017).
[76] J. Powell, A. Torres-Forné, R. Lynch, D. Trifirò, E. Cuoco, M. Cavaglià, I. S. Heng, and J. A. Font, Classical Quantum Gravity **34**, 034002 (2017).
[77] J. Powell, D. Trifirò, E. Cuoco, I. S. Heng, and M. Cavaglià, Classical Quantum Gravity **32**, 215012 (2015).
[78] S. Bahaadini, N. Rohani, S. Coughlin, M. Zevin, V. Kalogera, and A. K. Katsaggelos, arXiv:1705.00034.
[79] H. Shen, D. George, E. A. Huerta, and Z. Zhao, arXiv:1711.09919.
[80] M. Zevin, S. Coughlin, S. Bahaadini, E. Besler, N. Rohani, S. Allen, M. Cabero, K. Crowston, A. Katsaggelos, S. Larson, T. K. Lee, C. Lintott, T. Littenberg, A. Lundgren, C. Oesterlund, J. Smith, L. Trouille, and V. Kalogera, Classical Quantum Gravity **34**, 064003 (2017).
[81] D. George, H. Shen, and E. A. Huerta, arXiv:1711.07468.
[82] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Curran Associates, Inc., Red Hook, NY, 2015).
[83] K. Simonyan and A. Zisserman, arXiv:1409.1556.
[84] K. He, X. Zhang, S. Ren, and J. Sun, arXiv:1512.03385.
[85] B. J. Owen and B. S. Sathyaprakash, Phys. Rev. D **60**, 022002 (1999).
[86] T. J. O'Shea, J. Corgan, and T. C. Clancy, Convolutional radio modulation recognition networks, in *Engineering Applications of Neural Networks: 17th International Conference, EANN 2016, Aberdeen, UK, September 2–5, 2016, Proceedings*, edited by C. Jayne and L. Iliadis (Springer, New York, 2016), p. 213.
[87] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, Time series classification using multi-channels deep convolutional neural networks, in *Web-Age Information Management: 15th International Conference, WAIM 2014, Macau, China, June 16–18, 2014. Proceedings*, edited by F. Li, G. Li, S.-w. Hwang, B. Yao, and Z. Zhang (Springer, New York, 2014), p. 298.
[88] M. Pürrer, Phys. Rev. D **93**, 064041 (2016).
[89] K. Belczynski, S. Repetto, D. Holz, R. O'Shaughnessy, T. Bulik, E. Berti, C. Fryer, and M. Dominik, Astrophys. J. **819**, 108 (2016).
[90] B. P. Abbott *et al.*, Phys. Rev. X **6**, 041015 (2016).
[91] F. Löffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, R. Haas, I. Hinder, B. C. Mundim, C. D. Ott, E. Schnetter, G. Allen, M. Campanelli, and P. Laguna, Classical Quantum Gravity **29**, 115001 (2012).
[92] A. Torres-Forné, A. Marquina, J. A. Font, and J. M. Ibáñez, Phys. Rev. D **94**, 124040 (2016).
[93] A. H. Mroué, M. A. Scheel, B. Szilágyi, H. P. Pfeiffer, M. Boyle, D. A. Hemberger, L. E. Kidder, G. Lovelace, S. Ossokine, N. W. Taylor, A. Zenginoğlu, L. T. Buchman,

T. Chu, E. Foley, M. Giesler, R. Owen, and S. A. Teukolsky, Phys. Rev. Lett. **111**, 241104 (2013).
[94] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, Efficient backprop, in *Neural Networks: Tricks of the Trade*, edited by G. B. Orr and K.-R. Müller (Springer-Verlag, Berlin, 1998), p. 9.
[95] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Proc. IEEE **86**, 2278 (1998).
[96] S. Ioffe and C. Szegedy, arXiv:1502.03167.
[97] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, J. Mach. Learn. Res. **15**, 1929 (2014).
[98] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, arXiv:1512.01274.
[99] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, arXiv:1410.0759.
[100] A. H. Nitz, Classical Quantum Gravity **35**, 035016 (2018).
[101] T. Chu, H. Fong, P. Kumar, H. P. Pfeiffer, M. Boyle, D. A. Hemberger, L. E. Kidder, M. A. Scheel, and B. Szilagyi, Classical Quantum Gravity **33**, 165001 (2016).
[102] S. Privitera, S. R. P. Mohapatra, P. Ajith, K. Cannon, N. Fotopoulos, M. A. Frei, C. Hanna, A. J. Weinstein, and J. T. Whelan, Phys. Rev. D **89**, 024003 (2014).
[103] V. Tiwari, S. Klimenko, N. Christensen, E. A. Huerta, S. R. P. Mohapatra, A. Gopakumar, M. Haney, P. Ajith, S. T. McWilliams, G. Vedovato, M. Drago, F. Salemi, G. A. Prodi, C. Lazzaro, S. Tiwari, G. Mitselmakher, and F. Da Silva, Phys. Rev. D **93**, 043007 (2016).
[104] E. A. Huerta, C. J. Moore, P. Kumar, D. George, A. J. K. Chua, R. Haas, E. Wessel, D. Johnson, D. Glennon, A. Rebei, A. M. Holgado, J. R. Gair, and H. P. Pfeiffer, Phys. Rev. D **97**, 024031 (2018).
[105] E. A. Huerta, P. Kumar, B. Agarwal, D. George, H.-Y. Schive, H. P. Pfeiffer, R. Haas, W. Ren, T. Chu, M. Boyle, D. A. Hemberger, L. E. Kidder, M. A. Scheel, and B. Szilagyi, Phys. Rev. D **95**, 024038 (2017).
[106] E. A. Huerta, P. Kumar, S. T. McWilliams, R. O'Shaughnessy, and N. Yunes, Phys. Rev. D **90**, 084016 (2014).
[107] E. A. Huerta and D. A. Brown, Phys. Rev. D **87**, 127501 (2013).
[108] E. A. Huerta, S. T. McWilliams, J. R. Gair, and S. R. Taylor, Phys. Rev. D **92**, 063010 (2015),
[109] The Wolfram Language Image Identification Project, https://www.imageidentify.com/.
[110] S. Ioffe and C. Szegedy, *Proceedings of Machine Learning Research* 37, 448 (2015).
[111] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, arXiv:1706.02677.
[112] Y. Bengio and Y. LeCun, in *Large Scale Kernel Machines*, edited by L. Bottou, O. Chapelle, D. DeCoste, and J. Weston (MIT, Cambridge, MA 2007).
[113] L. Perreault Levasseur, Y. D. Hezaveh, and R. H. Wechsler, Astrophys. J. **850**, L7 (2017).
[114] W. Dai, C. Dai, S. Qu, J. Li, and S. Das, arXiv:1610.00087.
[115] Y. Xu, J. Du, L. R. Dai, and C. H. Lee, IEEE/ACM Transactions on Audio, Speech, and Language Processing **23**, 7 (2015).
[116] A. Kumar and D. Florêncio, arXiv:1605.02427.
[117] H. J. Pletsch and B. Allen, Phys. Rev. Lett. **103**, 181102 (2009).
[118] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (ACM, New York, 2015), p. 161.
[119] GPU-based deep learning inference: A performance and power analysis, https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf.
[120] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, SIGARCH Comput. Archit. News **44**, 243 (2016).
[121] G. E. Hinton and R. R. Salakhutdinov, Science **313**, 504 (2006).
[122] G. Cabrera-Vives, I. Reyes, F. Förster, P. A. Estévez, and J.-C. Maureira, Astrophys. J. **836**, 97 (2017).
[123] Wolfram Language System & Documentation Center, https://reference.wolfram.com/language/.
[124] LSC, LSC Algorithm Library software packages LAL, LALWRAPPER, and LALAPPS, http://www.lsc-group.phys.uwm.edu/lal.
[125] B. S. Sathyaprakash and B. F. Schutz, Living Rev. Relativity **12**, 2 (2009).
[126] D. Shoemaker, Advanced LIGO anticipated sensitivity curves (2010), https://dcc.ligo.org/cgi-bin/DocDB/ShowDocument?docid=2974.
[127] X. Glorot and Y. Bengio, *Proceedings of Machine Learning Research* 9, 249 (2010).
[128] NetModel, https://reference.wolfram.com/language/ref/NetModel.html.
[129] F. Löffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, R. Haas, I. Hinder, B. C. Mundim, C. D. Ott, E. Schnetter, G. Allen, M. Campanelli, and P. Laguna, Classical Quantum Gravity **29**, 115001 (2012), https://einsteintoolkit.org/.
[130] B. Wardell, I. Hinder, and E. Bentivegna, Simulation of GW150914 binary black hole merger using the Einstein Toolkit, https://doi.org/10.5281/zenodo.155394.