


Detection of gravitational-wave signals from binary neutron star mergers using machine learning

Marlin B. Schäfer^{1,2}, Frank Ohme^{1,2} and Alexander H. Nitz^{1,2}

¹Max-Planck-Institut für Gravitationsphysik, Albert-Einstein-Institut, D-30167 Hannover, Germany

²Leibniz Universität Hannover, D-30167 Hannover, Germany

 (Received 26 June 2020; accepted 13 August 2020; published 14 September 2020; corrected 2 October 2020)

As two neutron stars merge, they emit gravitational waves that can potentially be detected by Earth-bound detectors. Matched-filtering-based algorithms have traditionally been used to extract quiet signals embedded in noise. We introduce a novel neural-network-based machine learning algorithm that uses time series strain data from gravitational-wave detectors to detect signals from nonspinning binary neutron star mergers. For the Advanced LIGO design sensitivity, our network has an average sensitive distance of 130 Mpc at a false-alarm rate of ten per month. Compared to other state-of-the-art machine learning algorithms, we find an improvement by a factor of 4 in sensitivity to signals with a signal-to-noise ratio between 8 and 15. However, this approach is not yet competitive with traditional matched-filtering-based methods. A conservative estimate indicates that our algorithm introduces on average 10.2 s of latency between signal arrival and generating an alert. We give an exact description of our testing procedure, which can be applied not only to machine-learning-based algorithms but all other search algorithms as well. We thereby improve the ability to compare machine learning and classical searches.

DOI: [10.1103/PhysRevD.102.063015](https://doi.org/10.1103/PhysRevD.102.063015)

I. INTRODUCTION

The first direct detection of a gravitational-wave (GW) signal on September 14, 2015 [1] marked the dawn of gravitational-wave astronomy. During the first two observing runs, the LIGO and VIRGO Scientific Collaboration found 11 GWs [2] from coalescing compact binary systems. Two independent reanalyses of the data have discovered a further set of events, three of which are found to be of astronomical origin with probability $p_{\text{astro}} > 0.5$ by both studies [3–5]. The third observing run has identified tens of new GW candidate events [6] and so far reported four new GW detections [7–10]. With detector sensitivity improving further for future observing runs and KAGRA [11,12] joining the detector network, the rate of detections is expected to grow [13].

The most sensitive low-latency searches are tailored specifically to signals from coalescing compact binaries and use a fixed number of precalculated templates [14].

Each template is a unique combination of a waveform model and source parameters. These searches work by calculating an inner product between the data and every template to produce a signal-to-noise ratio (SNR) time

series. This process is known as matched filtering and is mathematically proven to be optimal for finding signals submerged in stationary, Gaussian noise [15].

If the SNR of a candidate exceeds a preselected threshold and the candidate is not excluded due to other factors, such as poor data quality or an implausible time of arrival difference between two different detectors, the low-latency search algorithms return a candidate event [16–19].

The computational cost of a matched-filter search scales linearly with the number of templates used. This number will grow with the improving detector sensitivity at low frequencies [20] of planned updates [13]. If currently neglected effects such as precession [21–24], higher-order modes [25–28], or eccentricity [29] are taken into account, even more templates would be required. More computationally efficient algorithms would enable searching for sources which cannot currently be targeted due to a fixed computational budget.

When detecting GWs from compact binary systems that contain at least one neutron star, the latency of the detection pipeline is critical, as these systems may produce electromagnetic (EM) signals. To detect these EM counterparts and maximize observation time, observatories need to be notified of possible events quickly. The number of false alarms, on the contrary, should be minimized, as telescope time is expensive. Current low-latency searches introduce a latency of $\mathcal{O}(10)$ s and operate at a false-alarm rate (FAR) of one per 2 months [14,16–19]. Any new search needs to meet or exceed these standards to be considered for production use.

Published by the American Physical Society under the terms of the Creative Commons Attribution 4.0 International license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI. Open access publication funded by the Max Planck Society.

Neural network (NN) machine learning algorithms are an interesting alternative to traditional search algorithms, as they have shown great improvements in many tasks such as image recognition [30], sound generation [31], or certain board and computer games [32,33]. NNs have also already found some application in the context of GW data analysis [34–42]. A few notable examples are the classification of non-Gaussian noise transients [34], the search for continuous GWs [35], and denoising of detector data to recover injected signals [36]. One key advantage of NNs is their computational efficiency once trained. Most of the computational cost is shifted to the training stage, resulting in very quick evaluation. The application of NNs to GW searches might therefore offer a way to reduce the computational cost of low-latency searches.

The authors of Refs. [43,44] were the first to directly apply deep NNs to time series strain data to detect GWs from binary black hole (BBH) mergers. They tested the sensitivity of these searches at estimated FARs $\mathcal{O}(10^3)$ per month.¹ Both analyses are able to closely reproduce the performance of a matched-filter search at these FARs at a fraction of the computational cost. The NNs excel at high FARs and low SNR. Both networks detected all signals with a SNR larger than 10 at estimated FARs of $\mathcal{O}(10^4)$ per month. These results are a promising first step, but the algorithms would need to be tested at the required FARs of one per 2 months on real detector data to demonstrate an improvement over established methods.

Starting from their network, we reshaped the architecture significantly to optimize it to detect signals from binary neutron star (BNS) mergers. Our network-based search estimates the SNR and a quantity we call the p score for the given input data. The p score is a measure for how likely the data are to contain a GW signal. It is explicitly not a probability. The network is trained on simulated data of nonspinning binary neutron star systems with masses between 1.2 and 1.6 solar masses, isotropically distributed over the sky. All noise is stationary and Gaussian and as such does not contain any transients or other contaminations that are present in real detector data [45–47]. The previous works [43,44] have used data from a single detector. To improve the performance of our search, we expand the algorithm to work with data from two detectors. Using multiple detectors may also enable real-time estimates of the sky position in the future.

Detecting BNS signals using a NN is inherently more difficult than finding a BBH signal, as (i) the GW of a BNS reaches higher frequencies and (ii) spends more time in the sensitive bands of the detectors. Because of (i), the data need to be sampled at a high rate. Combined with (ii), this

¹We estimate this FAR by multiplying the false-alarm probabilities given in Refs. [43,44] by the respective number of samples times the duration by which the position of the peak amplitude is varied within the training data samples.

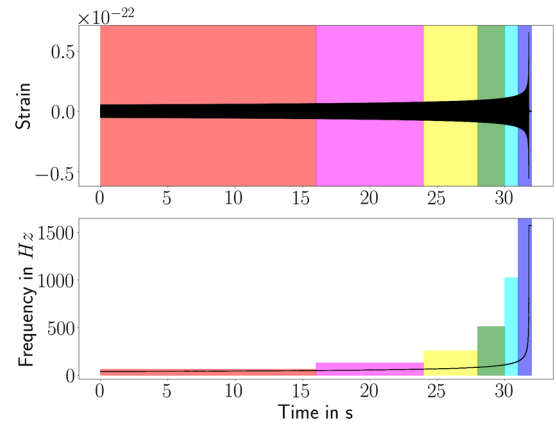


FIG. 1. The top panel shows the strain evolution of an example GW from a BNS merger in black. The bottom panel shows the corresponding frequency evolution in black. The colored boxes represent parts of the signal which we sample at different rates. The height of these boxes in the bottom panel represents the Nyquist frequency of the sample rate which is used for each part. To fully resolve the signal, the black curve must stay inside the colored boxes of the bottom panel at all times.

leads to a massive increase of data that need to be analyzed. As NNs tend to be difficult to optimize when the input data have many samples, it is not feasible to naively use the full time series sampled at a single rate as input. To solve this problem, we sample different parts of the signal at different rates. Frequencies emitted during the early inspiral are low and evolve slowly (see Fig. 1). High sample rates are necessary only during the final few cycles, where frequencies are high and grow rapidly.

The FARs probed by Refs. [43,44] are orders of magnitude larger than what is required for low-latency pipelines. Additionally, these FARs were estimated on a discrete set of samples which either contain a signal or consist of pure noise. The waveforms within these samples are always aligned in a similar way, and no signal is contained only partially in the analyzed segment. As the authors of Ref. [48] point out, FARs estimated on a discrete set of samples may for these reasons not be representative of a realistic search which has to work with a continuous stream of data.

We propose a standardized way of evaluating NN FARs and sensitivities. To calculate these metrics, we generate a long stretch of continuous time series data which contains many injected GWs that are roughly separated by the average duration of a BNS signal. Our network is applied to these data, and points of interest are clustered into events. All results we provide are derived from an analysis of ≈ 101 days of simulated continuous data. We test the network down to FARs of 0.6 per month and find sensitive distances of 130 Mpc down to FARs of ten per month.

We compare our search to the currently in-use low-latency detection pipeline PyCBC Live [18] and the results given by the authors of Ref. [49], who were the first to classify BNS signals with a machine learning algorithm. We find an

improvement in sensitivity of close to 400% for BNS signals with a SNR in the range of provided training examples ($8 \leq \text{SNR} \leq 15$) over the previous state-of-the-art machine learning algorithm. This makes our algorithm the best machine learning algorithm for detecting BNS signals at low SNRs. We are, however, not yet able to match the performance of template-based searches. To do so, we need to either increase the sensitive radius of our search at the lowest FARs by a factor of 6 or double the sensitive radius while lowering the FAR by an order of magnitude.

The trained network is public and can be found in the associated data release [50]. At the same location, we also provide example code of how to apply it to long stretches of data and a function that generates injections as they are used to derive FARs and sensitivities in this work.

The contents of this paper are structured as follows: Sec. II describes how search algorithms should be evaluated. It gives the general concepts in the first part and details on how to apply these concepts to NNs in the second part. Section III explains the multirate sampling and describes the data used for both training and validation of the network. The following Sec. IV gives an overview of the architecture and how this architecture is trained and tested. We present our results in Sec. V, of which we draw conclusions in Sec. VI.

II. FALSE-ALARM RATE AND SENSITIVITY OF GRAVITATIONAL-WAVE SEARCH ALGORITHMS

There are two important metrics that have been used to evaluate gravitational-wave searches in the past. These two are the FAR of the search and the corresponding sensitivity [51]. In principle, these metrics can directly be applied to GW searches that utilize NNs. As pointed out by the authors of Ref. [48], in practice the discrete nature of the data that are used to train these networks has lead to some divergence between the terminology used for NN and traditional search algorithms.

A. Calculation for general search algorithms

The main goal of a search algorithm is to detect GW signals in real data, where the input is a nearly continuous strain time series. A search, therefore, must produce a list of times of candidate events and rank them by a *ranking statistic* \mathcal{R} . The ranking statistic is a number which signifies how likely the data are to contain a signal. To evaluate the performance of an algorithm, it is applied to mock data containing known *injections*, i.e., additive GW signals with known parameters. The events generated from these data are compared to the list of injections and used to determine which injected signals were found, which missed, and which events are false alarms.

Any event that is reported by the search needs to be assigned a FAR to express the confidence in its detection. For a given value \mathcal{R} , the FAR is the number of false alarms

with a ranking statistic of at least \mathcal{R} per unit time. To estimate it on mock data, the number of false detections exceeding a ranking statistic \mathcal{R} is divided by the duration of the analyzed data.

The ability of the search to recover signals is quantified by the sensitivity which is a function of the FAR lower bound. It is often given in terms of the fraction of recovered injections. This fraction, however, strongly depends on the parameter distribution of the injected signals, as the amplitude of the signal in the detector depends on the orientation and location of the source. Thus, the fraction can be diminished by injecting sources at larger distances or unfavorable orientations. A more astrophysically motivated measure of sensitivity is the sensitive volume of the search algorithm. It is an estimate of the volume around the detector from which GW sources will be detectable. This volume may be calculated through

$$V(\mathcal{F}) = \int d\mathbf{x} d\Lambda \epsilon(\mathcal{F}; \mathbf{x}, \Lambda) \phi(\mathbf{x}, \Lambda), \quad (1)$$

where $\epsilon(\mathcal{F}; \mathbf{x}, \Lambda)$ is the efficiency of the search pipeline for signals with FAR \mathcal{F} , spatial coordinates \mathbf{x} , and injection parameters Λ . The function $\phi(\mathbf{x}, \Lambda)$ is a probability density function which describes the astrophysical distribution of signals [51]. When the distribution of injections matches the expected astrophysical distribution (i.e., uniform in volume, isotropic in sky location, etc.), Eq. (1) can be estimated by

$$V(\mathcal{F}) \approx V(d_{\max}) \frac{\# \text{found injections}(\mathcal{F})}{\# \text{total injections}}, \quad (2)$$

where d_{\max} is the maximal distance of injected signals, $V(d_{\max})$ is the volume of a sphere with radius d_{\max} , and the function $\# \text{found injections}(\mathcal{F})$ counts the number of detected injections with a FAR $\leq \mathcal{F}$.

We use the function `volume_montecarlo` of the PyCBC software library [52] to carry out this estimation.

Current searches notify astronomers of a GW event when the event is assigned a FAR of at most one per 2 months [18]. Any new search should, hence, be tested at least down to these FARs. To resolve FARs of that scale, at least 2 months of mock data are required.

For our tests we generated 100 files, each containing roughly 1 day of continuous data. Each file contains independently drawn data. For easier multiprocessing, each file is internally split into 22 chunks of duration 4096 s. We start by generating a list of injection times, requiring that two injections are separated by 180–220 s. The exact separation time is chosen uniformly from this interval. To avoid waveforms that are not completely within one chunk, we discard any injections that are within the first or final 256 s of each chunk. For every injection time, we generate a waveform using the inspiral-only waveform model TaylorF2 [53–55] with a lower frequency cutoff of 25 Hz. Its parameters are drawn from the distribution

TABLE I. The astrophysically motivated distribution of parameters used to generate injections. These are used to estimate the FAR and sensitivity of the search algorithm specified in this paper.

Parameter	Uniform distribution
Component masses	$m_1, m_2 \in (1.2, 1.6)M_\odot$
Spins	0
Coalescence phase	$\Phi_0 \in (0, 2\pi)$
Polarization	$\Psi \in (0, 2\pi)$
Inclination	$\cos \iota \in (-1, 1)$
Declination	$\sin \theta \in (-1, 1)$
Right ascension	$\varphi \in (-\pi, \pi)$
Distance	$d^2 \in (0^2, 400^2) \text{ Mpc}^2$

specified in Table I. Finally, the waveform is projected into the frame of the LIGO-Hanford and LIGO-Livingston detectors and added into simulated Gaussian noise such that the peak amplitude is positioned at the injection time. All noise is generated from the analytic estimate of the power spectral density (PSD) of the aLIGO final design sensitivity as provided by the software library LALSuite [56].

B. Calculation for neural network searches

A NN is a general purpose function approximator that finds a fit for a set of example input-output pairs. This fitting process is called training, and the example data used are called the training set. Once trained, the network can be applied to data that were not covered by the training set and will evaluate the fit function at this new point. It does so assuming that the unseen data samples originate from the same underlying process as the training data samples. The given output for any unseen data sample is thus an interpolation or extrapolation of the example outputs from the training set.

To generate FARs and sensitivities, previous works [43,44] generated a second set of noise and signal samples with the same duration and sample rate used in the training set. They then applied the network to this second set of data samples and determined FARs by counting how many noise samples were classified as signals and sensitivities by counting how many signal samples were classified as such.

There are two main problems with using a discrete set of data samples to determine FARs and sensitivities. The first stems from the structure of the data samples themselves. To make the training process more efficient, it is necessary to position the peak amplitude of the GW signal within a narrow band of the data sample. When applied to real data, this property cannot be ensured, and the assumption of the data being similar to the training set is not well approximated. Hence, if a FAR or sensitivity is calculated on a set where the alignment is guaranteed, it will not necessarily be representative of the performance on realistic data. The second problem is the required fixed duration of input data samples. Usually, a search algorithm is applied to long

stretches of time series data to find potential signals. To evaluate data of greater duration than the accepted input size of the network, it is applied multiple times via a sliding window. At each position, the output will give an estimate if a signal is present. If this is the case, it is initially not clear what a true positive is, as the network may predict the presence of a signal for multiple consecutive positions, the input window may only partially contain a signal, or the network may jump between predicting the presence and absence of a signal for multiple subsequent positions [48].

To generate representative FARs and sensitivities, we propose to use mock data of much greater duration than the input duration of the network. The network is then applied to these data by sliding it across. The step size should at most be half the size of the interval where peak amplitudes of the waveforms occur in the training set. This step size ensures that any waveform is positioned correctly for at least one position of the network.

If the output of the network is not binary but continuous, it can be interpreted as a ranking statistic. In this case, a threshold can be applied to find positions where the network predicts to have found a signal. Candidate events are identified in the resulting time series by applying a threshold and clustering.

Each event is assigned a specific time and ranking statistic. The resulting list of events is compared to the list of known injections as described in Sec. II A to calculate FARs and sensitivities. The specifics of our analysis and clustering algorithm are described in Sec. IV C.

As every event needs to be assigned a ranking statistic, we can calculate the metrics by using only events that exceed a given threshold. Doing so for many different values allows us to obtain the FAR as a function of the ranking statistic threshold and, subsequently, also the sensitivity as a function of the FAR.

We found that testing our NN on long stretches of data and applying clustering to obtain a list of events increased the FAR over FARs measured on a set of discrete samples at the same detection threshold by at least a factor of 2. To give comparable statistics, we therefore strongly recommend to test networks in the way described above.

III. DATA PROCESSING

To train and evaluate the network, three disjoint datasets with known contents are required. One of the three sets is the training set and used to optimize the parameters of the network. The network is usually trained on the training set multiple times, where each complete pass is called an epoch. After multiple epochs, the network may start to learn specifics of the provided data samples rather than the general structure. This behavior is called overfitting and can be detected by monitoring the performance of the network on a validation set. Different epochs are rated by their performance on this second set. Ranking the different

training stages of the network in this way introduces a selection bias and optimizes the network on the validation set. To give unbiased results, a testing set is required. This one should represent the data that the network will be applied to the closest and should optimally be generated independently from the training and validation set. To keep results as unbiased as possible, the testing set should ideally be analyzed only once. This section describes how an individual data sample needs to be formatted for our network and how the training and validation set are generated. Details on the testing set are described in Sec. II A.

A. Input data preparation

Previous works [43,44,57] have already successfully classified whitened time series data for BBH signals with a simple convolutional neural network. As input, they used 1 s of data sampled at 8192 Hz. For BBH signals, this is a sensible choice for the duration of analyzed data, as these signals sweep through the sensitive frequency range of the detectors in $\mathcal{O}(1)$ s, and the chosen sample rate is sufficient to resolve them. Signals from binary neutron star mergers, on the other hand, spend $\mathcal{O}(100)$ s in the sensitive band of the detectors. Using the usual signal duration as input to the network would lead to a hundredfold increase of data points over the BBH case. Training a NN with this many input samples is infeasible due to memory constraints and optimization problems.

To reduce the number of input samples, the authors of Ref. [49] use only the final 10 s of each signal as input. This enables them to not only differentiate between noise and BNS signal, but also distinguish GWs from BBH mergers. They test an *architecture*, i.e., a network structure, similar to those of Refs. [43,44] and are able to closely reproduce their results for BBH data. Their sensitivity to BNS signals looks very promising, but the search has yet to be tested at realistic FARs. The short duration of the input in comparison to the duration a BNS signal spends inside the sensitive band of the detectors reduces the SNR contained in the data by about 25% and, thus, limits the sensitivity of the search.

To retain as much SNR in the data as possible while at the same time reducing the input size to the network, we sample 32 s of data containing a potential signal at different rates. During the early inspiral, frequencies are low and the frequency evolution is slow. This allows us to sample a long stretch in the beginning at a low rate. The first 16 s are sampled at 128 Hz, the following 8 s are sampled at 256 Hz, and so on. The pattern continues until the final second, which is sampled at 4096 Hz but split into two parts of equal length. We ensure that no two sections overlap to reduce redundant information. This method of resampling generates seven parts containing 2048 samples each and ensures that for every part of the signal a sufficient sample rate is used. The number of samples is reduced by a

factor of 9 (see Fig. 1) and about 98% of the SNR is retained.²

Rather than resampling the data directly, we first *whiten* it using the analytic model `aLIGOZeroDetHighPower` for the aLIGO design sensitivity PSD as provided by the software library `LALSuite` [56]. Whitening of data is a procedure where every frequency bin is reweighted by the average power of the background noise in this bin. It ensures that power in any frequency bin in excess of unity is an indication for the presence of a signal. For computational efficiency during training, noise and signal samples are whitened individually. Since the whitening procedure is a linear operation, whitening the sum is equivalent to whitening both parts individually. Both parts are combined at run time on the first layer of the NN. The reason to store them separately is an increase in the effective number of samples which can be achieved by mixing and matching different signals and noise samples. It also helps to improve the efficiency of training by using the same signal template submerged in different realizations of noise.

When evaluating real samples, we cannot trivially separate the signal from the background and, thus, cannot whiten each part individually. Instead, we whiten the total signal by the same analytic model of the PSD used for the training and validation data. The whitened data are resampled and used as the signal input. As the signal input already contains the total signal including noise and the network sees only the sum of both inputs, the noise input is set to zero.

B. Generating training and validation set

All signals for the training and validation set are generated using the inspiral-only waveform model `TaylorF2` from the software library `LALSuite` [56] with all parameters but the distance drawn from the distribution given in Table I. The luminosity distance d is set indirectly by uniformly drawing a target network SNR from the interval [8, 15]. The waveform is first computed at a fiducial distance of 1 Mpc with a low-frequency cutoff of 20 Hz. Then the waveform is projected onto the two detectors Hanford and Livingston [59] and cropped to a length of 96 s. During this step, we shift the waveform such that the peak amplitude occurs within the final 4.25–4.75 s³

²Notice that this way of sampling the data differs from our previous work [58] in that we dropped the lowest sample rate. We found that for some signals a sample rate of 64 Hz for the first 32 s of a 64-s signal was not sufficient to resolve the highest frequencies during that stage of the binary evolution and introduced unwanted artifacts. We also sampled the PSD used for whitening the data too coarsely in our previous work [58], and signals were more difficult to find as a result.

³Altering the time of the peak amplitude of the waveform during training allows the network to be less sensitive to the exact position of the waveform within the input window. This enables us to slide the network across long stretches of input data with a larger step size. For this study, we chose to use an interval of 0.5 s. It may be possible to optimize this choice, but we have not done so here.

of the 96-s data segment. The exact position within this interval is drawn uniformly. Next, we calculate the network SNR by taking the root of the sum of the squares of the inner product of the waveforms with themselves [59], weighing each frequency by the analytic PSD `aLIGOZeroDetHighPower` of Ref. [56]. The waveforms are finally scaled by multiplying with the target network SNR and dividing by the network SNR at distance 1 Mpc. Afterward, the data are whitened, the initial and final 4 s are discarded to avoid filter wraparound errors, and the last 32 s of the remaining segment are resampled as described in Sec. III A. Noise samples are simulated from the analytic PSD used above, whitened, and resampled. As such, all noise is stationary and Gaussian.

The training set contains 25 000 different GW signals and 80 000 instances of noise. When training the network, we preselect 800 000 unique combinations of signal and noise at random and shuffle them with all 80 000 noise samples to obtain 880 000 training samples with a 10:1 split between signals and noise. To compensate for this inequality, we apply *sample weights* of 1/10 to all signal samples during training. Sample weights modify the loss by reweighting contributions from the according sample.

The validation set contains 1500 different GW signals and 8000 instances of noise. We again generate 24 000 unique combinations of signal + noise and shuffle them with the 8000 noise samples. This results in a validation set that contains 32 000 samples with a 3:1 split for signal: noise.

IV. THE SEARCH ALGORITHM

A. Neural network architecture

When working with neural networks, the details of the implementation of the machine learning algorithm are mostly defined by the architecture of the network. There is no known optimal procedure for designing a network that works well for a given problem.

The architecture presented in this paper is highly optimized for the problem of detecting BNS signals and relies on the input data format described in Sec. III A. Some of the main improvements over a standard convolutional architecture will be more general and may be of use for different data formats and similar problems.

We started our research by adjusting the architecture given in Refs. [43,44] for data sampled at multiple rates, by using one channel for every sample rate and detector combination. In convolutional networks, channels represent different features of the data and are correlated by the convolutional filters. With this as a starting point, we made iterative improvements. The three changes that had the greatest positive effect were the replacement of convolutional layers by inception modules [60], the use of a temporal convolutional network (TCN) as a signal amplifier [36,61,62], and using different stacks of inception modules for each sample rate. A detailed description of

the evolution of the network can be found in Ref. [58]. The architecture presented here differs from an earlier iteration presented in Ref. [58] only by removing the lowest sample rate as input and adjusting the structure accordingly.

For computational efficiency, we provide the noise and signal time series not as a sum but as separate inputs to the network. They are combined on the first layer of each parallel stack of layers (see Fig. 2). This sum is passed to a TCN, which tries to recover the pure signal. The denoised data are added back onto the input of the TCN to amplify potential signals. The amplified data are preprocessed by convolutional layers before two inception modules with very small kernel sizes are applied. Afterward, two adjacent stacks are concatenated and used as input to further inception modules. The process is repeated until only a single stack is left. This stack is reduced down to the desired output format by applying dense layers. A high-level overview of the architecture can be found in Fig. 2.

Inception modules are the main building block of the network. Their use was motivated by recent developments for image recognition tasks [30,60].

The main advantage of inception modules over convolutional layers is the reduced number of parameters. In convolutional layers, often many weights are close to zero after training. Ideally, sparse operations would be used in such cases. Sparse operations are, however, not computationally efficient. The idea of inception modules is to combine many small, dense filters in parallel to form an effective large, sparse filter. This approach allows for the use of efficient dense operations while reducing the number of trainable parameters at the same time [60].

The final outputs of our network are one scalar for the SNR estimate and a tuple of length 2 estimating the p score. The p score is a measure for how confident the network is that the data contain a GW and the content of the corresponding tuple is (p score, 1 – p score) for technical reasons. For the training and validation set, signal samples are labeled with a p score of 1, and noise samples are labeled with a p score of 0. The network output, on the other hand, is not binary but continuous. We thus interpret both the SNR and the p-score output as ranking statistics.

Alongside the two outputs described above, the network is equipped with 13 further auxiliary outputs. The purpose of these outputs is to prevent vanishing gradients [60] or provide the intermediate layers with more information on their purpose. The auxiliary outputs thus improve the training efficiency of the network. Seven of the auxiliary outputs are the outputs of the TCNs. They are trained using the pure signals as the target. We found that the network is significantly more sensitive if it cannot decide how to use the parameters of the TCNs freely but is forced to learn to recover the GW. Five of the remaining six outputs are taken after all but the final concatenation layer. They receive the injected SNR as the target. Since the output of the concatenation layers is not a scalar, we use a few pooling

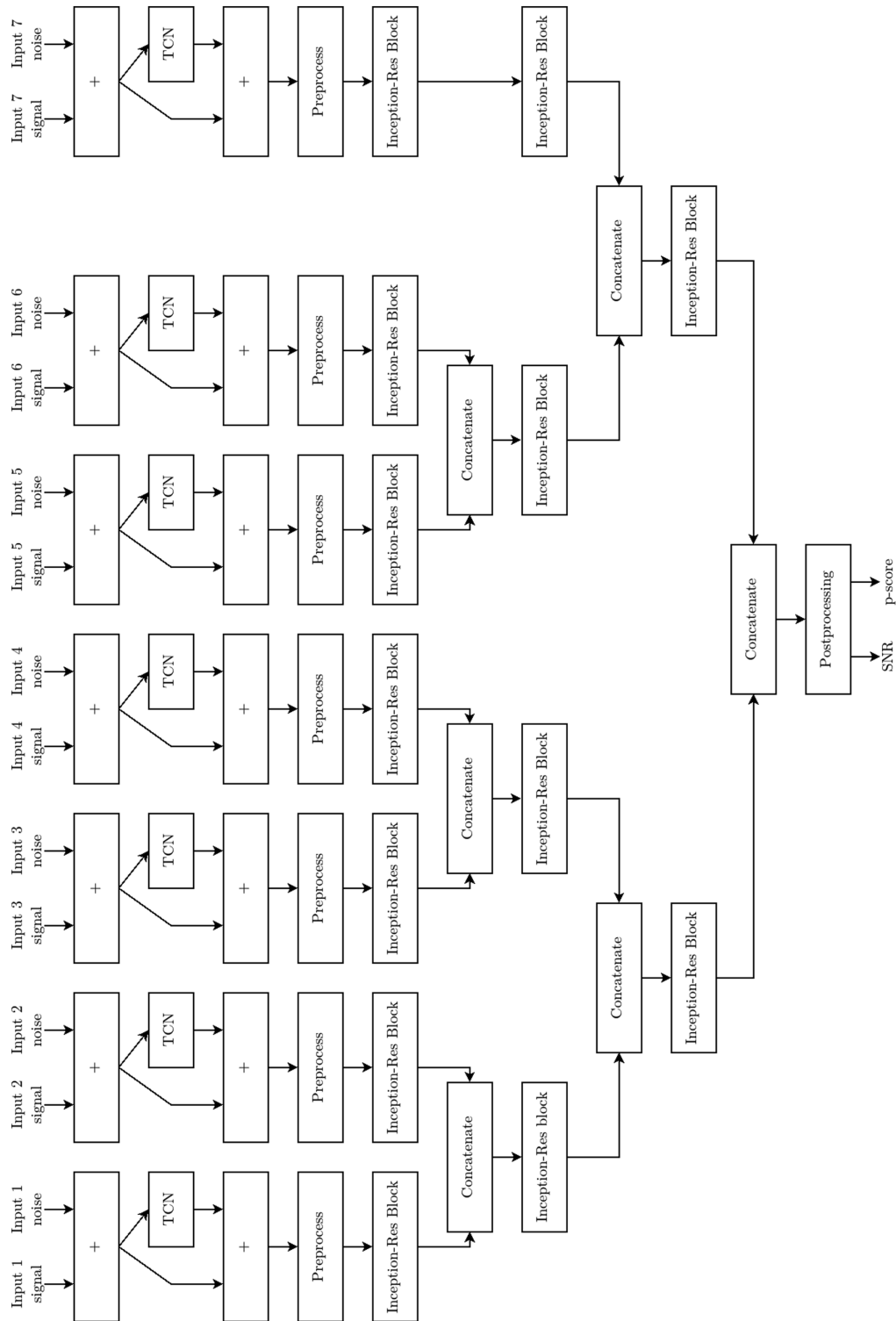


FIG. 2. A high-level overview of the architecture presented in this work. Details on every block can be found in Ref. [58]. The network takes signal and noise inputs 1–7, where each number corresponds to a different part of the resampled raw data described in Sec. III A. It outputs an estimate of the SNR contained in the input and a p score, which rates how likely the data are to contain a BNS signal. All auxiliary outputs that are used only for training are not shown. The network adds the noise and signal input for every resampled part individually, and the remaining layers operate only on this sum. The output of this addition is amplified by a TCN and processed by an inception network. Afterward, the outputs of two inception networks from adjacent sample rates are concatenated and further analyzed by another inception network. The parallel inception networks are concatenated until only a single one remains. A few final dense layers, which are summarized as the postprocessing block, are applied to reduce the output shape to the desired dimensions of the SNR estimate and p score. The preprocessing block is inspired by Ref. [60] and contains a small convolutional network.

and dimensional reduction layers [58,63] to reduce the output shape. The final auxiliary output is taken after the first two inception modules of the lowest sample rate and treated in the same way as the auxiliary outputs after the concatenation layers. The network is trained as a complete unit, and the loss takes into account all 15 outputs.

The complexity of this architecture comes at the cost of memory size and speed. The model has 2.5 million trainable parameters. The computational cost is a problem when optimizing the architecture, as it is costly to compare two different designs. We therefore suspect that the details of this architecture can be further optimized.

B. Training

The network requires 17 GB of memory to be trained with a minibatch size of 32. We used an NVIDIA V-100 GPU with 32 GB of video memory for training. On this hardware, each training epoch plus the subsequent validation step takes roughly 5 h to complete. Because of time constraints and instabilities during training, the network is trained for 24 epochs only. The instabilities are discussed below and manifest as a sudden drop of the sensitivity during training.

The total loss of the network is the weighted sum of the individual losses for each of the outputs discussed in Ref. IV A. All auxiliary outputs are assigned a mean squared error as the individual loss and given a weight of 0.1. The same loss function is used for the SNR output, but it receives a weight of 1. Finally, the p-score output uses the categorical cross entropy and a weight of 0.5. The total loss is thus given by

$$\begin{aligned}
 L(\mathbf{y}_{\text{true}}, \mathbf{y}_{\text{pred}}) &= \text{MSE}(\text{SNR}_{\text{target}}, \text{SNR}_{\text{pred}}) \\
 &+ \frac{1}{2} \sigma(\text{p score}_{\text{target}}, \text{p score}_{\text{pred}}) \\
 &+ \frac{1}{10} \sum_{i=1}^{13} \text{MSE}(y_{i,\text{target}}, y_{i,\text{pred}}), \quad (3)
 \end{aligned}$$

where $\text{MSE}(x, y) := (x - y)^2$ is the mean squared error, $\sigma(x, y) := -\sum_{i=1}^2 x_i \log(y_i)$ is the categorical cross entropy, a subscript “target” indicates the known target values, a subscript “pred” indicates the network output, and the y_i are the auxiliary outputs. The different weights are used to inform the optimization algorithm on the importance of each individual output. The auxiliary outputs are used only during training and discarded during inference. Their value is unimportant as long as using them improves the performance of the SNR and p-score output. We use the default implementation of the “Adam” optimizer from the machine learning library Keras [64] to train the entire network using the total loss. Altering the initial learning rate in either direction reduced the sensitivity of the network.

During training, we monitor an estimate of the sensitivity of our network. To do so, we calculate the true positive rate on the validation set, by choosing the maximum predicted SNR and p-score value of all noise samples from the validation set as a threshold. All signals that are estimated with a ranking statistic higher than these thresholds are counted as detected. The number of detected signals is then divided by the total number of signal samples to get a true positive rate. We rank the epochs based on this estimate of the sensitivity and thoroughly test the best one.

We found that the network experienced strong overfitting. While the training loss fell by 25% from the initial to the last epoch, the validation loss doubled. If the loss and the sensitivity were strongly correlated, it would be expected that the sensitivity drops with an increasing validation loss. We find the opposite and reach the highest true-positive rate of 16.5% on epoch 21. At this point, the validation loss grew by 75% over the initial epoch. The loss in use is, therefore, at best loosely correlated with the sensitivity. Designing a loss function that is better suited to the problem might improve the search further. The strong overfitting also indicates the possibility to simplify the architecture significantly without a strong loss of sensitivity or improving the performance of the current architecture by increasing the size of the training set significantly.

When training networks that predict if a GW signal is present in some time series data, we found that after some number of epochs the sensitivity estimate drops to zero for both the SNR and the p-score output. Initially, it often recovers on the next epoch, but drops become more frequent. After some point, the network does not seem to recover at all and the estimated sensitivity stays at zero. This behavior is caused by noise samples that are estimated with very high confidence to contain a GW. These are sometimes appointed physically nonsensical SNR values. The number of these misclassified noise samples is low, and, thus, the impact on the total loss is negligible. Furthermore, the values that are given for these noise samples grow over time, which is the reason why the drop occurs only after training for a while. In principle, these outliers may be vetoed by their SNR value at the cost of some sensitivity at low FARs. We disfavor this approach, as it introduces artificial constraints on the search algorithm. It is currently unknown what causes the predicted values of the ranking statistics to grow or how the issue can be resolved. To avoid problems, we stop training before the sensitivity estimate stays at zero for many consecutive epochs.

Previous works [43,49] have reported that a training strategy known as *curriculum learning* improved the performance of their networks. During curriculum learning, the network is trained on high SNR examples in the beginning, with training samples getting more difficult to differentiate from noise during later epochs. We tested this approach during early stages of our research and could not

find a significant impact on the performance of our networks at the time. We, therefore, did not pursue the strategy during later stages of our project. However, we do suspect that curriculum learning might help to mediate the deficiencies of our network for loud signals.

C. Testing on binary neutron star injections

To evaluate the sensitivity of the network, we use the test data described in Sec. II A. It contains $8\,794\,112\text{ s} \approx 101$ days of data split into 100 files. With this dataset, FARs down to ≈ 0.3 false alarms per month can be resolved.

To analyze continuous stretches of data, we use a sliding window of duration 72 s with a step size of 0.25 s. We chose the step size based on the training set in which the exact position of the peak amplitude was varied by ± 0.25 s around a central position.

The content of every window is whitened by the analytic model PSD of the advanced LIGO detectors as provided by the software library LALSuite [56]. To avoid filter-wraparound errors, the initial and final 4 s are discarded. The final 32 s of the remaining data are resampled and formatted as described in Sec. III A.

To assign the correct times to each window, the alignment of the waveforms in the training set needs to be considered. The central position for the peak amplitude in the training set is set to 0.5 s from the end. If the merger time is defined as the time of the peak amplitude of the waveform, it will on average be positioned 31.5 s from the first sample of the 32-s input window. Considering the 36 s that are discarded at the beginning of each window, the first position of a GW merger we are sensitive to is located at 67.5 s from the start of each continuous segment. The reported sample times are, therefore,

$$t(n) = t_{\text{start}} + 67.5\text{ s} + (n - 1) \cdot 0.25\text{ s}, \quad (4)$$

where $t(n)$ is the time of the n th sample and t_{start} is the starting time of the analyzed data segment.

By applying our network in this way, we obtain two time series. One estimates the SNR at each window position, while the other gives a p score at every step. We apply a fixed threshold of SNR 4 and p score 0.1 to the respective time series. Every position that exceeds the corresponding threshold is marked. All marked positions are then clustered by assuming that two marked positions are generated by the same underlying process if they are within 1 s of each other. The clusters are expanded until there are no marked positions within 1 s of the boundaries of the cluster. Each cluster is an event and assigned the time and value of the maximum SNR or p score, respectively, inside this cluster. An event is said to be a true positive if an injection was placed within ± 3 s of the reported time. The times used for clustering and accepting a signal as a true positive were empirically found to work well on a different dataset and are arbitrary choices.

V. RESULTS

A. False-alarm rate and sensitivity

The analysis of the BNS test data described in Sec. IV C returns a list of events. Each event is assigned a ranking statistic. We obtain the FAR as a function of the ranking statistics SNR and p score (Fig. 3) by considering only those events that exceed the given threshold. Subsequently, we can generate the sensitivity as a function of the FAR (Fig. 4). We choose a range of SNR 4–20 and p score 0.1–1 to generate these plots.

We find that the SNR estimate is able to resolve FARs down to 0.6 per month, whereas the p-score output is able to resolve FARs down to 12 per month. Both curves drop steeply with the corresponding ranking statistic until they reach a FAR of $\mathcal{O}(10)$. At this point, both curves level off significantly. Our previous work [58] was able to resolve FARs down to ≈ 30 per month and was tested on a set of roughly half the duration used in this paper. We also observed a change in gradient of the FAR in Ref. [58] although at smaller ranking statistics. For the SNR output, this change lined up well with the lower limit of the SNR

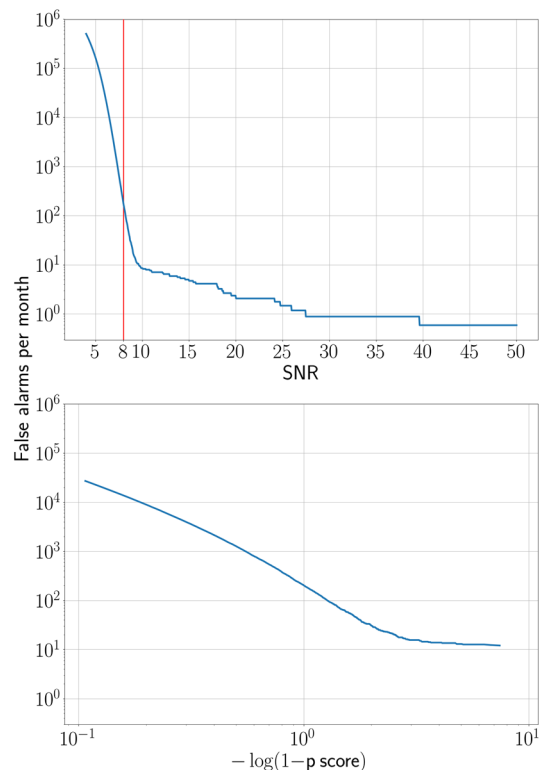


FIG. 3. The estimated FAR as a function of the threshold value used for either output. The top panel shows the FAR of the SNR output. The red line in this plot points out the lowest SNR of training samples. In our previous work [58], we found a change in gradient at this position. For the current search, this change appears at a higher SNR. The bottom panel shows the FAR of the p-score output. It is logarithmic and is scaled to give a sense for the distance to p score = 1.

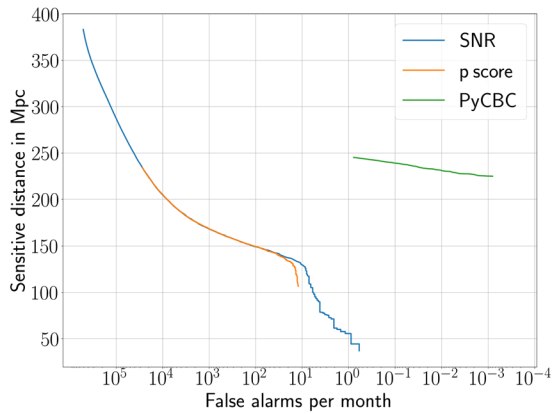


FIG. 4. The sensitive distance as a function of the FAR. The blue curve shows the sensitive distance when the SNR is used to classify events. The yellow curve shows the sensitive distance when the p score is used. The green curve is generated from the data found in Ref. [65] by counting all signals at a higher injection SNR than the corresponding FAR. We are able to resolve a small overlap region between the two different searches but find that the sensitivity of our search drops close to zero for FARs below ten per month. At high FARs, both outputs of our network perform equally well; for low FARs the SNR shows superior performance.

contained in the training samples. This may be a hint that the network presented in Ref. [58] successfully learned the lower bound on the SNR in the training set.

For high FARs, both outputs show equivalent sensitivities. At low FARs, on the other hand, the SNR output is more sensitive and has non-negligible sensitivities down to a FAR of ten per month, where it reaches a sensitive radius of ≈ 130 Mpc. The sensitivity of the p-score output becomes negligible around a FAR of 20 per month and also reaches a sensitive radius of ≈ 130 Mpc.

In our previous work [58], we observed the opposite behavior with regards to which of the two outputs is more sensitive at low FARs. We do not know what causes either output to perform better than the other.

We can also observe a change in gradient in the sensitivity curves shown in Fig. 4. The locations where the sensitivity starts to suddenly drop steeply line up with the point where the FAR levels off observed in Fig. 3. At FARs below this point, the sensitivity becomes negligible quickly.

B. Comparison to PyCBC Live

We compare our search to PyCBC Live [65], which is a low-latency analysis and has been used in the second and third observing runs [2,7,8,14]. The green curve in Fig. 4 is estimated from Fig. 1 in Ref. [65] on our test set by assuming that all injections with optimal $\text{SNR} > \mathcal{R}$ are found and all others are missed. Here, \mathcal{R} is the network SNR reweighted by signal consistency tests corresponding to a given FAR. At a FAR of 0.5 per month, the PyCBC

Live search has a sensitive radius of ≈ 245 Mpc. At a comparable FAR of 0.6 per month, our search reaches 1/6 the sensitive radius. At a FAR of ten per month, where our search is still marginally sensitive, the radius increases to ≈ 130 Mpc, which is still about half the radius of the reference value from the PyCBC Live search. To reach this reference value, we would need to operate at a FAR of ≈ 35000 per month.

To compare the computational cost of our search to that of PyCBC Live [18], we analyze the resources both algorithms require to evaluate incoming data in real time. One pass of our network on the correctly whitened, resampled, and formatted data takes 206 ms on an Intel i7-6600U dual-core, four-thread laptop processor. Neglecting the cost of preprocessing the raw input data, our search would be able to produce estimates of the SNR and p score slightly faster than real time with the above-mentioned resources, as each step processes a time span of 250 ms. We can estimate the number of CPU cores PyCBC Live would require to run a search for BNS signals of the distribution our network was trained on by counting the number of templates the corresponding filter bank would use. To produce a nonspinning filter bank with component masses between 1.2 and 1.6 solar masses at 3.5 PN order, we use the `pycbc_geom_nonspinbank` program from the PyCBC software library [52]. The minimum match is set to 0.965. With these settings, the bank contains 1960 templates per detector. The PyCBC Live search is able to evaluate 6300 templates per core in real time [18]. The required 3920 templates for a two-detector search could therefore be evaluated on a single core in real time.

At a FAR of ten per month, our search introduces an average latency of 10.2 s for true positives. This value is calculated by taking the difference between the latest position in any cluster at the given FAR that belongs to a real injection and the reported time of the corresponding event. To ensure that the cluster is complete, we add 1 s on top of that latency and another 0.206 s to ensure the network has finished its calculations. We then average over all clusters evaluated that way. Our search algorithm has not yet been optimized for low-latency analysis, and we assume that the latency can be reduced by about an order of magnitude by choosing a different clustering algorithm without a large impact on the sensitivity. The reported latency does not take into account any time lost due to whitening, resampling, or formatting of the raw input data. PyCBC Live for comparison operates at an average latency of 16 s. This latency can be reduced to 10 s at the cost of doubling the required computational resources [18].

C. Comparison to another machine learning algorithm

The authors of Ref. [49] were the first to search for BNS signals using a machine learning algorithm. They used a convolutional network very similar in structure to those found in Refs. [43,44] to analyze 10 s of data sampled

at 4 kHz. This setup allowed them to differentiate the three classes: “pure noise,” “BBH signal,” and “BNS signal.” They found that their algorithm is able to distinguish the three classes and is as sensitive to BBH signals as the previous comparable search algorithms [43,44]. The sensitivity to BNS signals is below that of BBH signals for all signal strengths and false-alarm probabilities tested.

During the preparation of this paper, the original preprint [66] was rewritten and published. In that version, the sensitivity to BNS signals was found to be on par with the sensitivity to BBH signals. However, all results were given in terms of peak signal-to-noise ratio (pSNR) instead of optimal or matched-filter SNR. The new version removes this hurdle and gives results in terms of optimal SNR. We therefore comment on both versions.

To convert between pSNR and matched-filter SNR, the authors of Ref. [66] quote a factor of 13, which was derived on BBH data. We calculated this factor on BNS data and find a conversion of optimal SNR \approx matched-filter SNR $\approx 41.2 \cdot$ pSNR. Furthermore, they used data from a single detector. Signals detected at SNR ρ gain on average a factor of $\sqrt{2}$ when a network of two detectors is used. Our results are compared to the findings of Ref. [66] by using the conversion optimal SNR = $41.2 \cdot \sqrt{2} \cdot$ pSNR.

The comparison between our work and the results given in Ref. [49] still takes the scaling factor of $\sqrt{2}$ into account to compensate for the two-detector setup.

Figure 5 compares the true positive rates of the search algorithm presented here to the one found in Refs. [49,66] at a fixed FAR of 8500 per month.⁴ We compute it by fixing the detection threshold to the corresponding values of SNR ≈ 6.53 and p-score ≈ 0.185 . The injections are then binned by their SNR, and for each bin the fraction of detected over total injections is calculated. We find that our search does not generalize well to very strong signals. The loudest missed signal at this FAR was injected with a SNR of 46.65 which means that our search reaches 100% sensitivity only above SNR 46.65. The search described in Ref. [66] is more sensitive to signals above SNR 25 and saturates already around SNR 35. The algorithm described in Ref. [49] saturates even earlier at SNR 25. For current detectors, on the other hand, most signals are expected to be measured with a low SNR [59]. Within the SNR range covered by the training set (marked gray in Fig. 5), our search is almost 10 times as sensitive when compared to Ref. [66] and still about 4 times as sensitive when compared to Ref. [49].

⁴In Ref. [49], no FAR is stated explicitly. All results are given in terms of a false-alarm probability. We estimate the FAR from this probability by dividing with the step size used to slide the network across long stretches of data. We then rescale it to false alarms per month. The step size was estimated to be 0.3 s to match the interval duration within which the peak amplitude for BNS signals is varied.

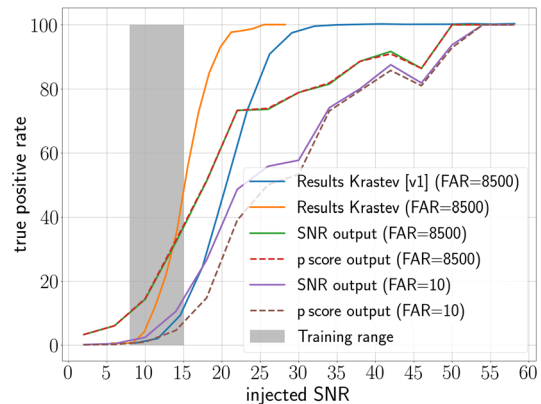


FIG. 5. To compare our search to the work of Ref. [49], we plot their true positive rate at a fixed FAR of 8500 per month in yellow and our true positive rate at the same FAR in green and red. On the x axis, we track the injected optimal network SNR. The blue curve shows the data from Ref. [66], where the results were given in terms of pSNR. We use the conversion $\text{SNR} = 41.2 \cdot \sqrt{2} \cdot \text{pSNR}$. To obtain these curves, we bin the injected signals by their optimal injection SNR and a bin size of 4. For high SNRs, some bins are empty. Empty bins are interpolated linearly from the remaining data. The area marked gray highlights the region covered by the training set. We find that our search performs better for low SNRs but is less sensitive for strong signals. We also show the true positive rate of our search at a FAR of 10 in purple and brown. Within the training range, we find that our search closely matches the true positive rate of Ref. [66] at a higher FAR.

As the authors of Ref. [49] successfully applied curriculum learning, we would expect an increase in sensitivity at high SNRs if the range in our training set were expanded to include high SNR signals.

The plot also shows the true positive rate of our network at a FAR of ten per month, where, within the SNR range of the training set, the SNR output roughly matches the true positive rate of the algorithm proposed in Ref. [66] at an 85 times higher FAR. The network proposed in Ref. [49] is more sensitive to signals with SNR > 8 when compared to our network operating at a FAR of ten per month. One can also observe that at a FAR of ten per month the p-score output is significantly worse over the entire range of injected signals.

Figure 6 shows the recovered SNR against the injected SNR at a fixed FAR of ten per month. For any missed injection, we give the value of the estimated SNR time series that is closest to the injection time. The strongest missed injection at this FAR has a SNR of 50.83. We find that the injected SNR is recovered with a mean squared error of ≈ 181 . Our search is therefore able to distinguish signals from the background, but the estimation of the SNR is uninformative. At this FAR, there are no injections that are detected only in the p-score output. The plot can visually be split into three vertical zones. The lowest zone (red) contains all missed injections. They are recovered

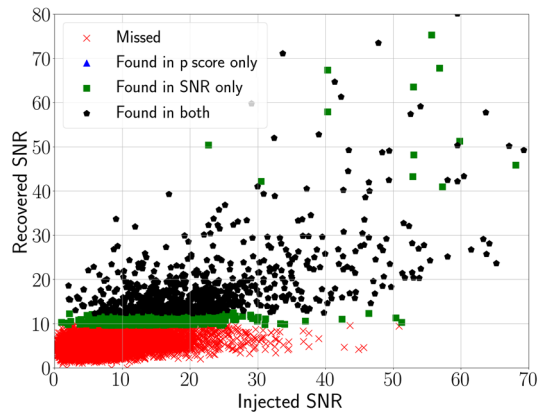


FIG. 6. The plot shows the estimated SNR against the optimal injected SNR of the test set. There are a few injections with SNR > 70 which are not shown here, but all of them are detected. The red cross corresponds to injections that the search did not recover in either of the two outputs at a fixed FAR of ten per month. Injections that are found in the p score output but not in the SNR output would be shown as blue triangles, but no injections of this type exist. Green squares show injections that are found only in the SNR output. Black hexagons represent injections that were found in both outputs. A clear vertical separation can be seen in this figure. We suspect that the network learns an estimate of the SNR internally and maps only the p score to this internal representation. Otherwise, borders would not be so sharp, and some blue triangles should be seen in the red area.

with a SNR below the threshold for the SNR output. If the p-score output was independent of the SNR output, we would expect to find a few blue triangles in this region. The second zone (green) contains injections that are recovered only in the SNR output. The clear separation to the third zone (black) indicates that the p-score output operates very similarly to the SNR output and assigns a value based on the internal SNR estimate. The louder the injected signals are, the more likely the network is to detect it in both outputs.

D. Binary black hole injections

Realistic continuous data will not only contain signals from BNS mergers but also prominently signals from BBH events. It is therefore interesting to test the response of the NN to these kinds of signals.

To do so, we generate a mock dataset containing BBH injections. We use the process described in Sec. II A but adjust the parameter ranges to represent a distribution for BBH signals. The masses are uniformly distributed in the range from 10 to 30 solar masses, the maximal distance is increased to 4000 Mpc to adjust for the louder signals, and the waveform model is changed to SEOBNRv4_opt. As signals from BBHs are within the sensitive band of the detectors for a shorter duration, the average signal separation can be reduced to 20 s with a variance of ± 2 s. The

duration of the sections in the beginning and end of each chunk that do not contain any injections is reduced to 16 s.

As we want only to make a qualitative statement about the sensitivity of our analysis to BBH signals, we generated and evaluated 40960 s ≈ 11 h of mock data, containing 2147 injected signals. The data are processed in the same way as the data containing BNS injections.

For this test set, we find that our network has negligible sensitivity to BBH mergers. The BBH waveforms, which are short compared to BNS signals, are consistently classified as noise.

VI. CONCLUSIONS

We presented a new machine-learning-based search pipeline for BNS signals. To allow our network to efficiently process up to 32 s of data, we introduced multirate sampling, a technique that samples different parts of a GW signal at different sample rates.

Our search improves upon the sensitivity of other machine-learning-based algorithms at low FARs and for signals with a low SNR. For signals with a SNR contained within the SNR boundaries of our training set, we find an improvement of 400% over previous machine-learning-based searches [49].

We probed, for the first time, the sensitivity of a machine-learning-based algorithm at FARs down to 0.5 per month. This enabled a direct comparison to the template-based low-latency search PyCBC Live [18]. We found that our machine-learning-based algorithm is computationally more expensive than using PyCBC Live with a template bank equivalent to our training set. At the same time, the sensitive radius of our search algorithm is lower by a factor of 6. We, therefore, conclude that machine-learning-based search algorithms are not yet sufficiently sensitive or efficient to match the algorithms currently in use.

We do, however, find an improvement in the latency when compared to traditional searches. PyCBC Live introduces on average a latency of 16 s between signal arrival and generating an alert. A very conservative estimate of the latency introduced by our search finds 10.2 s. This value is limited not by the computational cost of applying the network but by processing the data it outputs. Choosing a different algorithm to do so is straightforward and might improve the latency by roughly an order of magnitude. The latency of PyCBC Live can be reduced to a similar duration by increasing the computational cost of the analysis. There are also other search algorithms targeted specifically at low-latency detection of candidate events which are already able to achieve latencies of $\mathcal{O}(1)$ s [19]. The computational cost of all of these searches scales with the size of the template bank used. NNs, on the other hand, have often proven to adapt well to a large variety of features in the input space. It is therefore not unreasonable to believe that machine learning search algorithms may be able to provide

low-latency detections at constant or only slightly increased computational cost when the parameter space is enlarged. We think that this is a strong motivation to further pursue a machine-learning-based search algorithm.

To help compare different search algorithms, we proposed a standardized test procedure that can be applied to neural networks as well. We want to stress the importance of providing FARs and sensitivities for machine-learning-based searches which are derived on as realistic a dataset as possible.

Future works might try to reduce the complexity of the network proposed here to minimize the computational cost and make machine-learning-based searches a viable alternative. Reducing the complexity of the network may also help to improve the sensitivity of the search. Previous works [57] have shown that a network which works well

with simulated noise adapts well to real detector noise if retrained. The algorithm at hand should thus also be extended to be trained and tested on real detector noise. It would further be of interest to test if a computationally less expensive network could be used at a high FAR to be followed up by a matched-filter search with a heavily reduced template bank.

ACKNOWLEDGMENTS

We thank Tobias Blenke, Christoph Dreißigacker, and Tobias Florin for their comments and suggestions. We acknowledge the Max Planck Gesellschaft and the Atlas cluster computing team at Albert-Einstein Institut (AEI) Hannover for support. F. O. was supported by the Max Planck Society's Independent Research Group Program.

-
- [1] B. P. Abbott *et al.* (LIGO Scientific and Virgo Collaborations), *Phys. Rev. Lett.* **116**, 061102 (2016).
 - [2] B. P. Abbott *et al.* (LIGO Scientific and Virgo Collaborations), *Phys. Rev. X* **9**, 031040 (2019).
 - [3] T. Venumadhav, B. Zackay, J. Roulet, L. Dai, and M. Zaldarriaga, *Phys. Rev. D* **101**, 083030 (2020).
 - [4] A. H. Nitz, T. Dent, G. S. Davies, S. Kumar, C. D. Capano, I. Harry, S. Mazzon, L. Nuttall, A. Lundgren, and M. Tápai, *Astrophys. J.* **891**, 123 (2020).
 - [5] A. H. Nitz, C. Capano, A. B. Nielsen, S. Reyes, R. White, D. A. Brown, and B. Krishnan, *Astrophys. J.* **872**, 195 (2019).
 - [6] LIGO Scientific and Virgo Collaborations, Gracedb—Gravitational-wave candidate event database.
 - [7] LIGO Scientific and Virgo Collaborations, [arXiv:2004.08342](https://arxiv.org/abs/2004.08342).
 - [8] B. P. Abbott, R. Abbott, T. D. Abbott, S. Abraham, F. Acernese, K. Ackley, C. Adams, R. X. Adhikari, V. B. Adya, C. Affeldt *et al.*, *Astrophys. J.* **892**, L3 (2020).
 - [9] R. Abbott *et al.*, *Astrophys. J.* **896**, L44 (2020).
 - [10] R. Abbott *et al.* (LIGO Scientific and Virgo Collaborations), *Phys. Rev. Lett.* **125**, 101102 (2020).
 - [11] Y. Aso, Y. Michimura, K. Somiya, M. Ando, O. Miyakawa, T. Sekiguchi, D. Tatsumi, and H. Yamamoto (KAGRA Collaboration), *Phys. Rev. D* **88**, 043007 (2013).
 - [12] T. Akutsu *et al.*, [arXiv:2005.05574](https://arxiv.org/abs/2005.05574).
 - [13] B. P. Abbott *et al.* (LIGO Scientific, Virgo Collaboration, and KAGRA Collaboration), *Living Rev. Relativity* **21**, 3 (2018).
 - [14] LIGO Scientific and Virgo Collaborations, Online pipelines (2018), <https://emfollow.docs.ligo.org/userguide/analysis/searches.html>.
 - [15] M. Maggiore, *Gravitational Waves* (Oxford University Press, Oxford, 2008).
 - [16] S. Sachdev *et al.*, [arXiv:1901.08580](https://arxiv.org/abs/1901.08580).
 - [17] T. Adams, D. Buskulic, V. Germain, G. M. Guidi, F. Marion, M. Montani, B. Mours, F. Piergiovanni, and G. Wang, *Classical Quantum Gravity* **33**, 175012 (2016).
 - [18] A. H. Nitz, T. Dal Canton, D. Davis, and S. Reyes, *Phys. Rev. D* **98**, 024050 (2018).
 - [19] S. Hooper, S. K. Chung, J. Luan, D. Blair, Y. Chen, and L. Wen, *Phys. Rev. D* **86**, 024012 (2012).
 - [20] T. D. Canton and I. W. Harry, [arXiv:1705.01845](https://arxiv.org/abs/1705.01845).
 - [21] I. Harry, S. Privitera, A. Bohé, and A. Buonanno, *Phys. Rev. D* **94**, 024012 (2016).
 - [22] M. Hannam, P. Schmidt, A. Bohé, L. Haegel, S. Husa, F. Ohme, G. Pratten, and M. Pürrer, *Phys. Rev. Lett.* **113**, 151101 (2014).
 - [23] S. Khan, K. Chatziioannou, M. Hannam, and F. Ohme, *Phys. Rev. D* **100**, 024059 (2019).
 - [24] Y. Pan, A. Buonanno, A. Taracchini, L. E. Kidder, A. H. Mroué, H. P. Pfeiffer, M. A. Scheel, and B. Szilágyi, *Phys. Rev. D* **89**, 084006 (2014).
 - [25] I. Harry, J. C. Bustillo, and A. Nitz, *Phys. Rev. D* **97**, 023004 (2018).
 - [26] L. London, S. Khan, E. Fauchon-Jones, C. García, M. Hannam, S. Husa, X. Jiménez-Forteza, C. Kalaghatgi, F. Ohme, and F. Pannarale, *Phys. Rev. Lett.* **120**, 161102 (2018).
 - [27] S. Khan, F. Ohme, K. Chatziioannou, and M. Hannam, *Phys. Rev. D* **101**, 024056 (2020).
 - [28] R. Cotesta, A. Buonanno, A. Bohé, A. Taracchini, I. Hinder, and S. Ossokine, *Phys. Rev. D* **98**, 084028 (2018).
 - [29] A. H. Nitz, A. Lenon, and D. A. Brown, *Astrophys. J.* **890**, 1 (2020).
 - [30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *Int. J. Comput. Vis.* **115**, 211 (2015).
 - [31] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, [arXiv:1609.03499](https://arxiv.org/abs/1609.03499).
 - [32] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou,

- V. Panneershelvam, M. Lanctot *et al.*, *Nature (London)* **529**, 484 (2016).
- [33] Openai five (2018).
- [34] S. Bahaadini, V. Noroozi, N. Rohani, S. Coughlin, M. Zevin, J. Smith, V. Kalogera, and A. Katsaggelos, *Information Sciences (NY)* **444**, 172 (2018).
- [35] C. Dreissigacker, R. Sharma, C. Messenger, R. Zhao, and R. Prix, *Phys. Rev. D* **100**, 044009 (2019).
- [36] W. Wei and E. A. Huerta, *Phys. Lett. B* **800**, 135081 (2020).
- [37] E. Cuoco *et al.*, [arXiv:2005.03745](https://arxiv.org/abs/2005.03745).
- [38] S. R. Green, C. Simpson, and J. Gair, [arXiv:2002.07656](https://arxiv.org/abs/2002.07656).
- [39] H. Gabbard, C. Messenger, I. S. Heng, F. Tonolini, and R. Murray-Smith, [arXiv:1909.06296](https://arxiv.org/abs/1909.06296).
- [40] J. P. Marulanda, C. Santa, and A. E. Romano, [arXiv:2004.01050](https://arxiv.org/abs/2004.01050).
- [41] M. L. Chan, I. K. Siong Heng, and C. Messenger, [arXiv:1912.13517](https://arxiv.org/abs/1912.13517).
- [42] A. Iess, E. Cuoco, F. Morawski, and J. Powell, *Mach. Learn. Sci. Technol.* **1**, 025014 (2020).
- [43] D. George and E. A. Huerta, *Phys. Rev. D* **97**, 044039 (2018).
- [44] H. Gabbard, M. Williams, F. Hayes, and C. Messenger, *Phys. Rev. Lett.* **120**, 141103 (2018).
- [45] B. P. Abbott *et al.*, *Classical Quantum Gravity* **33**, 134001 (2016).
- [46] L. K. Nuttall, *Phil. Trans. R. Soc. A* **376**, 20170286 (2018).
- [47] M. Cabero, A. Lundgren, A. H. Nitz, T. Dent, D. Barker, E. Goetz, J. S. Kissel, L. K. Nuttall, P. Schale, R. Schofield, and D. Davis, *Classical Quantum Gravity* **36**, 155010 (2019).
- [48] T. D. Gebhard, N. Kilbertus, I. Harry, and B. Schölkopf, *Phys. Rev. D* **100**, 063015 (2019).
- [49] P. G. Krastev, *Phys. Lett. B* **803**, 135330 (2020).
- [50] M. B. Schäfer, F. Ohme, and A. H. Nitz, Data release: Detection of gravitational-wave signals from binary neutron star mergers using machine learning, <https://github.com/gwastro/bns-machine-learning-search> (2020).
- [51] S. A. Usman *et al.*, *Classical Quantum Gravity* **33**, 215004 (2016).
- [52] A. Nitz *et al.*, gwastro/pycbc: Pycbc release v1.13.5 (2019), <http://dx.doi.org/10.5281/zenodo.2581446>.
- [53] S. Droz, D. J. Knapp, E. Poisson, and B. J. Owen, *Phys. Rev. D* **59**, 124016 (1999).
- [54] L. Blanchet, *Living Rev. Relativity* **5**, 3 (2002).
- [55] G. Faye, S. Marsat, L. Blanchet, and B. R. Iyer, *Classical Quantum Gravity* **29**, 175004 (2012).
- [56] Ligo Scientific Collaboration, LIGO Algorithm Library—LALSuite, free software (GPL), (2018).
- [57] D. George and E. Huerta, *Phys. Lett. B* **778**, 64 (2018).
- [58] M. B. Schäfer, Analysis of gravitational-wave signals from binary neutron star mergers using machine learning (2019), <http://dx.doi.org/10.15488/7467>.
- [59] B. F. Schutz, *Classical Quantum Gravity* **28**, 125023 (2011).
- [60] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).
- [61] S. Bai, J. Z. Kolter, and V. Koltun, [arXiv:1803.01271](https://arxiv.org/abs/1803.01271).
- [62] A. Schmitt, K. Fu, S. Fan, and Y. Luo, in *Proceedings of the 2nd International Conference on Computer Science and Software Engineering*, CSSE 2019 (ACM, New York, 2019), pp. 73–78.
- [63] M. Lin, Q. Chen, and S. Yan, [arXiv:1312.4400](https://arxiv.org/abs/1312.4400).
- [64] F. Chollet *et al.*, Keras, <https://keras.io> (2019).
- [65] A. H. Nitz, T. Dent, T. D. Canton, S. Fairhurst, and D. A. Brown, *Astrophys. J.* **849**, 118 (2017).
- [66] P. G. Krastev, [arXiv:1908.03151v1](https://arxiv.org/abs/1908.03151v1).

Correction: The license statement contained an error and has been fixed.