SoftwareX 26 (2024) 101639

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

PycWB: A user-friendly, Modular, and python-based framework for gravitational wave unmodelled search

Yumeng Xu [a], Shubhanshu Tiwari [a,*], Marco Drago [b,c]

[a] Physik-Institut, University of Zurich, Winterthurerstrasse 190, 8057, Zürich, Switzerland
[b] Dipartimento di Fisica, Università di Roma "La Sapienza", Piazzale Aldo Moro 2, I-00185 Roma, Italy
[c] INFN, Sezione di Roma, Piazzale Aldo Moro 2, I-00185 Roma, Italy

ARTICLE INFO

Keywords:
Gravitational waves
Burst search

ABSTRACT

Unmodelled searches and reconstruction is a critical aspect of gravitational wave data analysis, requiring sophisticated software tools for robust data analysis. This paper introduces `PycWB`, a user-friendly and modular Python-based framework developed to enhance such analyses based on the widely used unmodelled search and reconstruction algorithm Coherent Wave Burst (`cWB`). The main features include a transition from C++ scripts to YAML format for user-defined parameters, improved modularity, and a shift from complex class-encapsulated algorithms to compartmentalized modules. The `PycWB` architecture facilitates efficient dependency management, better error-checking, and the use of parallel computation for performance enhancement. Moreover, the use of Python harnesses its rich library of packages, facilitating post-production analysis and visualization. The `PycWB` framework is designed to improve the user experience and accelerate the development of unmodelled gravitational wave analysis.

Code metadata

Current code version	v0.17.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00558
Permanent link to Reproducible Capsule	
Legal Code License	GNU General Public License
Code versioning system used	git
Software code languages, tools, and services used	C++, python, Javascript, HTML
Compilation requirements, operating environments & dependencies	gcc, ROOT, wheel, setuptools
If available Link to developer documentation/manual	https://yumeng.xu.docs.ligo.org/pycwb
Support email for questions	yumeng.xu@ligo.org

1. Motivation and significance

The choice of programming language significantly influences the design and usage of scientific software. The benefits of having a Python-based software or Python interface for critical software in gravitational waves (GW) data analysis are outlined in [1,2]. Python, as of now, is on its way to becoming the default programming language in GW data analysis. This statement can be corroborated by the emergence of Python-based gravitational waveform models like `pySEOBNR`[2], `gwsurrogate`[3], inference software like `BILBY`[4], `PyCBC-inference`[5], gravitational wave background search software like `pygwb`[6], and the success and wide usage of GW data analysis algorithms like `PyCBC` [7].

Despite these advancements, there remain several opportunities where Python-based software can accelerate the usage and development of GW data analysis algorithms. One specific example is the so-called unmodelled search and waveform reconstruction algorithms in GW data analysis. The lack of readily available Python-based open-source software restricts the development and usage of un-modelled

---

* Corresponding author.
E-mail address: shubhanshu.tiwari@physik.uzh.ch (Shubhanshu Tiwari).

https://doi.org/10.1016/j.softx.2024.101639
Received 29 August 2023; Received in revised form 8 December 2023; Accepted 18 January 2024

algorithms, limiting it primarily to researchers proficient in languages like C/C++ which is a low-level programming language. Creating Python-based solutions and interfaces will enhance participation and development in the field.

The Coherent Wave Burst (`cWB`) algorithm has been at the forefront of advancements in GW astrophysics [8]. The range of applicability of `cWB` for GW transient data analysis is very wide as it is an all-sky morphology-independent algorithm i.e. it does not rely on the waveform models or the sky direction of the source. Instead `cWB` relies on the coherent energy produced by the GW signal in the network of detectors. `cWB` has played a major role in the discovery of the first detection of GW signal GW150914 [9] and more recently it has proved itself to be a crucial method to detect interesting transient GW signals that are not well modelled like GW190521 [10,11]. `cWB` is routinely used in a variety of GW transient searches for the LIGO-Virgo-KAGRA collaborations like IMBH searches [12], eBBH searches [13] and generic searches for transients with short [14] and long duration [15].

While `cWB` offers an extensive array of functionalities and scripts, however, it falls short in facilitating user-specific modifications not inherently supported by the framework. Although `cWB` does provide plugin support, these plugins are required to access and manipulate global variables at the specific point of invocation. This approach demands a comprehensive understanding of the underlying code and risks unintentional disruptions or alterations to the variables. Moreover, the lack of clear dependencies between the modules further complicates the task for developers aiming to make modifications as the understanding of the interaction between different components becomes challenging. The `PycWB` framework addresses these issues and offers a more straightforward and stable environment for customization and code alteration.

This paper introduces `PycWB`, a modularized Python package for the `cWB` algorithm. This package will enable the easier integration of the future machine learning algorithm and new Python-based waveform models. The remainder of this paper is structured as follows: Section 2 provides an introduction to the structure and features of `PycWB`. Then, in Section 3, we present several use cases that demonstrate the user-friendliness and efficiency of `PycWB`, comparing its application with the traditional `cWB`. Finally, we share our conclusions and insights on the impact and potential of our new framework in the concluding section.

2. Software description

The software framework in focus is implemented in Python and leverages the coherent Wave Burst (`cWB`) software originally developed on `ROOT` [16]. The description of the core `cWB` algorithm and the code can be found here [17,18]. The native `pyROOT` interface of `ROOT` has immensely facilitated this Python implementation, saving the need for rewriting the entire suite of algorithms used for `cWB`. Instead, the core `cWB` code is integrated, specifically the `WAT` module, which is included in the package and automatically compiled upon installation using `pip`. The installation process is streamlined to avoid the usually intricate `cWB` setup.

To install the `PycWB`, we provide the easiest way which is to use conda due to its dependencies on `ROOT` and `HEALPix`[19] for `cWB` core code

```
conda create -n pycwb "python>=3.9,<3.11"
conda activate pycwb
conda install -c conda-forge root=6.26.10
    healpix_cxx=3.81 nds2-client python-nds2-
    client lalsuite setuptools_scm cmake pkg-
    config
python3 -m pip install pycwb
```

In its design, the software takes a modular approach. This way, the core `cWB` code is divided into different modules, providing a roadmap for future transitions, where the existing C++ codes can be replaced seamlessly with Python modules (see Fig. 1).

2.1. User configurations

The `PycWB` framework makes it simpler for users to set their own parameters. Instead of using C++ scripts like before, `PycWB` uses YAML format, a more user-friendly way that does not need C++ knowledge. Users or developers can easily set their own parameters using a Python dictionary. This dictionary is set up like a JSON schema, allowing flexibility in defining parameters. This system makes sure the user inputs are correct, checks them against the types and ranges that are already set, and provides default values when needed. Additionally it can automatically generate guides or instructions from this schema, keeping users updated with the software's requirements. This new approach makes it simpler for users to work with the software, manage their settings, and helps in faster development.

Listing 1: cWB user parameters

```
strcpy(analysis,"2G");

nIFO = 3;
cfg_search = 'r';
optim=false;

strcpy(ifo[0],"L1");
strcpy(ifo[1],"H1");
strcpy(ifo[2],"V1");
strcpy(refIFO,"L1");

strcpy(channelNamesRaw[0],"L1:GWOSC-4
    KHZ_R1_STRAIN");
strcpy(channelNamesRaw[1],"H1:GWOSC-4
    KHZ_R1_STRAIN");
strcpy(channelNamesRaw[2],"V1:GWOSC-4
    KHZ_R1_STRAIN");

strcpy(frFiles[0],"input/L1_frames.in");
strcpy(frFiles[1],"input/H1_frames.in");
strcpy(frFiles[2],"input/V1_frames.in");
```

Listing 2: PycWB user parameters

```
analysis: "2G"
cfg_search: "r"

optim: False

ifo: ["L1","H1","V1"]
refIFO: "L1"

channelNamesRaw: ['L1:GWOSC-4KHZ_R1_STRAIN', 'H1
    :GWOSC-4KHZ_R1_STRAIN', 'V1:GWOSC-4
    KHZ_R1_STRAIN']
frFiles: ["input/L1_frames.in", "input/H1_frames.
    in", "input/V1_frames.in"]
```

2.2. Modular and classes

The original `cWB` framework presented a challenging structure where class-based constructions excessively encapsulated numerous methods and key algorithms. This approach led to dense coding, which was difficult to comprehend and modify due to its high interdependencies and complexity. However, in the `PycWB` framework, a fundamental shift towards modularity and clarity is adopted. Essential functions are transferred to Python classes, serving as standard data formats or interfaces between different modules. Meanwhile, key algorithms have been detached from their original class environments and restructured into independent modules. This revamped architecture facilitates efficient dependency management. The necessary variables are initialized before

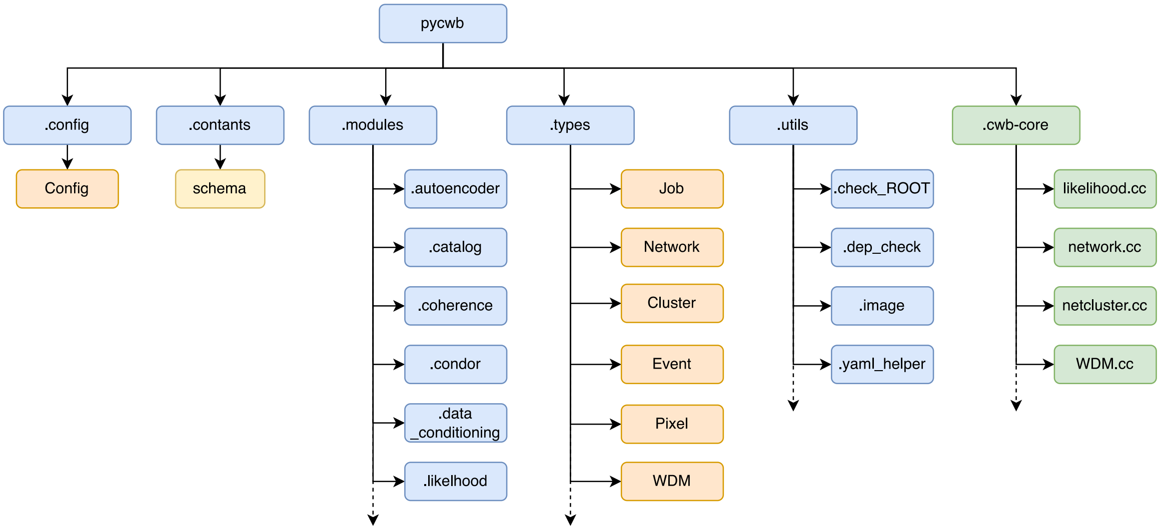
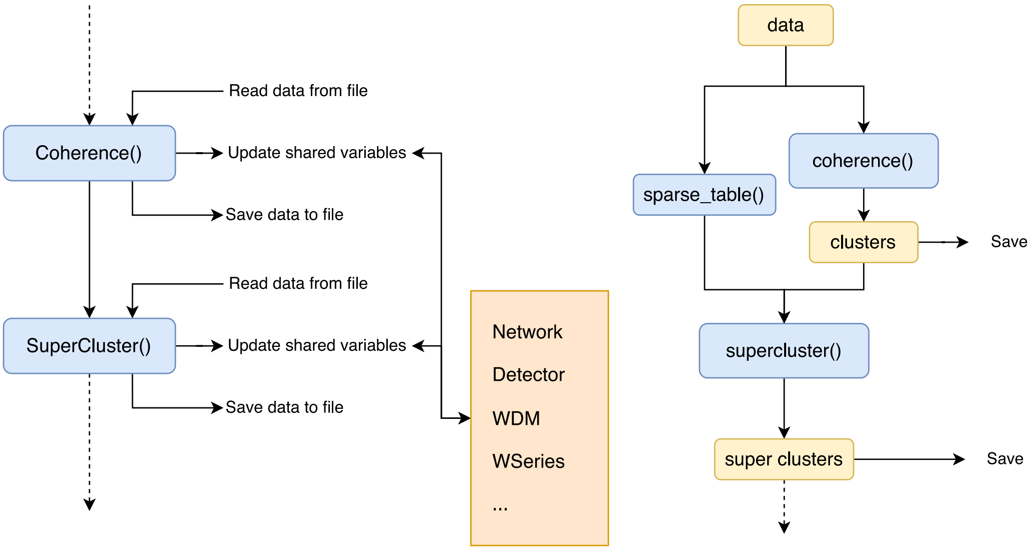


Fig. 1. This figure shows the structure of the `PycWB`. The blue blocks represent the Python modules and the orange blocks represent the Python class. The green blocks show the external C/C++ code embedded in `PycWB`, while the yellow blocks highlight the key variables. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) Part of the workflow for `cWB` (b) Part of the workflow for `PycWB`

Fig. 2. This figure shows the modular design of `PycWB`. On the left, the workflow appears streamlined, but the updates on shared variables are scattered within the modules making it resemble a black box. On the right, we demonstrate our approach to modularization in `PycWB`. We have isolated each module to ensure it only depends on the input and subsequently delivers the output to the next steps. This architecture enhances the transparency and flexibility of the process, allowing for easier comprehension and customization.

each function call, and functions are called as pure procedures, thereby significantly enhancing the software's comprehensibility, usability, and adaptability (see Fig. 2).

2.3. *Parallelization*

The modularization in Python facilitates easy parallelization of various processes in `PycWB`, using Python's `multiprocessing` library. As a result, computations across layers are expedited, bringing about a speedup of 4 to 6 times in the pixel finding and clustering stage. Reading data and data conditioning also enjoy speed improvements, leading to an overall speedup of 2 to 3 times.

A further enhancement of performance by `PycWB` can be envisaged by the integration of GPU acceleration. With the increase in the number of GW events and data the integration of GPU acceleration will become essential for `cWB` algorithm. To this end `PycWB` interface provides a much more straightforward solution with Numba [20].

2.4. *Post production processing*

With Python's wealth of packages for data processing, visualization, and machine learning, post-production data processing is seamless. The modular framework lets users easily select and implement the post-processing modules they need without requiring code modifications or recompilation. For instance, integrating the autoencoder neural network for glitch detection [21] into `PycWB` is as simple as interfacing a few lines of TensorFlow [22] code with the framework. This ensures that advanced techniques such as machine learning can be employed efficiently and effectively without the need for extensive code modifications (see Fig. 3).

2.5. *Web interface*

Similar to `cWB`, `PycWB` also provides a web interface. But the web interface in `PycWB` is structured as a separate module. This module

Fig. 3. A selection of python packages that can be seamlessly integrated for post-production in `PycWB`.

Table 1
Performance comparison: `PycWB` shows 2–3 times of performance improvement due to the parallelization compared to `cWB` on the selected events.

Event name	`cWB` speed factor	`PycWB` speed factor	relative improvement
GW150914	10X	20.1X	2 times
GW170809	4.76X	12.8X	2.7 times
GW190521	8.5X	24.5X	2.9 times

Table 2
The results obtained from the `PycWB` analysis are consistent with those obtained from `cWB`, accommodating only the differences inherent in the different resampling methods and the data types between Python and C++.

Event name	Software	rho[0]	likelihood	start time
GW150914	`cWB`	16.70109	634.7065	1126259462.125
	`PycWB`	16.76488	641.5996	1126259462.125
GW170809	`cWB`	6.045537	117.5504	1186302519.6875
	`PycWB`	6.045362	117.5518	1186302519.6875
GW190521	`cWB`	10.13721	215.8701	1242442967.125
	`PycWB`	10.64826	236.0143	1242442967.125

contains HTML, CSS and Javascript frameworks as a web app, along with simple Python functions to copy the static webpage files to the designated output directory. As a result, there is no need for HTML webpage generation. This separation of the web interface and the Python code contributes to the modularity and usability of `PycWB`.

3. Illustrative examples

3.1. Real data analysis

To validate the performance of `PycWB`, we conducted identical analyses on real LIGO-Virgo events GW150914 [9], GW170809 [23], and GW190521 [10] using both `cWB` and `PycWB`. For the `cWB` analysis, we used the `cwb_gwosc` command to download data from GWOSC [24, 25] and process them.

```
cwb_gwosc GW=EVENT_ID IFO=V1 TAG=TSTXY all
```

For `PycWB`, we utilized the same user configuration file and data, executing the analysis via a simple command.

```
pycwb_search user_parameters.yaml
```

Table 1 outlines the events analysed and the speed factors for both `cWB` and `PycWB`. The speed factor, calculated as the ratio of computation time to the length of the data, indicates a 2–3 times overall performance boost with `PycWB`. This notable increase in speed is primarily attributed to effective parallelization made very simple to implement due to the Python interface.

In addition, we highlight key parameters of the recovered events to showcase the consistency in accuracy between `cWB` and `PycWB`. Given that both platforms use the same algorithms and setups, similar accuracy levels are to be expected. Any minor deviations could potentially be attributed to differences in data types between Python and C++. These results affirm `PycWB`'s capabilities in maintaining the robustness of the algorithm while enhancing performance (see Table 2).

Fig. 4 presents an example of an analysis conducted on GW150914 using `PycWB`. The left panel displays the time–frequency map of the likelihood of selected pixels, providing a visual representation of how the event's signal changes over time and frequency. The right panel illustrates the reconstructed waveform of the event.

3.2. Batch injection with Python script

The `PycWB` framework simplifies the handling of batch injections involving large parameter sets. In `cWB`, injecting complex GW signals (like binary black holes populations) is achieved with cumbersome scripts which create XML files, in particular the LIGO Light-Weight (LIGOLW) XML format [26] for parameters. Moreover, one needs to modify the XML table manually when dealing with keys not predefined in the LIGOLW XML table. On the other hand, `PycWB` provides the option to generate an array of parameters directly through a Python function. This function returns a list of parameters, offering significant flexibility and efficiency. Recently search algorithms like PyCBC etc provide HDF5 injection file support to mitigate the issues with LIGO-LW XML file formats, these injection files can be seemlessly integrated in `PycWB` to have consistent injections between pipelines.

The parameters are passed directly to the waveform generator, thus allowing users to employ their own waveform generators with additional parameters, circumventing the need for any code modification within `PycWB`. The Python function can be constructed as demonstrated in the code snippet 3.

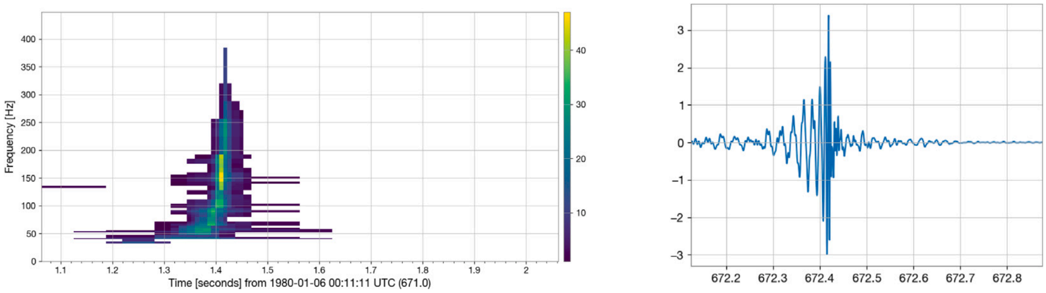
Listing 3: generate_parameters.py: A simple example for batch injection script

```
def get_injection_parameters():
    return [{
        ’mass1’: 20,
        ’mass2’: 20,
        ’spin1z’: spin1z,
        ’spin2z’: 0,
        ’distance’: 200,
        ’inclination’: 0,
        ’polarization’: 0,
        ’gps_time’: 1126259462.4,
        ’coa_phase’: 0,
        ’ra’: 0,
        ’dec’: 0
    } for spin1z in [0, 0.3, 0.6]]
```

To implement this, the `parameters_from_python` keyword can be used in place of the `parameters` keyword in the YAML file containing user parameters. The example shows in code snippet 4.

Listing 4: Example user parameter file for generating injection parameters from script

```
injection:
  segment: # . . . .
  parameters_from_python:
    file: "generate_parameters.py"
    function: "get_injection_parameters"
  approximant: #. . .
  generator: #. . .
```



(a) Likelihood map

(b) Reconstructed waveform

Fig. 4. A selection of the plots from the `PycWB` analysis on GW150914, the first detected GW event. These plots were created using Matplotlib, a popular data visualization library in Python, which takes the output from the `likelihood` module. In comparison with C++, Python's Matplotlib requires significantly less code for plotting, demonstrating the efficiency and ease of use provided by `PycWB` in visualizing analysis results.

3.3. Injection with third-party Python waveform models

While `PycWB` uses PyCBC as the default for waveform generation during injection, users may wish to utilize their own models or adopt new ones that have yet to be implemented in PyCBC. To accommodate this, `PycWB` features an option within its `read_data` module that allows for the integration of any waveform generator. The only requirement is for the user to write a wrapper function that accepts input from their injection parameters and returns the '+' (plus) and '×' (cross) polarization. This flexible approach enables seamless integration of a diverse array of waveform models, thereby expanding the analytical capabilities of the `PycWB` framework.

We demonstrated this ability by running a data injection with the cutting-edge waveform model, `pyseobnr`. This model, only implemented in Python, utilizes the effective-one-body method and numerical relativity (NR) calibration, serving as a testament to the framework's compatibility and flexibility.

Listing 5: Example code using the pySEOBNR waveform model for injection

```
from pyseobnr.generate_waveform import GenerateWaveform

def waveform_generator(mass1, mass2, spin1x, spin1y, spin1z, spin2x, spin2y, spin2z, distance, inclination, polarization, coa_phase, f_lower, delta_t, **kwargs):
    parameters = {
        ’mass1’: mass1,
        ’mass2’: mass2,
        ’spin1x’: spin1x,
        ’spin1y’: spin1y,
        ’spin1z’: spin1z,
        ’spin2x’: spin2x,
        ’spin2y’: spin2y,
        ’spin2z’: spin2z,
        ’distance’: distance,
        ’inclination’: inclination,
        ’polarization’: polarization,
        ’coa_phase’: coa_phase,
        ’f_ref’: f_lower,
        ’f22_start’: f_lower,
        ’deltaT’: delta_t,
        "approximant": "SEOBNRv5HM",
    }
    wfm_gen = GenerateWaveform(parameters)
    hp, hc = wfm_gen.generate_td_polarizations_conditioned_2()

    return hp, hc
```

To integrate a third-party waveform model that is not natively supported in `PycWB`, the user only needs to write a wrapper function, as demonstrated in code snippet 5. Then, to incorporate this into a run, the user simply adds a `generator` key to the user parameter file, specifying the location and function name, as shown in code snippet 6.

Listing 6: Example for using the waveform model wrapper in user parameter file

```
injection:
  segment: # ...
  parameters_from_python: # ...
  approximant: "SEOBNRv5HM"
  generator:
    module: "PATH_TO_THE_CODES"
    function: waveform_generator
```

4. Impact

The Coherent WaveBurst (cWB) is playing a key role in the discovery and analysis of gravitational waves [27–29]. Despite its importance, the accessibility and user-friendliness of its interface have been a challenge, primarily due to the complex and highly technical nature of the C++ language it was originally written in.

In addressing this challenge, we present `PycWB`, a Python-based modular adaptation of the cWB. This transformation not only makes cWB more accessible to the scientific community but also unlocks the potential for numerous innovations in the field. The `PycWB` framework allows seamless integration with machine learning libraries in Python even at the core of the algorithm, paving the way for more sophisticated and automated analyses [21,30,31]. This adaptability extends to developers, who can easily design new modules or add GPU-accelerated capabilities with Python packages like `numba`, without an in-depth understanding of the entire `PycWB` structure. Such potential acceleration will empower us to analyse a greater number of pixels in the time–frequency domain. This enables us to process more data and ensures that potential events are not overlooked. Moreover, the seamless compatibility with PyCBC, third-party Python-based waveform models such as `pySEOBNR` [2] and `gwsurrogate` [3], and extended waveform model with addtional parameter such as `HyperbolicTD` greatly simplifies injection studies [32–34].

By leveraging the power and simplicity of Python, `PycWB` makes complex analyses in gravitational wave research more manageable and user-friendly. The capability to use `PycWB` within interactive Jupyter Notebooks simplifies the learning process for new users, making it significantly more approachable. Additionally, data post-processing is much easier. Its Python-based data output from each module makes data manipulation more intuitive. Users have the flexibility to choose the specific data they wish to save at each stage of the process.

5. Conclusions

In summary, `PycWB` offers a user-friendly, flexible, and potent framework, setting the stage for novel research avenues in gravitational waves. By simplifying the user experience and boosting adaptability, `PycWB` ensures that both seasoned researchers and novices can contribute valuably to the ever-evolving field of gravitational wave research.

CRediT authorship contribution statement

Yumeng Xu: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing. **Shubhanshu Tiwari:** Conceptualization, Project administration, Supervision, Methodology, Writing – original draft, Writing – review & editing. **Marco Drago:** Validation, Software, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

We are grateful to Giovanni Andrea Prodi, Francesco Salemi, Sophie Bini, Dixeena Lopez, Michael Ebersold, Edoardo Milotti and Gabriele Vedovato for their helpful comments and encouragement throughout this project. Y. Xu is supported by China Scholarship Council. S. Tiwari is supported by the Swiss National Science Foundation (SNSF) Ambizione Grant Number: PZ00P2-202204. M. Drago acknowledges the support from the Amaldi Research Center funded by the MIUR program ‘Dipartimento di Eccellenza’, Italy (CUP:B81I18001170001) and the Sapienza School for Advanced Studies (SSAS), Italy. The authors are grateful for computational resources provided by the LIGO Laboratory and supported by National Science Foundation, Switzerland Grants PHY-0757058 and PHY-0823459. This paper has document number LIGO-P2300267.

This research has made use of data or software obtained from the Gravitational Wave Open Science Center (gwosc.org), a service of the LIGO Scientific Collaboration, the Virgo Collaboration, and KAGRA. This material is based upon work supported by NSF ’s LIGO Laboratory which is a major facility fully funded by the National Science Foundation, Switzerland, as well as the Science and Technology Facilities Council (STFC) of the United Kingdom, the Max-Planck-Society (MPS), Germany, 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. KAGRA is supported by Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan Society for the Promotion of Science (JSPS) in Japan; National Research Foundation (NRF) and Ministry of Science and ICT (MSIT) in Korea; Academia Sinica (AS) and National Science and Technology Council (NSTC) in Taiwan.

References

[1] Wette K. SWIGLAL: Python and octave interfaces to the LALSuite gravitational-wave data analysis libraries. SoftwareX 2020;12:100634. http://dx.doi.org/10.1016/j.softx.2020.100634, arXiv:2012.09552.

[2] Mihaylov DP, Ossokine S, Buonanno A, Estelles H, Pompili L, Pürrer M, et al. pySEOBNR: a software package for the next generation of effective-one-body multipolar waveform models. 2023, arXiv:2303.18203.

[3] Field S, Galley C, Blackman J. GWSurrogate: An easy-to-use interface to gravitational wave surrogate models. In: APS April meeting abstracts. APS meeting abstracts, vol. 2018, 2018, p. G14.005.

[4] Ashton G, et al. BILBY: A user-friendly Bayesian inference library for gravitational-wave astronomy. Astrophys J Suppl 2019;241(2):27. http://dx.doi.org/10.3847/1538-4365/ab06fc, arXiv:1811.02042.

[5] Biwer CM, Capano CD, De S, Cabero M, Brown DA, Nitz AH, et al. PyCBC inference: A python-based parameter estimation toolkit for compact binary coalescence signals. Publ Astron Soc Pac 2019;131(996):024503. http://dx.doi.org/10.1088/1538-3873/aaef0b, arXiv:1807.10312.

[6] Renzini AI, et al. pygwb: A python-based library for gravitational-wave background searches. Astrophys J 2023;952(1):25. http://dx.doi.org/10.3847/1538-4357/acd775, arXiv:2303.15696.

[7] Usman SA, et al. The PyCBC search for gravitational waves from compact binary coalescence. Classical Quantum Gravity 2016;33(21):215004. http://dx.doi.org/10.1088/0264-9381/33/21/215004, arXiv:1508.02357.

[8] Klimenko S, et al. Method for detection and reconstruction of gravitational wave transients with networks of advanced detectors. Phys Rev D 2016;93(4):042004. http://dx.doi.org/10.1103/PhysRevD.93.042004, arXiv:1511.05999.

[9] Abbott BP, et al. GW150914: First results from the search for binary black hole coalescence with advanced LIGO. Phys Rev D 2016;93(12):122003. http://dx.doi.org/10.1103/PhysRevD.93.122003, arXiv:1602.03839.

[10] Abbott R, et al. GW190521: A binary black hole merger with a total mass of $150 M_{\odot}$. Phys Rev Lett 2020;125(10):101102. http://dx.doi.org/10.1103/PhysRevLett.125.101102, arXiv:2009.01075.

[11] Szczepańczyk M, et al. Observing an intermediate-mass black hole GW190521 with minimal assumptions. Phys Rev D 2021;103(8):082002. http://dx.doi.org/10.1103/PhysRevD.103.082002, arXiv:2009.11336.

[12] Abbott R, et al. Search for intermediate-mass black hole binaries in the third observing run of advanced LIGO and advanced virgo. Astron Astrophys 2022;659:A84. http://dx.doi.org/10.1051/0004-6361/202141452, arXiv:2105.15120.

[13] Abbott BP, et al. Search for eccentric binary black hole mergers with advanced LIGO and advanced virgo during their first and second observing runs. Astrophys J 2019;883(2):149. http://dx.doi.org/10.3847/1538-4357/ab3c2d, arXiv:1907.09384.

[14] Abbott R, et al. All-sky search for short gravitational-wave bursts in the third advanced LIGO and advanced virgo run. Phys Rev D 2021;104(12):122004. http://dx.doi.org/10.1103/PhysRevD.104.122004, arXiv:2107.03701.

[15] Abbott R, et al. All-sky search for long-duration gravitational-wave bursts in the third advanced LIGO and advanced virgo run. Phys Rev D 2021;104(10):102001. http://dx.doi.org/10.1103/PhysRevD.104.102001, arXiv:2107.13796.

[16] Brun R, Rademakers F. ROOT: An object oriented data analysis framework. In: Werlen M, Perret-Gallix D, editors. Nucl Instrum Methods A 1997;389:81–6. http://dx.doi.org/10.1016/S0168-9002(97)00048-X.

[17] Drago M, Klimenko S, Lazzaro C, Milotti E, Mitselmakher G, Necula V, et al. coherent WaveBurst, a pipeline for unmodeled gravitational-wave data analysis. SoftwareX 2021;14:100678. http://dx.doi.org/10.1016/j.softx.2021.100678, URL https://www.sciencedirect.com/science/article/pii/S2352711021000236.

[18] Klimenko S, Vedovato G, Necula V, Salemi F, Drago M, Poulton R, et al. cWB pipeline library: 6.4.1. Zenodo; 2021, http://dx.doi.org/10.5281/zenodo.5798976.

[19] Górski KM, Hivon E, Banday AJ, Wandelt BD, Hansen FK, Reinecke M, et al. HEALPix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. Astrophys J 2005;622(2):759. http://dx.doi.org/10.1086/427976.

[20] Lam SK, Pitrou A, Seibert S. Numba: a LLVM-based python JIT compiler. 2015, http://dx.doi.org/10.1145/2833157.2833162.

[21] Bini S, Vedovato G, Drago M, Salemi F, Prodi GA. An autoencoder neural network integrated into gravitational-wave burst searches to improve the rejection of noise transients. Classical Quantum Gravity 2023;40(13):135008. http://dx.doi.org/10.1088/1361-6382/acd981, arXiv:2303.05986.

[22] Abadi M, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. 2016, arXiv:1603.04467.

[23] Abbott BP, et al. GWTC-1: A gravitational-wave transient catalog of compact binary mergers observed by LIGO and virgo during the first and second observing runs. Phys Rev X 2019;9(3):031040. http://dx.doi.org/10.1103/PhysRevX.9.031040, arXiv:1811.12907.

[24] Abbott R, et al. Open data from the first and second observing runs of advanced LIGO and advanced virgo. SoftwareX 2021;13:100658. http://dx.doi.org/10.1016/j.softx.2021.100658, arXiv:1912.11716.

[25] Abbott R, et al. Open data from the third observing run of LIGO, Virgo, KAGRA, and GEO. Astrophys J Suppl 2023;267(2):29. http://dx.doi.org/10.3847/1538-4365/acdc9f, arXiv:2302.03676.

[26] The LIGO Scientific Collaboration. LIGO light-weight XML I/O library for python. 2022, URL https://git.ligo.org/kipp.cannon/python-ligo-lw.

[27] Abbott BP, et al. GWTC-1: A gravitational-wave transient catalog of compact binary mergers observed by LIGO and virgo during the first and second observing runs. Phys Rev X 2019;9(3):031040. http://dx.doi.org/10.1103/PhysRevX.9.031040, arXiv:1811.12907.

[28] Abbott R, et al. GWTC-2: Compact binary coalescences observed by LIGO and virgo during the first half of the third observing run. Phys Rev X 2021;11:021053. http://dx.doi.org/10.1103/PhysRevX.11.021053, arXiv:2010.14527.

[29] Abbott R, et al. GWTC-3: Compact binary coalescences observed by LIGO and virgo during the second part of the third observing run. 2021, arXiv:2111.03606.

[30] Cuoco E, et al. Enhancing gravitational-wave science with machine learning. Mach Learn Sci Tech 2021;2(1):011002. http://dx.doi.org/10.1088/2632-2153/abb93a, arXiv:2005.03745.

[31] Lopez D, Gayathri V, Pai A, Heng IS, Messenger C, Gupta SK. Utilizing Gaussian mixture models in all-sky searches for short-duration gravitational wave bursts. Phys Rev D 2022;105(6):063024. http://dx.doi.org/10.1103/PhysRevD.105.063024, arXiv:2112.06608.

[32] Abac AG, et al. Search for eccentric black hole coalescences during the third observing run of LIGO and virgo. 2023, arXiv:2308.03822.

[33] Ramos-Buades A, Tiwari S, Haney M, Husa S. Impact of eccentricity on the gravitational wave searches for binary black holes: High mass case. Phys Rev D 2020;102(4):043005. http://dx.doi.org/10.1103/PhysRevD.102.043005, arXiv:2005.14016.

[34] Ebersold M, Tiwari S. Search for nonlinear memory from subsolar mass compact binary mergers. Phys Rev D 2020;101(10):104041. http://dx.doi.org/10.1103/PhysRevD.101.104041, arXiv:2005.03306.