Deep Residual Networks for Gravitational Wave Detection

Paraskevi Nousi,1 Alexandra E. Koloniari,2 Nikolaos Passalis,1 Panagiotis Iosif,2 Nikolaos Stergioulas,2 and Anastasios Tefas1

1Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
2Department of Physics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

(Dated: June 30, 2023)

Traditionally, gravitational waves are detected with techniques such as matched filtering or unmodeled searches based on wavelets. However, in the case of generic black hole binaries with non-aligned spins, if one wants to explore the whole parameter space, matched filtering can become impractical, which sets severe restrictions on the sensitivity and computational efficiency of gravitational-wave searches. Here, we use a novel combination of machine-learning algorithms and arrive at sensitive distances that surpass traditional techniques in a specific setting. Moreover, the computational cost is only a small fraction of the computational cost of matched filtering. The main ingredients are a 54-layer deep residual network (ResNet), a Deep Adaptive Input Normalization (DAIN), a dynamic dataset augmentation, and curriculum learning, based on an empirical relation for the signal-to-noise ratio. We compare the algorithm’s sensitivity with two traditional algorithms on a dataset consisting of a large number of injected waveforms of non-aligned binary black hole mergers in real LIGO O3a noise samples. Our machine-learning algorithm can be used in upcoming rapid online searches of gravitational-wave events in a sizeable portion of the astrophysically interesting parameter space. We make our code, AResGW, and detailed results publicly available at https://github.com/vivinousi/gw-detection-deep-learning

I. INTRODUCTION

In the first three observing runs (O1-O3) of the LIGO-Virgo Collaboration [1,2] (recently joined by Kagra [3]) a number of \( O(90) \) confident gravitational wave (GW) detections were found in the data, including mostly binary black holes (BBH), but also a few binary neutron stars (BNS) and neutron star black hole systems (NSBH) [4–6]. The anticipated significant increase in the number of detections during the fourth observing run (O4), which is scheduled to start in spring of 2023 and even more so during O5 and the observing runs of the planned 3rd-generation detectors (e.g. Cosmic Explorer [7] and Einstein Telescope [8]) will make the application of traditional matched-filtering techniques increasingly costly or impractical [2]. This is both for reasons of computational efficiency, as the goal is to obtain near real-time detection triggers, as well as for reasons of accuracy, since the confident detection of near-threshold systems with random spin directions requires a much larger parameter space than the aligned-spin case. The situation will become even more challenging, if template banks with departures from general relativity (GR) will be included. Unmodeled search algorithms, on the other hand, naturally have a limited sensitivity, depending on the particular GW source.

An attractive solution to the above problem that has been investigated in the last few years is the implementation of machine-learning (ML) methods, such as convolutional neural networks (CNN) or auto-encoders, see e.g. schafer2022training [10–30] and [31] for a review, but it has been difficult to evaluate the effectiveness of such efforts in a realistic setting. Recently, the first Machine-Learning Gravitational-Wave Mock Data Challenge (MLGWSC-1) was completed [32], defining an objective framework for testing the sensitivity and efficiency of ML algorithms on modeled injections in both Gaussian and real O3a detector noise, in comparison to traditional algorithms. Here, we present the leading ML algorithm in the case of injections of BBH template waveforms in real O3a noise and show that with further improvements it surpasses, for the first time, the results obtained with standard configurations of traditional algorithms in this specific setting. This is achieved for a component mass range between 7 – 50\( M_\odot \) (which corresponds to 70% of the announced events in the cumulative GWTC catalog [6]) and a relatively low false-alarm rate (FAR) as small as one per month. We thus demonstrate that our ML algorithm is sufficiently mature to be implemented in GW search pipelines.

A key indicator for the effectiveness of a GW search algorithm is the sensitive distance it can achieve at a given FAR [14]. Our ML algorithm is a novel combination of several key ingredients, each of which boosts the sensitive distance to higher values. The base algorithm is a 54-layer one-dimensional (1D) deep residual network (ResNet) [33], which allows for training of a much deeper network, in comparison to the simpler CNN. The second important ingredient that allowed the algorithm to work well at small FAR is the addition of a Deep Adaptive Input Normalization (DAIN) [34], to deal with the non-stationary nature of real O3a noise. Next, we obtained improvements by dynamically augmenting the dataset during training. The execution speed was significantly increased with the implementation of a framework-specific, module-based whitening layer, which computes the power
spectral density (PSD) over a period of a few seconds in batched tensor format. Finally, we saw a large boost in sensitive distance employing curriculum learning, in which the network was learning the injected waveforms with highest signal-to-noise ratio (SNR) first.

The network was implemented in PyTorch and its training (including validation) on 12 days of training data was achieved within 31 hours on an A6000 GPU with tensor cores (for 14 epochs). The runtime for the evaluation was achieved within 31 hours on an A6000 GPU with ten-
and injections with merger times separated by a random time between 24 s to 30 s (which translates to about 96k injections over the whole duration). The code for creating the different datasets was provided by the organizers of MLGWSC-1 and is publicly available. For the main test dataset we used the same random seed of 2514409456 and offset of 0 (first 4 weeks) as in [32]. We created additional test datasets with the same total duration, but different random seeds and offset, in order to assess the variance of our main results.

III. COMPUTATIONAL METHODS

A. Data Preprocessing and Normalization

Following approaches from traditional matched filtering as applied in GW detections, we first whiten the training data [10, 37]. As is common in deep learning (DL) methods, we then use an input normalization layer [34, 38], before feeding the data to the neural network for classification.

Whitening. The data segments were whitened using the Welch method [39] for computing the PSD and an inverse-spectrum truncation method [40] for smoothing. Both methods were implemented in PyTorch [35], to allow for batch processing of multiple input segments in a single forward pass, which gave a significant speed up with respect to a CPU implementation. First, we crop 4.25s segments from the original timeseries, for both the training and validation sets, with a stride of 1.4s. An injection is done in a random quarter segment (of 1.25s duration), which is whitened using the 4.25s noise segment surrounding it. After whitening, the first and last 0.125s (0.25s in total) are removed from each sample, leaving a 1 s sample as input for the neural network.

Adaptive Normalization. As the real noise of the H1 and L1 detectors is non-stationary and affected by a multitude of factors, we opted for an adaptive input normalization method, specifically Deep Adaptive Input Normalization (DAIN) [34], which is applied before feeding the data to the network. DAIN has been successfully used in tasks that involve non-stationary timeseries, such as financial timeseries forecasting [11].

The goal of this normalization layer is to learn how the input timeseries should be normalized. DAIN is used before feeding the input to the subsequent layers and it is trained by back-propagating the network’s gradients to its parameters, as shown in Fig. 2. Furthermore, DAIN differs from other normalization schemes, since it can dynamically adjust the applied normalization scheme based on the input at inference time, allowing for handling non-stationary data. More specifically, DAIN aims to learn how the measurements $x \in \mathbb{R}^{2048 \times 2}$ fed to the neural network should be normalized by appropriately shifting and scaling them:

$$\tilde{x}_j = (x_j - \alpha) \odot \beta,$$

where $x_j \in \mathbb{R}^2$ refers to the $j$-th observation (out of 2048 included in the current window), $\odot$ is the Hadamard (entrywise) division operator and $\beta$ is a scaling operator. To this end, we first build a summary representation of the current window as:

$$a = \frac{1}{2048} \sum_{j=1}^{2048} x_j \in \mathbb{R}^2.$$  \hspace{1cm} (3)

This average is used to estimate the mode of the distribution that generated the observed data. Then, DAIN learns how to appropriate shift the data based on the observed mode by estimating the value for the shifting operator $\alpha$ as:

$$\alpha = W_\alpha a \in \mathbb{R}^2,$$  \hspace{1cm} (4)

where $W_\alpha \in \mathbb{R}^{2 \times 2}$ are the trainable parameters of the shifting operator.

| residual blocks | filters | strided input $D$ |
|-----------------|---------|-------------------|
| 4               | 8       | $2 \times 2048$   |
| 1               | 16      | $8 \times 2048$   |
| 2               | 16      | $16 \times 1024$  |
| 1               | 32      | $16 \times 1024$  |
| 2               | 32      | $32 \times 512$   |
| 1               | 64      | $32 \times 512$   |
| 2               | 64      | $64 \times 256$   |
| 1               | 64      | $64 \times 256$   |
| 2               | 64      | $64 \times 128$   |
| 1               | 64      | $64 \times 128$   |
| 2               | 64      | $64 \times 64$    |
| 5               | 32      | $64 \times 64$    |
| 3               | 16      | $32 \times 64$    |

TABLE I. Feature extraction backbone of our ResNet54-like architecture. The first column lists the number of residual blocks, totalling 27, each consisting of two convolutional layers. The second column lists the number of filters in each corresponding block. The third column indicates those blocks that are 2-strided (see text for details) and the last column displays the dimensionality $D$ of the input tensor for each block.

FIG. 2. Deep Adaptive Normalization (DAIN) is applied before feeding the data to the employed DL model. DAIN involves three adaptive normalization steps, i.e., adaptive shifting, adaptive scaling and adaptive gating, increasing the ability of DL models to handle non-stationary data.
After this shifting process, the data are also scaled by employing the scaling operator $\beta$, as shown in \( \text{(2)} \). Again, we calculate a summary representation as:

$$b_k = \sqrt{\frac{1}{2048} \sum_{j=1}^{2048} (x_{j,k} - \alpha_k)^2}, \quad k = 1, 2,$$

and then define the scaling operator as:

$$\beta = W_j b \in \mathbb{R}^2,$$

where $W_b \in \mathbb{R}^{2 \times 2}$ are its parameters.

Finally, the shifted and scaled observations are fed to an adaptive gating layer which aims to appropriately modulate the features according to their usefulness for the task at hand as:

$$\tilde{x}_j = \tilde{x}_j \odot \gamma,$$

where $\odot$ is the Hadamard (entrywise) multiplication operator and

$$\gamma = \text{sign}(W_c, c + d) \in \mathbb{R}^2,$$

$\text{sign}(x) = 1/(1 + \exp(-x))$ is the sigmoid function, $W_c \in \mathbb{R}^{2 \times 2}$ and $d \in \mathbb{R}^2$ are the parameters of the gating layer that are learned through back-propagation. The updated summary representation $c$ is calculated as:

$$c = \frac{1}{2048} \sum_{j=1}^{2048} \tilde{x}_j \in \mathbb{R}^2.$$

The non-linearity introduced by this layer allows for the suppression of the normalized features during inference. This can help to reduce the effect of features that could harm the generalization abilities of the network. The parameters introduced by the DAIN layer are fitted using gradient descent:

$$\Delta \left( W_a, W_b, W_c, d, W \right) = -\eta_a \frac{\partial L}{\partial W_a},$$

$$\eta_b \frac{\partial L}{\partial W_b}, \eta_c \frac{\partial L}{\partial W_c}, \eta_d \frac{\partial L}{\partial d}, \eta_\gamma \frac{\partial L}{\partial \gamma}$$

where $L$ denotes the final loss function of the network and $W$ its weights. Separate learning rates can be used for the parameters of each sub-layer, i.e. $\eta_a$, $\eta_b$, and $\eta_c$, if necessary, to ensure the stability of the training process.

### B. Deep Residual Networks

Residual neural networks $\text{i}3$ use so-called skip connections to improve training, by allowing gradients to better reach the earliest layers of a neural network architecture, effectively solving the vanishing gradient problem $\text{i2}$ and leading to more effective training as the number of layers increases, especially when paired with carefully designed training methods $\text{i4}$. This allows for much deeper networks to be trained, in comparison to simple CNN. Depth in neural networks has been linked to higher levels of semantic information, and better performance in tasks like image recognition $\text{i41}$, detection $\text{i15}$, etc.

We designed a deep residual network, based on 1D convolutions, for the purpose of the binary classification of 1s long (i.e. $2 \times 2048$-dimensional) segments into positive (containing an injection) or negative (pure noise) segments. Our network has a depth of 54 layers, grouped into 27 blocks comprising two convolutional layers with a varying number of filters. The network’s backbone (except for the final two convolutional layers) is summarized in Table $\text{1}$, where $D$ denotes the dimensionality of the input tensor. Blocks 5, 8, 11, 14 and 17 are 2-strided, meaning that dimensionality is halved and an additional layer is used in the residual connection.

A graphical depiction of the network architecture is displayed in Figure $\text{3}$. The output of a residual block of layers can be written schematically as:

$$g = f(x) + h(x),$$

where $f(x)$ is a block of two convolutional layers, and $x$ the input (i.e. the $2 \times 2048$-dimensional timeseries) to its first layer. The residual function $h(x)$ is either the identity function, $h(x) = x$, or a strided convolutional layer. Each convolutional layer is followed by batch normalization and a ReLU activation function. Two final

---

$^2$ We remind that the sampling rate was 2048Hz and there are two channels, one for each of the aLIGO detectors.
individual convolutional layers gradually reduce the output to a binary outcome (noise plus injected waveform vs. noise only).

C. Training

The network is fed with the training dataset (925k 1s segments of noise and an equal number of 1s segments containing injections) using a mini-batch size of 400 (each mini-batch also contains an equal number of only noise and noise plus injection segments, randomly chosen from the whole dataset). The Adam optimizer\(^3\) was used during backpropagation and the regularized binary cross entropy (a variant of the cross entropy loss function that is designed to be finite\(^2\)) was used as the objective function.

Dynamic augmentation. During training, we also employed dynamic augmentation, by randomly replacing a noise segment for the L1 channel, by another segment. This augmentation is performed with a random probability of 40% and virtually increases the total number of samples that the network sees during training.

SNR-based Curriculum Learning. We implemented a learning strategy, so that the network is first trained on the loudest injections and only at later epochs it is trained on weaker injections. To achieve this, we used

\[ \text{SNR} = \frac{1261 \text{ Mpc}}{D} \left( \frac{M_\text{c}}{M_\odot} \right)^{5/7} \left[ 0.7 + 0.3 \cos(2 \varpi) \right]. \]

Note that the first two training epochs are a warm up period, during which the learning rate is annealed from $10^{-8}$ to $4.5 \times 10^{-3}$.

\(^3\) Note that the first two training epochs are a warm up period, during which the learning rate is annealed from $10^{-8}$ to $4.5 \times 10^{-3}$.

\(^2\) The Adam optimizer is a method for stochastic optimization that is often used in deep learning. It is an extension of the stochastic gradient descent algorithm that uses adaptive learning rates for different weights in the network.

\(^1\) The regularized binary cross entropy is a loss function used in binary classification problems to handle situations where the class distribution is highly imbalanced.

---

**FIG. 4.** Comparison of the SNR histogram computed with the empirical relation Eq. (13) to the optimal SNR of the injections (using a random subset of 10000 injections). A similar distribution is obtained. The shaded areas correspond to the restricted values of the SNR used in the first eight epochs of the learning strategy (from right to left, see Table II).

**FIG. 5.** Histogram of the errors between the SNR computed with the empirical relation Eq. (13) to the optimal SNR of the same injections as in Fig. 4. The errors follow a normal distribution with standard deviation $\sigma = 1.1$. The optimal signal-to-noise ratio (SNR) of the injected signal

\[ \text{SNR} = 2 \sqrt{\int_0^\infty df |\hat{h}(f)|^2 \frac{S_n(f)}{S_n(f)}}, \]

where $\hat{h}(f)$ is the amplitude of the Fourier transform of the injected signal and $S_n(f)$ is the PSD of the detector’s noise. An accurate computation of the SNR of each segment during training would add some computational overhead, so we chose to construct an empirical relation that gives an approximate prediction of the SNR. Specifically, by varying only the chirp mass, the distance and the inclination $\varpi$ (and setting all other parameters to some fixed values), we arrived at the following approximate empirical relation:

\[ \text{SNR} = \frac{1261 \text{ Mpc}}{D} \left( \frac{M_\text{c}}{M_\odot} \right)^{5/7} \left[ 0.7 + 0.3 \cos(2 \varpi) \right]. \]
TABLE II. Learning strategy: During the first 4 epochs the network learns only strong signals with estimated optimal SNR larger than 7.63. Next, the network gradually learns weaker signals (for the subsequent epochs the estimated SNR range is shown). After the tenth epoch, the network learns all signals.

| epochs | min(SNR) | max(SNR) |
|--------|----------|----------|
| 4      | 7.63     | 100      |
| 2      | 3.63     | 100      |
| 2      | 1        | 3.63     |
| 2      | 1        | 7.63     |
| remaining | 0    | ∞        |

FIG. 6. Loss as a function of epoch. Due to the adopted learning strategy (strong signals first), initial losses are small. After ten epochs, when the network is learning all signals, the loss of the training set converges with that of the validation set.

in Table II and the first eight epochs are also depicted in Fig. 4 in relation to the distribution of the SNR.

Fig. 6 shows the evolution of the loss function as a function of the training epoch. Due to the adopted learning strategy, initial losses are small. When the network is finally learning all signals (after ten epochs), the loss of the training set converges with that of the validation set. The corresponding evolution of the accuracy as a function of the training epoch is shown in Fig. 7.

FIG. 7. Accuracy as a function of epoch. Due to the adopted learning strategy (strong signals first), initial accuracy is high. After ten epochs, when the network is learning all signals, the accuracy of the training set converges with that of the validation set.

FIG. 8. Representative example of the ranking statistic for overlapping segments (with a stride of 0.1s). Segments with \( R < 0.5 \) are classified as negatives. Segments with \( R > 0.5 \) that cluster within 0.3s of a known injection at time \( t_0 \) are reported as a single true positive.

IV. DETECTION OF BBH INJECTIONS IN REAL NOISE

The trained network is run on segments of the test dataset described in Sec. II and produces a binary output, corresponding to the probabilities that the waveform is present or that the segment contains only noise. The first output (waveform present) serves as a ranking statistic \( R \) with values between 0 and 1 for each 1s segment. When \( R > 0.5 \), a positive outcome is recorded.

During deployment on the test dataset, we crop an amount of 4.25 seconds of data, from which we compute the PSD in real time. To increase speed, the Welch method for computing the PSD was implemented in PyTorch [35], whereas whitening is implemented as the first layer of the final detection module. Then, the input is reorganized as a batch of 1.25s segments with a stride of 0.1s, i.e., with an overlap of 1.15s. This batch is whitened with a single forward pass of the implemented whitening module. The last valid segment starts at the 3s mark, which leads to 31 segments of duration 1.25s. After whitening, the 1.25s whitened segments are cropped unilaterally to 1s total duration. Notice that when an injection is present, multiple overlapping segments can have \( R > 0.5 \). Because of this, we cluster positives and report any positives that are detected within a time span of 0.3s as a single detection (see Fig. 8 for a representative example).

After the deployment, the output is evaluated every 0.1s comparing it to the known injection times in the test dataset. If a positive output differs less than 0.3s from the nominal merger time for a particular injection, then it is classified as a true positive, otherwise as a false positive.
FIG. 9. Sensitive distance vs. false alarm rate for three different input normalizations: DAIN [34], Batch Normalization (BN) [38] and Adaptive Batch Normalization (ADA) [47] (for a shorter test dataset of duration of one day). In contrast to BN and ADA, DAIN maintains a good sensitive distance down to the lowest FAR.

To evaluate the effectiveness of the search algorithm, we first need to determine the false alarm rate \( F \) as a function of the ranking statistic, i.e. the function \( F(R) \). Next, the sensitive distance of the search is determined as a function of the ranking statistic, which finally produces a relation between the sensitive distance and the \( F \) (see [32] for definitions and the detailed methodology).

In Fig. 9 we demonstrate the importance of using DAIN [34] as the input normalization, by comparing the sensitive distance at different FAR (for a shorter test dataset of duration of one day) to the corresponding results obtained with either Batch Normalization (BN) [38] or Adaptive Batch Normalization (ADA) [47]. Whereas the three methods achieve similar sensitive distances at \( \text{FAR} > 30/d \), the performance of BN and ADA deteriorates, dropping to 0 at FAR of several per day. In contrast, the input normalization with DAIN allows the network to maintain a good sensitive distance down to the lowest FAR (notice that for the comparison shown in Fig. 5 we did not activate the SNR-based curriculum learning, to isolate the effect of the input normalization).

Fig. 10 displays the sensitive distance as a function of FAR for our best model (ResNet54d+SNR), in comparison with the simpler setup (ResNet54) used in [32] and two widely used algorithms for GW detection, Coherent WaveBurst (cWB) and PyCBC (the latter using aligned-spin templates only, see text for details). All codes were run on the same test dataset established in [32]. Our best model surpasses the performance of the other algorithms at all FAR in this setting (notice that the PyCBC run was based on a template bank with only aligned-spin waveforms).

Recently enhanced with machine-learning techniques [53] (for details on the particular cWB setup used on the test dataset, see [32]), PyCBC implements matched filtering of waveform templates, but in [32] only aligned-spin templates were used (for more general waveforms, the method could become computationally too costly). Since the injections in the test dataset are based on more general waveforms, this particular PyCBC search cannot reach ideal fitting factors, leaving room for other algorithms to surpass it.

As seen in Fig. 10, our best model, which includes the SNR-based curriculum learning, surpasses the PyCBC results at all FAR. The sensitive distance as a function of FAR is nearly level at the lowest FAR of 1 per month, indicating that our algorithm may maintain good performance even when extended to lower FAR. Our best model also significantly exceeds the sensitivity of the unmodeled cWB search.

V. SUMMARY AND DISCUSSION

We used a novel combination of ML algorithms and arrived at sensitive distances for injected BBH GW signals that surpass traditional algorithms, even at small false alarm rates. The main ingredients are a 54-layer deep residual network (ResNet), a Deep Adaptive Input Normalization (DAIN), a dynamic dataset augmentation, and curriculum learning, based on an empirical relation for the signal-to-noise ratio. Our best ML model surpasses the sensitive distance achieved with traditional algorithms in a specific setting that uses a dataset consisting of a large number of injected non-aligned spin waveforms in real LIGO O3a noise samples. The matched-

---

4 This setup has half the number of filters, as was the case in [32], and does not include the SNR-based curriculum learning.
filtering PyCBC run on the same dataset included only aligned-spin templates and could therefore not reach optimal sensitivity.

We examined the variance of our main result using additional test datasets with different offsets and starting times, see Fig. 11. The different sensitivity curves we obtain are within a few percent from the average, demonstrating that our trained neural network runs robustly on differently datasets, with consistent sensitivity.

Our ML model operates at a fraction of the computational cost of matched filtering (see [32] for detailed comparisons) and it can thus be deployed in upcoming rapid online searches of gravitational-wave events in a sizeable portion of the astrophysically interesting parameter space.

The performance of our ML model could be further improved using a more accurate empirical relation for the signal-to-noise ratio and a more fine-tuned curriculum learning. We are planning to extend the training dataset to include lower or higher black hole masses. It is likely that dedicated networks will need to be trained to cover the edges of the parameter space. Furthermore, we are planning to include additional channels for the operating advanced Virgo [2] and Kagra [3] detectors. In the future, additional detectors, such as the planned LIGO-India [54] detector could be included in the training of the network.

Our code, AResGW, and detailed results are publicly available at [55].

A. Acknowledgments

We are grateful to Agata Trovato, Marlin Schaefer and Ondrej Zelenka for useful comments on the manuscript. We are indebted to the organizers of the MLGWSC-1 challenge, for all their work and for providing routines for creating different datasets. We acknowledge the support provided by the IT Center of the Aristotle University of Thessaloniki (AUTH) as our results have been produced using the AUTH High Performance Computing Infrastructure and Resources. We also acknowledge support from COST Action (European Cooperation in Science and Technology) CA17137 (G2Net).

This research has made use of data or software obtained from the Gravitational Wave Open Science Center (gw-openscience.org), a service of LIGO Laboratory, the LIGO Scientific Collaboration, the Virgo Collaboration, and KAGRA. LIGO Laboratory and Advanced LIGO are funded by the United States National Science Foundation (NSF) as well as the Science and Technology Facilities Council (STFC) of the United Kingdom, the Max-Planck-Society (MPS), and the State of Niedersachsen/Germany for support of the construction of Advanced LIGO and construction and operation of the GEO600 detector. Additional support for Advanced LIGO was provided by the Australian Research Council. Virgo is funded, through the European Gravitational Observatory (EGO), by the French Centre National de Recherche Scientifique (CNRS), the Italian Istituto Nazionale di Fisica Nucleare (INFN) and the Dutch Nikhef, with contributions by institutions from Belgium, Germany, Greece, Hungary, Ireland, Japan, Monaco, Poland, Portugal, Spain. The construction and operation of KAGRA are funded by Ministry of Education, Culture, Sports, Science and Technology (MEXT), and Japan Society for the Promotion of Science (JSPS), National Research Foundation (NRF) and Ministry of Science and ICT (MSIT) in Korea, Academia Sinica (AS) and the Ministry of Science and Technology (MoST) in Taiwan.

[1] J. Aasi et al. (LIGO Scientific Collaboration), Advanced LIGO, Classical and Quantum Gravity 32, 074001 (2015).
[2] F. Acernese et al. (Virgo Collaboration), Advanced Virgo: a second-generation interferometric gravitational wave detector, Classical and Quantum Gravity 32, 024001 (2014).
[3] T. Akutsu et al., Overview of KAGRA: Detector design and construction history, Progress of Theoretical and Experimental Physics 2021, 10.1093/ptep/ptaa125 (2020), 05A101.
[4] B. P. Abbott et al. (LIGO Scientific Collaboration and Virgo Collaboration), Gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs, Phys. Rev. X 9, 031040 (2019).
