Testing the robustness of simulation-based gravitational-wave population inference

Damon H. T. Cheung,[1,*] Kaze W. K. Wong,[2] Otto A. Hannuksela,[1] Tjonnie G. F. Li,[1,3,4] and Shirley Ho[2,5,6,7]

[1]*Department of Physics, The Chinese University of Hong Kong, Shatin, Hong Kong*
[2]*Center for Computational Astrophysics, Flatiron Institute, New York, New York 10010, USA*
[3]*Institute for Theoretical Physics, KU Leuven, Celestijnenlaan 200D, B-3001 Leuven, Belgium*
[4]*Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium*
[5]*Department of Astrophysical Sciences, Princeton University, Princeton, New Jersey 08540, USA*
[6]*Physics Department, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, USA*
[7]*Department of Physics, New York University, New York, New York 10012, USA*

(Received 14 January 2022; revised 4 May 2022; accepted 26 September 2022; published 19 October 2022)

Gravitational-wave population studies have become more important in gravitational-wave astronomy because of the rapid growth of the observed catalog. In recent studies, emulators based on different machine learning techniques are used to emulate the outcomes of the population synthesis simulation quickly. In this study, we benchmark the performance of two emulators that learn the truncated power-law phenomenological model by using Gaussian process regression and normalizing flows technique to see which one is a more capable likelihood emulator in the population inference. We benchmark the characteristic of the emulators by comparing their performance in the population inference to the phenomenological model using mock and real observation data. Our results suggest that the normalizing flows emulator can recover the posterior distribution by using the phenomenological model in the population inference with up to 300 mock injections. The normalizing flows emulator also underestimates the uncertainty for some posterior distributions in the population inference on real observation data. On the other hand, the Gaussian process regression emulator has poor performance on the same task and can only be used effectively in low-dimension cases.

DOI: 10.1103/PhysRevD.106.083014

I. INTRODUCTION

Since the first detection of a gravitational-wave (GW) event was announced by the LIGO-Virgo Collaboration in 2016 [1], GW events are being detected routinely at a rapid pace. Construction of second-generation global GW detection networks was done in the past five years including the upgrade of Advanced LIGO/Virgo [2,3] and the Kamioka Gravitational Wave Detector (KAGRA) in Japan [4]. We are getting more observed GW events from these detectors and recently, the LIGO-Virgo-KAGRA Collaboration (LVK) announced 35 candidate events in the second half of the third observational run (O3b) [5]. By adding the observed events in the first, second, and the first half of the third observational run (O1, O2, and O3a) [6,7], the third Gravitational-Wave Transient Catalog (GWTC-3) contains over 60 events across the first three observing runs [8]. Moreover, there is a proposed construction of more powerful detectors with higher sensitivity, such as Einstein Telescope in Europe [9] and Cosmic Explorer in the USA [10]. The rapid growth of observed events in the catalog is forecasted as $\sim 10^6$ GW events to be observed per year in this third-generation detectors network [11]. It opens up a unique window to study the population properties of compact objects [e.g., black holes (BHs) and neutron stars (NSs)] which helps to find the fundamental physics of the Universe. For example, the population studies can improve the existing GW constraints on theory-agnostic modifications to general relativity and explore gravity theories beyond general relativity [12].

Typical analyses assume the underlying population of GW sources is described by some phenomenological model (e.g., [13,14]). For instance, one can assume the mass distribution of a merging binary BH follows a power law. Such a simple assumption of the likelihood can provide a low computational cost and a clear statistical interpretation behind it. As the size of the GW catalog grows rapidly, a more complex phenomenological model is needed to capture a more sophisticated statistical relation. However, phenomenological models do not come from a first principle physics simulation or calculation. Alternatively, one can use population synthesis simulations that are based on some physical parameters such as the BH natal kick velocity [15], the common envelope efficiency of binary evolution [16], and

[*]damoncheung@link.cuhk.edu.hk

the metallicity of the environment [17]. The population synthesis simulations simulate complex physics rather than only describing the structure of the distribution. However, they take a long time to simulate and have high computational costs. In this case, we can use a nonparametric density estimator to emulate the outcomes of the population synthesis simulation with fast speed. Then, we can perform the population inference efficiently by using the estimator to provide direct physical insights into GW population studies.

There are recent developments on emulating simulations through machine learning to build the population probability density emulator for GW population studies [18–20]. The emulator interpolates simulation output without going through sophisticated simulations that are fast enough to be used in hierarchical Bayesian analysis (HBA) for population inference. One of these techniques is to combine Gaussian process regression (GPR), principal component analysis (PCA), and space-filling algorithms to train the emulator [18]. Another example is applying a deep generative flow technique; normalizing flows (NF) [20] to train the emulator. However, we lack benchmark results to show how well they perform and what limitations they have when utilized in the GW population inference.

In this study, we investigate the performance of the emulators trained by using these two techniques, which we refer to as the GPR emulator and the NF emulator. We train the emulators to learn the truncated power-law phenomenological model [21] and demonstrate their ability by comparing the performance in event sampling and population inference to the phenomenological model. More specifically, we use the emulators to sample GW events to see if the distributions match the phenomenological model. We also implement the emulators in the HBA framework to act as the population probability density emulator and compare the sampled posterior distribution to the phenomenological model by injecting mock and GWTC-2 data. Note that the selection bias caused by the limitation of instruments [22] will bias the sampled posterior distribution in the population inference [19,23]. Therefore, we account for the selection bias in the population inference when benchmarking the emulators.

This paper is structured as follows. In Sec. II, we review the GW population data analysis pipeline. In Sec. III, we describe the details of the two machine learning emulators we used in this paper. In Sec. IV, we present the emulators' performance by using both mock data and GWTC-2 data. And lastly, in Sec. V, we discuss the limitation of the emulators and the future directions of this work.

II. METHOD

A. Hierarchical Bayesian analysis

In this section we summarize the pipeline of GW population data analysis. We start with the data analysis on a single GW event. Then we show how we make use of a set of observed GW events to study the population properties using the HBA. First, GW data $\boldsymbol{d}$ announced by LVK is usually in the form of a time series which contains no physical quantities directly. The data can be modeled using a waveform model characterized by source properties $\boldsymbol{\theta}$ (e.g., masses, masses ratio, redshift, spin) known as the event parameters. In order to extract physical quantities from a time series, we often use Bayesian inference to adopt a parameter estimation process [24]. Given a time series $\boldsymbol{d}$, the event posterior $p(\boldsymbol{\theta}|\boldsymbol{d})$ can be obtained using Bayes' theorem,

$$p(\boldsymbol{\theta}|\boldsymbol{d}) = \pi(\boldsymbol{\theta}) \frac{p(\boldsymbol{d}|\boldsymbol{\theta})}{p(\boldsymbol{d})}, \tag{1}$$

where $\pi(\boldsymbol{\theta})$ is the prior of event parameters, $p(\boldsymbol{d}|\boldsymbol{\theta})$ is the event likelihood of observing the data given the source properties with a specific waveform model, and $p(\boldsymbol{d})$ is the evidence. $\pi(\boldsymbol{\theta})$ carries the physical intuition (e.g., mass cannot be negative and masses ratio cannot be greater than 1) which can also affect the estimation result [25,26].

To study the population, we can employ a phenomenological population model or simulation-based model which is characterized by the hyperparameters $\boldsymbol{\lambda}$ and then infer the hyperparameters favored by the observed catalog [27]. For example, if we take the route of employing a phenomenological model, the distribution of mass can follow a power law with spectral index α, i.e., $p(\boldsymbol{\theta}|\boldsymbol{\lambda}) = p(m|\alpha) \sim m^{\alpha}$ with α being the hyperparameter. On the other hand, we can employ a simulation-based model that can provide a synthetic catalog of GW events instead of an analytical expression of the population probability density function. In this case, we can train an emulator on this model using machine learning techniques to emulate the $p(\boldsymbol{\theta}|\boldsymbol{\lambda})$, where the hyperparameters could be some physical parameters such as the metallicity of the environment [17]. In the following, we summarize a statistical framework of inferring the hyperparameters given a set of observations in the content of the GW population. The framework is commonly labeled as hierarchical modeling [27]. We refer interested readers to more detailed explanations in the literatures [27–29]. Similar to the parameter estimation of a single GW event, we now want to infer the hyperparameters of the population model given some time series data. Therefore, we start by writing down Bayes' theorem in terms of $\boldsymbol{d}$, and $\boldsymbol{\lambda}$ as

$$p(\boldsymbol{\lambda}|\boldsymbol{d}) = \pi(\boldsymbol{\lambda}) \frac{p(\boldsymbol{d}|\boldsymbol{\lambda})}{p(\boldsymbol{d})}, \tag{2}$$

where $p(\boldsymbol{\lambda}|\boldsymbol{d})$ is the population posterior, $p(\boldsymbol{d}|\boldsymbol{\lambda})$ is the likelihood of observing the data set given the population model characterized by the hyperparameters $\boldsymbol{\lambda}$, $\pi(\boldsymbol{\lambda})$ is the prior of hyperparameters, and $p(\boldsymbol{d})$ is the evidence. However, population synthesis simulations give the population in terms of event parameters instead of time series. Therefore, we need to expand the marginalized likelihood $p(\boldsymbol{d}|\boldsymbol{\lambda})$ as $p(\boldsymbol{d}|\boldsymbol{\lambda}) = \int p(\boldsymbol{d}|\boldsymbol{\theta}) p_{\mathrm{pop}}(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta}$ and replace $p(\boldsymbol{d}|\boldsymbol{\theta})$ by using Eq. (1) to get

$$p(\boldsymbol{\lambda}|\boldsymbol{d}) = \pi(\boldsymbol{\lambda}) \int \frac{p(\boldsymbol{\theta}|\boldsymbol{d}) p_{\mathrm{pop}}(\boldsymbol{\theta}|\boldsymbol{\lambda})}{\pi(\boldsymbol{\theta})} d\boldsymbol{\theta}, \tag{3}$$

where $p_{\mathrm{pop}}(\boldsymbol{\theta}|\boldsymbol{\lambda})$ is the population probability density of observing the event given the population model characterized by the hyperparameters.

Furthermore, if the dataset contains multiple observed GW events that are drawn independently from the population, in other words, the signals are not overlapping and the parameter estimation is not correlated for different events, we can separate the likelihood of observing that particular set of events [i.e., $p(\boldsymbol{d}|\boldsymbol{\lambda})$ in Eq. (2)], the integral in Eq. [(3)]) into the product of the individual likelihoods. Therefore, we can rewrite Eq. (3) as

$$p(\boldsymbol{\lambda}|\boldsymbol{d}) = \pi(\boldsymbol{\lambda}) \prod_{i=1}^{N_{\mathrm{obs}}} \int \frac{p_i(\boldsymbol{\theta}_i|\boldsymbol{d}_i)}{\pi_i(\boldsymbol{\theta}_i)} p_{\mathrm{pop}}(\boldsymbol{\theta}_i|\boldsymbol{\lambda}) d\boldsymbol{\theta}, \tag{4}$$

where $\boldsymbol{d}_i$ refers to the segment of the whole time series which contains the ith event characterized by $\boldsymbol{\theta}_i$ and N_{obs} is the number of observed events. By separating the likelihood into a product of individual likelihoods, we assume the event parameter estimation is not correlated. This is a valid assumption for the current generation detector. However, we need to revise the assumption for detectors in the next generation, such as the Einstein Telescope and Cosmic Explorer. They can detect a large number of GW signals and may eventually overlap, and the interference of the overlapped waveform may affect the parameter estimation and thus the population inference [30].

In real life data, there is often a selection bias that comes from the limitations of ground-based interferometers. The detectors can only detect signals from a specific frequency range above a signal-to-noise ratio threshold [31]. It limits the ability to detect weak signals and misses many low mass GW events. In addition, the sensitivity of the detectors depends on the sky location [22] and the luminosity distance correlates with the redshift of the event [32,33]. As a result, some events are easier to observe than others, which introduces a bias on the observed population. Whether we can detect the event is not a binary yes or no because the detectors are noisy. The best we can do is to calculate $p_{\mathrm{det}}(\boldsymbol{\theta})$ i.e., the probability of detecting an event with event parameters $\boldsymbol{\theta}$ [23]. When we account the selection bias, Eq. (4) becomes

$$p(\boldsymbol{\lambda}|\boldsymbol{d}) = \pi(\boldsymbol{\lambda}) \prod_{i=1}^{N_{\mathrm{obs}}} \int \frac{p_i(\boldsymbol{\theta}_i|\boldsymbol{d}_i)}{\pi_i(\boldsymbol{\theta}_i)} \frac{p_{\mathrm{pop}}(\boldsymbol{\theta}_i|\boldsymbol{\lambda})}{\alpha(\boldsymbol{\lambda})} d\boldsymbol{\theta}, \tag{5}$$

where $\alpha(\boldsymbol{\lambda}) = \int p_{\mathrm{pop}}(\boldsymbol{\theta}'|\boldsymbol{\lambda}) p_{\mathrm{det}}(\boldsymbol{\theta}') d\boldsymbol{\theta}'$ is the selection bias term. Notice that the computation of multidimensional $\boldsymbol{\theta}$ integrals are very expensive. Therefore, the event posterior $p_i(\boldsymbol{\theta}_i|\boldsymbol{d}_i)$ is often given in a form of discrete samples from event parameter estimation [34,35]. We can then separate the event parameter estimation from sampling the population posterior. First, we perform event parameter estimation and save the event posterior samples. Then, we compute $p(\boldsymbol{\lambda}|\boldsymbol{d})$ using the event posterior samples to avoid unnecessary recomputation of event parameter estimation which significantly reduces the computation load for each run. The above process is equivalent to computing the integral in Eq. (5) as the expectation value of the population probability density that has been reweighted by the prior. That is, replace the integral with the sum of the discrete-event posterior samples as

$$p(\boldsymbol{\lambda}|\boldsymbol{d}) = \pi(\boldsymbol{\lambda}) \prod_{i=1}^{N_{\mathrm{obs}}} \frac{1}{S_i} \sum_{j=1}^{S_i} \frac{p_{\mathrm{pop}}(\boldsymbol{\theta}_i^j|\boldsymbol{\lambda})}{\pi(\boldsymbol{\theta}_i^j)\alpha(\boldsymbol{\lambda})}, \tag{6}$$

where j labels the jth posterior sample of the ith event and S_i is the number of discrete posterior samples for the ith event. Notice that we do not include event rate in our derivation so Eq. (6) is the governing equation for the HBA framework. Once we get all the ingredients, we can sample the posterior using various methods such as nested sampling and Markov Chain Monte Carlo (MCMC). In this study, we use MCMC to sample the posterior. The samples represent $p(\boldsymbol{\lambda}|\boldsymbol{d})$ which tells us the inferred hyperparameters of the simulation that favored by the observation data. We summarize the pipeline in a schematic diagram shown in Fig. 1.

B. Computation of selection bias

In order to compute $\alpha(\boldsymbol{\lambda})$ in Eq. (6), one needs to inject a large amount of signals and recover them with a search pipeline to estimate $\alpha(\boldsymbol{\lambda})$; this incurs an expensive computational cost not to mention $\alpha(\boldsymbol{\lambda})$ will be computed for each step in the MCMC. Therefore, we approximate it via Monte Carlo with importance sampling [36]. By drawing events from a known distribution $\boldsymbol{\theta} \sim p_{\mathrm{draw}}$, we can then get the selection bias term by averaging the population probability of detectable events over the drawn samples as

$$\alpha(\boldsymbol{\lambda}) \approx \frac{1}{N_{\mathrm{draw}}} \sum_{j=1}^{N_{\mathrm{det}}} \frac{p_{\mathrm{pop}}(\boldsymbol{\theta}|\boldsymbol{\lambda})}{p_{\mathrm{draw}}(\boldsymbol{\theta})}, \tag{7}$$

where j labels the jth detectable sample of drawn events, N_{draw} is the number of event samples to be drawn from p_{draw}, and N_{det} is the number of detected events from drawn event samples.

When we inject the O1 + O2 + O3a catalog into the GW population data analysis pipeline, $\alpha(\boldsymbol{\lambda})$ is evaluated by

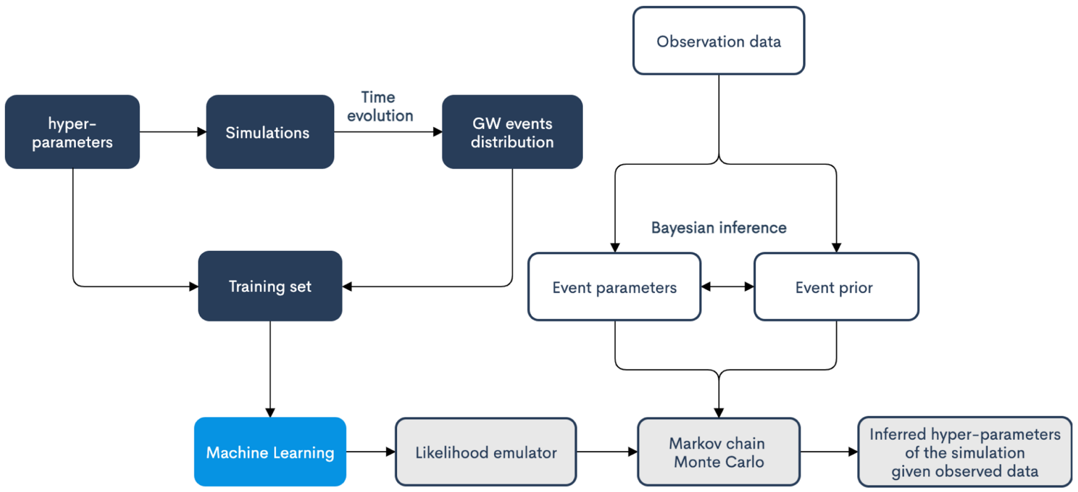


FIG. 1. Schematic diagram of GW event population data analysis pipeline. Training data is generated by running a set of simulations with different hyperparameters inputs. Then we use the training data to train the likelihood emulator with a machine learning technique. Event parameters and priors are obtained by performing Bayesian inference on observation data. Lastly, given the event parameters, prior and the population probability density emulator, we sample the posterior of hyperparameters using the MCMC method.

reweighting an injection campaign done by the LVC [21]. When we test our pipeline by injecting mock data, $\alpha(\boldsymbol{\lambda})$ is evaluated by using the $p_{\mathrm{det}}(\boldsymbol{\theta})$ function in the developed interpolation package GWDET [37] for simplicity.

III. EMULATOR

In computing the population posterior $p(\boldsymbol{\lambda}|\boldsymbol{d})$ in Eq. (6), the population probability density function $p_{\mathrm{pop}}(\boldsymbol{\theta}|\boldsymbol{\lambda})$ is the most important part since it relates the event distribution and the hyperparameters of the simulation. A common approach is writing down an analytic or semianalytic population probability density function. However, in the case of population synthesis simulation, it is not so simple. As a result, we need to simulate each sampling step to calculate $p(\boldsymbol{\lambda}|\boldsymbol{d})$ in Eq. (6). In addition, a typical population synthesis simulation takes $\sim$3–4 hours to complete. If we simulate each step, the population pipeline becomes computationally expensive [38–41]. Hence, we train a emulator by using machine learning techniques to learn the likelihood function $p_{\mathrm{pop}}(\boldsymbol{\theta}|\boldsymbol{\lambda})$ from population synthesis simulations. A trained emulator can approximate the output of simulation by giving the hyperparameters $\boldsymbol{\lambda}$ without going through the sophisticated simulations. To benchmark the capability of the emulators, we test whether they can recover the phenomenological model likelihood and compare their performance in the HBA framework. The inferred posterior by using the phenomenological model is treated as a control set for the comparison. In this study, truncated power law phenomenological model [21] (see Appendix) is chosen for the comparison with the GPR emulator and the NF emulator respectively. The truncated power law phenomenological model's event parameters are m_1, m_2, and z which correspond to the primary mass, secondary mass, and redshift of the GW event, respectively, while the hyperparameters $\boldsymbol{\lambda} = [\alpha, \beta, m_{\mathrm{min}}, m_{\mathrm{max}}]$. In the following three subsections, we will present the method used for generating training data. Then, we review how we train our GPR emulator and NF emulator.

A. Training data

In this study, we use 400 simulations as our training set. We choose the hyperparameters $\boldsymbol{\lambda}_{\mathrm{train}} = \{\boldsymbol{\lambda}_i\}$, $i = 1, 2, \ldots, 400$ of the simulations by Latin-hypercube sampling (LHS). LHS gives the advantage that stratifies each univariate margin simultaneously [42] with variance reduction form compared with uniform random sampling [43]. It does not contain a duplicate number in each hyperparameter dimension. Therefore, LHC gives more uniform coverage in the hyperparameter space than Cartesian grid sampling to help the training. We use python package pyDOE [44] to carry out the LHS. Then we draw 10^5 events from each simulation by rejection sampling as the training set for both emulators. 100 more simulations are generated as a validation set for training the NF emulator.

B. Gaussian process regression emulator

Gaussian process regression is a nonparametric density estimation method. Instead of parametrizing the underlying density function, it places a Gaussian prior characterized by a mean and covariance to describe the possible density function. Then we can fit the density function by inferring the mean and covariance with the training data [45]. We follow the method in [18] tightly to construct the GPR emulator. First, we produce histograms with equal-sized bins over event parameters to summarize the event distribution for simulations characterized by different hyperparameters. Then, we form a matrix $A_{m\times n}$ by using the information of histograms, where m is the number of simulations in the training set and n is the number of flatten bins over all event parameters in the histograms. The probability of having the event (represented by the bin) is proportional to the height of each histogram bin; therefore, we can apply GPR to learn how the input hyperparameters affect the height. However, some components of the basis obtained by such naive binning might be unnecessary if the training data can be described only by some main features. It will increase the computational cost exponentially. Therefore, before applying GPR, we use PCA to form a new set of data-driven basis which is smaller in number than the basis obtained in the naive binning method. PCA decomposes the data matrix $A_{m\times n}$ as

$$A_{m\times n} = U_{m\times m} S_{m\times n} W^T_{n\times n} \tag{8}$$

where U and W are constituted by orthonormal eigenvectors chosen from AA^T and A^TA respectively, A^T is the transpose of A. $S_{m\times n}$ is a positive-semidefinite matrix which can be interpreted as a rectangular diagonal matrix with the variance σ_m of each basis. With this form, we can then eliminate the basis corresponding to $\sigma_m < \epsilon$ to reduce the dimensions of U, S, W as

$$\begin{aligned} A_{m\times n} &\overset{\text{PCA}}{\Rightarrow} (A_{m\times n})_{\sigma_m>\epsilon} \\ &= \tilde{A}_{m'\times n'} \\ &= \tilde{U}_{m'\times m'}\tilde{S}_{m'\times n'}\tilde{W}^T_{n'\times n'} \\ &= \left(\frac{\tilde{U}_{m'\times m'}\tilde{S}_{m'\times n'}}{\sqrt{N_{\text{basis}}}}\right)(\sqrt{N_{\text{basis}}}\tilde{W}^T_{n'\times n'}), \end{aligned} \tag{9}$$

where ϵ is a small number, N_{basis} is the number of basis after reducing dimensions and $\tilde{U}$, $\tilde{S}$, $\tilde{W}$ are formed by restricting U, S, W on a basis with $\sigma_m < \epsilon$ condition. The columns of $\tilde{U}_{m'\times m'}\tilde{S}_{m'\times n'}/\sqrt{N_{\text{basis}}}$ are the principal components (PCs) of the data matrix, while $\sqrt{N_{\text{basis}}}\tilde{W}^T_{n'\times n'}$ is the projection of the original histogram heights into the new basis. This helps to reduce data complexity without losing too much information as the basis after PCA describes the main features of the whole training data set. Then, we use SCIKIT-LEARN [46] to apply GPR on each basis with correct PC weighting by inferring training data with a Gaussian prior. The trained emulator gives the resulting posterior-predictive distribution with Gaussian noise that comes from the credible region. We can obtain a point prediction of $p_{\text{pop}}(\boldsymbol{\theta}|\boldsymbol{\lambda})$ using the mean of the posterior distribution.

In this study, we are not able to obtain a satisfactory GPR emulator by using 400 simulations, each containing 10^5 or even 10^6 events as the training data. Therefore, we construct the matrix $A_{m\times n}$ using the theoretical probability density of the phenomenological model as the height for each histogram bin. After reducing the complexity using PCA, we keep 114 PCs to train the emulator.

C. Normalizing flows emulator

Another approach we use to emulate conditional probabilities is using conditional neural density estimators, in particular, a flow-based generative (often referred to as normalizing flows) model [47]. Unlike other neural density estimators using variational autoencoders [48] or generative adversarial networks [49] that can only generate new data that mimics the target distribution (in our case, the GW event distribution of the simulation), a flow-based generative model can also provide an estimate of the probability density which can be evaluated fast enough in HBA. In this section we present the basic principles behind the model we use.

The idea of NF is to transform a simple probability density (e.g., a Gaussian) $z \sim p_z$ into a target probability density which is much more complicated $x \sim p_x$ by an invertible transformation with tractable Jacobian. The transformation is a mapping function $\boldsymbol{g}\colon \mathbb{R}^d \to \mathbb{R}^d$ for $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^d$. We can then get the change of variable relation from the normalization condition of probabilities as

$$\begin{aligned} p_x(\boldsymbol{x}) &= p_z(z)\left|\det\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right| \\ &= p_z(\boldsymbol{g}^{-1}(x))\left|\det\frac{\partial \boldsymbol{g}^{-1}(x)}{\partial \boldsymbol{x}}\right|, \end{aligned} \tag{10}$$

which requires the transformation to be invertible and thus gives a tractable Jacobian to evaluate $p_x(\boldsymbol{x})$. For estimating more complex high-dimensional distribution, we need more complex transformations, which can be done by applying a series of invertible transformations as

$$\boldsymbol{x} = \boldsymbol{z}_k = \boldsymbol{g}_k \circ \boldsymbol{g}_{k-1} \circ \cdots \circ \boldsymbol{g}_1(z_0), \tag{11}$$

where $\boldsymbol{z}_k$ is the distribution after the kth transformation function. The condition on invertibility is still fulfilled since the composition of invertible functions is invertible. At the same time, we need to be aware of the time for training since the computation of high-dimensional Jacobian determinants is expensive. In conclusion, the transformations $\boldsymbol{g}$ should be invertible and simple.

Then, we can write down the overall transformation as

$$\begin{aligned} p_x(\boldsymbol{x}) &= p_{z_0}(\boldsymbol{z}_0)\prod_{k=1}^{N_{\text{transf}}}\left|\det\frac{\partial \boldsymbol{z}_{k-1}}{\partial \boldsymbol{z}_k}\right| \\ &= p_{z_0}(\boldsymbol{z}_0)\prod_{k=1}^{N_{\text{transf}}}\left|\det\frac{\partial \boldsymbol{g}_k^{-1}(z_k)}{\partial \boldsymbol{z}_k}\right|, \end{aligned} \tag{12}$$

where N_{transf} is the number of transformations.

However, choosing the correct transformation is crucial to designing an efficient network for a specific problem. Our target is to emulate $p_{\text{pop}}(\boldsymbol{\theta}|\boldsymbol{\lambda})$ so the network should be capable to model conditional probabilities. A specifically designed flow-based generative model known as masked autoregressive flow (MAF) [47] is capable of such a purpose. It is a particular implementation of the NF that uses the masked autoencoder for distribution estimation (MADE) [50] as a building block instead of the fully-connected layer. MADE masks some autoencoder's parameters of hidden layers to respect autoregressive constraints that each node is only from previous inputs in a given ordering so that the node only depends on some nodes from the previous layer. It expands the joint probability into the products of the conditional probabilities' relation with a different order [51]. Therefore, we use MAF with a 10 layer network as the "flow" in the emulator.

We can then train the emulator with a loss function defined as

$$\mathcal{L} = -\frac{1}{|\mathcal{D}|}\sum_{x\in\mathcal{D}} \log(p_x(\boldsymbol{x})), \tag{13}$$

where $\mathcal{L}$ is the loss function, $\mathcal{D}$ is the dataset and $p_x(\boldsymbol{x})$ is the entire transformation. $\mathcal{L}$ gives the indicator whether the emulator provides a similar distribution compared to target distribution $\boldsymbol{x}$. Then, the emulator with the best transformation is obtained by finding the global minimum of $\mathcal{L}$ [52].

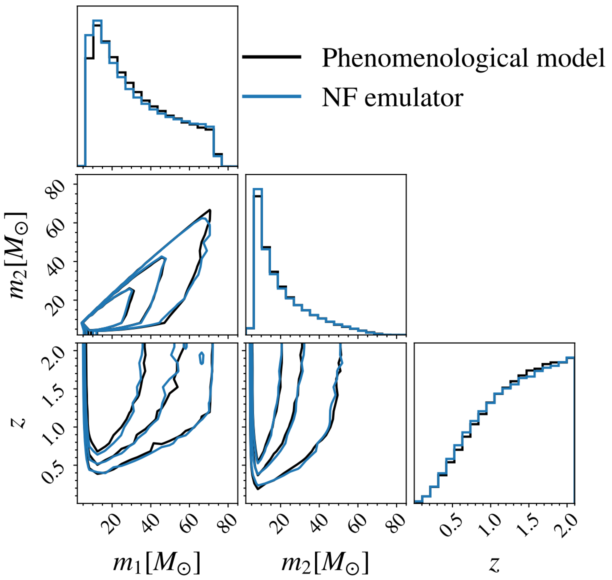
IV. RESULT

A. Comparison on event distribution

We compare the performance of emulators on sampling GW events to the phenomenological model. In Fig. 2, we show the event distributions predicted by two emulators with test hyperparameter $[\alpha, \beta, m_{\min}, m_{\max}] = [3.0, 0.5, 6.0, 74.0]$. The event distributions predicted by the phenomenological model represent the true event distributions. The distributions predicted by the GPR emulator have significant discrepancies with the true event distributions. In marginalized m_1 distribution, the one predicted by the GPR emulator does not agree with the true event distributions where the largest discrepancies appear at the mass limits. Furthermore, the joint distributions of m_1 have an irregular shape when compared to the true event distributions. Although the marginalized distribution of m_2 and z have low discrepancies, the joint distribution is still significantly different from the true event distributions. On the other hand, both the marginalized and joint distributions predicted by the NF emulator match the true distributions. It can also recover the truncation characteristic at the limit of the event parameters. The distribution similarity between using the NF/GPR emulator and the phenomenological model, as quantified by the Kullback-Leibler divergence [53], is $D_{\mathrm{KL}} = 0.141,\ 0.277$ nat respectively. A smaller D_{KL} indicates that two distributions are more similar, implying that the NF emulator's event distributions are more similar to the true distributions than the GPR emulator.

B. Inference on mock data

Next, we examine the performance of the emulators on a population level. To begin with, we build a mock catalog



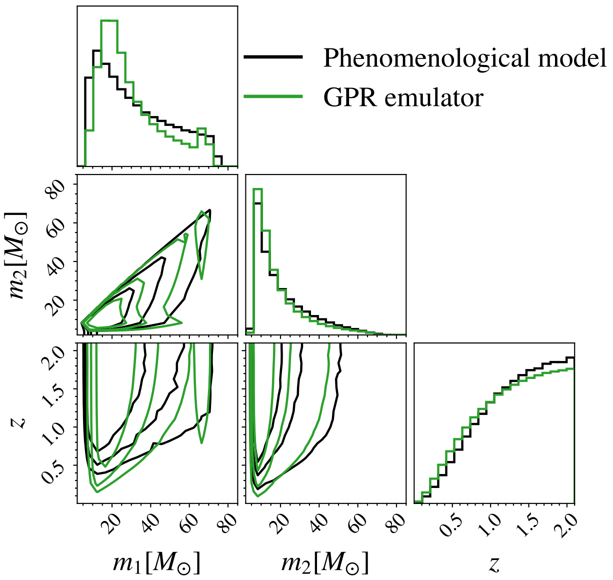


FIG. 2. Test of the NF emulator and the GPR emulator. The test hyperparameter $[\alpha, \beta, m_{\min}, m_{\max}] = [3.0, 0.5, 6.0, 74.0]$ is not included in the training set and validation set. The black curves show the events sampled by using the phenomenological model (true distributions), the blue curves (left panel) show those by using the NF emulator, and the green curves (right panel) show those by using the GPR emulator. The diagonal plots are marginalized distributions for each event parameter while the off-diagonal plots are the joint distributions between the event parameters. The three contour levels represent the 50%, 70%, and 90% credible regions of the distributions.

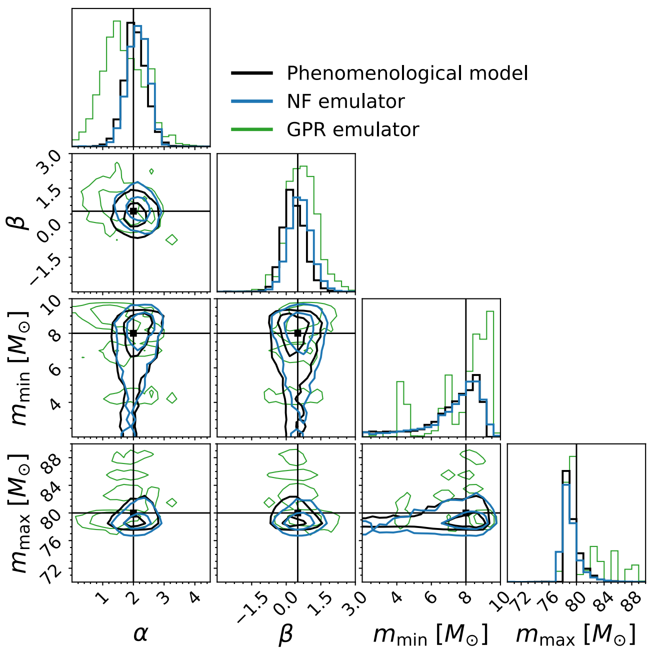


FIG. 3. The sampled posterior distribution after injecting 50 events from a mock catalog with selection bias. The hyperparameters that characterize the mock catalog $[\alpha, \beta, m_{\min}, m_{\max}] = [2.0, 0.5, 8.0, 80.0]$ (true answers) are marked by the black lines. The black curves show the sampled posterior distributions by using the phenomenological model (true posterior distributions), the blue curves show those by using the NF emulator, and the green curves represent those by using the GPR emulator. The two contour levels represent the 50% and 90% credible regions of the distributions. The posterior distribution obtained by using the GPR emulator can barely recover the true posterior distribution. Even after training the GPR emulator with more data, the distributions remain to scatter and diverge. In contrast, the NF emulator recover the true posterior distribution.

with the truncated power-law model by using rejection sampling. The hyperparameters that characterize it (true hyperparameters) are $[\alpha, \beta, m_{\min}, m_{\max}] = [2.0, 0.5, 8.0, 80.0]$. Then, we evaluate the performance of two emulators by injecting 50 GW events from the mock catalog with selection bias. The sampled posterior distributions by using the phenomenological model represent the true posterior distributions.

In Fig. 3, we show the joint and marginalized posterior distributions of the hyper-parameter that favor the injected 50 GW events. The sampled posterior distributions by using the GPR emulator diverge and scatter with only 50 GW events. They have multiple local minimums which are different from the true posterior distributions. The result suggests that the GPR emulator is not capable to act as a likelihood emulator even at a low-injection regime. On the other hand, using the NF emulator can recover the true posterior distribution with low discrepancy. They agree with each other in both marginalized distributions, joint distributions, and the most probable values.

As the number of observed GW events is rapidly increasing, the bias of inferred hyperparameters in simulation-based inference will become significant. Therefore, we evaluate the performance of the NF emulator on more injections and compare with the phenomenological model. We draw 50, 100, 150, 300 GW events from the mock catalog and infer the hyperparameters. Figure 4 shows the violin plots of marginalized posterior distribution of inferred hyperparameters by using the phenomenological model and the NF emulator. Although the NF emulator's posterior distribution does not perfectly match that of the phenomenological model, the inferred hyperparameters agree with the true answers with 90% credible regions up to 300 injections. The marginalized posterior distribution of inferred hyperparameters by using the phenomenological model converges toward the true answer as the number of injections increases. On the other hand, the NF emulator gives similar convergence behavior. The result shows the NF emulator is still a capable likelihood estimator when $N_{\text{inj}} = 300$ and selection bias is included. However, notice that we use a smooth model (NF) to interpolate a hard cutoff phenomenological model. Therefore, the poor $m_{\min}/m_{\max}$ convergence performance is inevitable. Figure 5 shows the variance of inferred $m_{\min}/m_{\max}$ posterior distribution against the number of injections. When compared to the phenomenological model, the $m_{\max}$ uncertainty shrinks slower when using the NF emulator, but there is no big difference for the $m_{\min}$ uncertainty. Because we have more observed events near $m_{\max}$ than $m_{\min}$, the limitation of the NF emulator is more clearly shown in the inferred $m_{\max}$ posterior distribution.

C. Data from GWTC-2

We also evaluate the performance of the emulator on real data. For simplicity, we only use 44 high-significance events from GWTC-2 [7] because our goal is to compare the emulators' performance to the phenomenological model. The dataset is the same subset chosen for the population analyses in [21]. In particular, we exclude three events with a large false-alarm rate (GW190426, GW190719, GW190909) and three events with $m_2 < 3\ M_\odot$ (GW170817, GW190425, GW190814).

We performed two population inferences on GWTC-2 data by using the GPR emulator as shown in Fig. 6. One only trained the GPR emulator on three hyperparameter $[\alpha, \beta, m_{\min}]$ (left panel), another one trained on all hyperparameters (right panel). Since those hyperparameters are independent of each other, the result of inferring three or four hyperparameters will be the same. For inferring three hyperparameters, the GPR emulator can recover the posterior distribution by using the phenomenological model with fair convergence. In the four hyperparameters cases, they have irregular and diverge distributions, which do not agree with the sampled distributions obtained by using the phenomenological model.

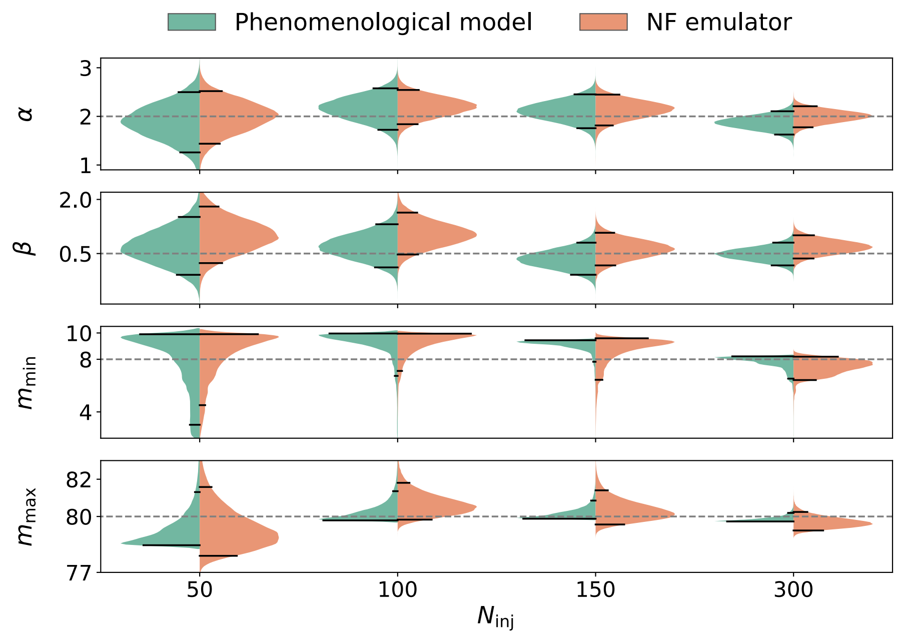


FIG. 4. Marginalized posterior distribution on inferred hyperparameters by using the phenomenological model (green) and the NF emulator (orange). Inferences on different numbers of injections $N_{\rm inj} = 50, 100, 150, 300$ are performed. Horizontal black ticks and dashed grey lines mark the 90% credible regions and true answers. The inferred hyper-parameters by using the NF emulator agree with the true answers with 90% credible region up to 300 injections.

The performance of the population inference on GWTC-2 data by using the NF emulator is shown in Fig. 7. The NF emulator is capable of recovering the sampled distribution by using the phenomenological model except for the $m_{\rm min}$ and $m_{\rm max}$ related distributions. Although their most probable values are aligned, the uncertainties predicted by the NF emulator are smaller for $m_{\rm min}$ and $m_{\rm max}$. The result shows the NF emulator can not learn the truncation characteristic perfectly.

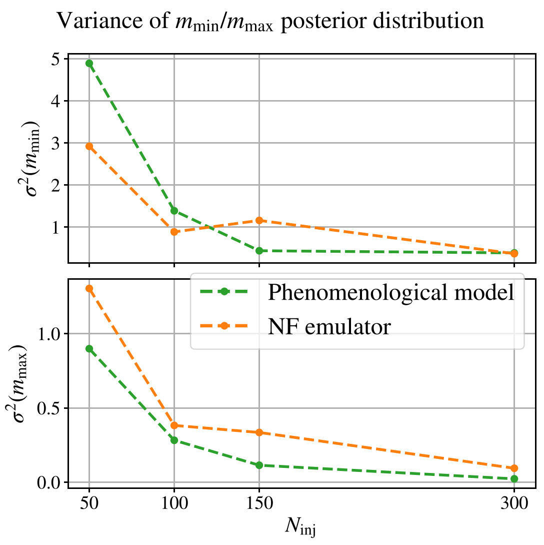


FIG. 5. Variance of inferred $m_{\rm min}$, $m_{\rm max}$ posterior distribution in Fig. 4. The upper/bottom panel shows the variance of inferred $m_{\rm min}/m_{\rm max}$ posterior distribution. Green lines represent the sampled posterior distributions by using the phenomenological model while orange lines represent those by using the NF emulator. The $m_{\rm max}$ uncertainty shrinks slower when using the NF emulator.

V. DISCUSSION

We showed the performance of the GPR and NF emulators by employing a truncated power-law model. For the GPR emulator, it is hard to sample the correct event distribution near the truncation as shown in Fig. 2. In addition, the sampled posterior distribution of $m_{\rm min}$, $m_{\rm max}$ scatter more strongly than α and β in Fig. 3. The results reveal the inability of the GPR emulator to learn the truncation property. For a truncated power-law distribution, we have relatively fewer data near the truncation. However, we need relatively more data to learn the sharp edge. As a result, we have insufficient training data to learn the truncation property and thus have poor performance on the population inference. Moreover, we need to specify the number of bins and bin width to train a GPR emulator. It introduces the Poisson uncertainty in each bin which is $\propto \frac{1}{\sqrt{N}}$, where N is the number of events in that bin. The number of events in some bins may equal zero even if the theoretical probability density is not zero; consequently, it will produce a large Poisson uncertainty. A sufficiently large number of events is needed to recover the theoretical probability density before training. For high-dimensional cases, the number of events needed to recover the

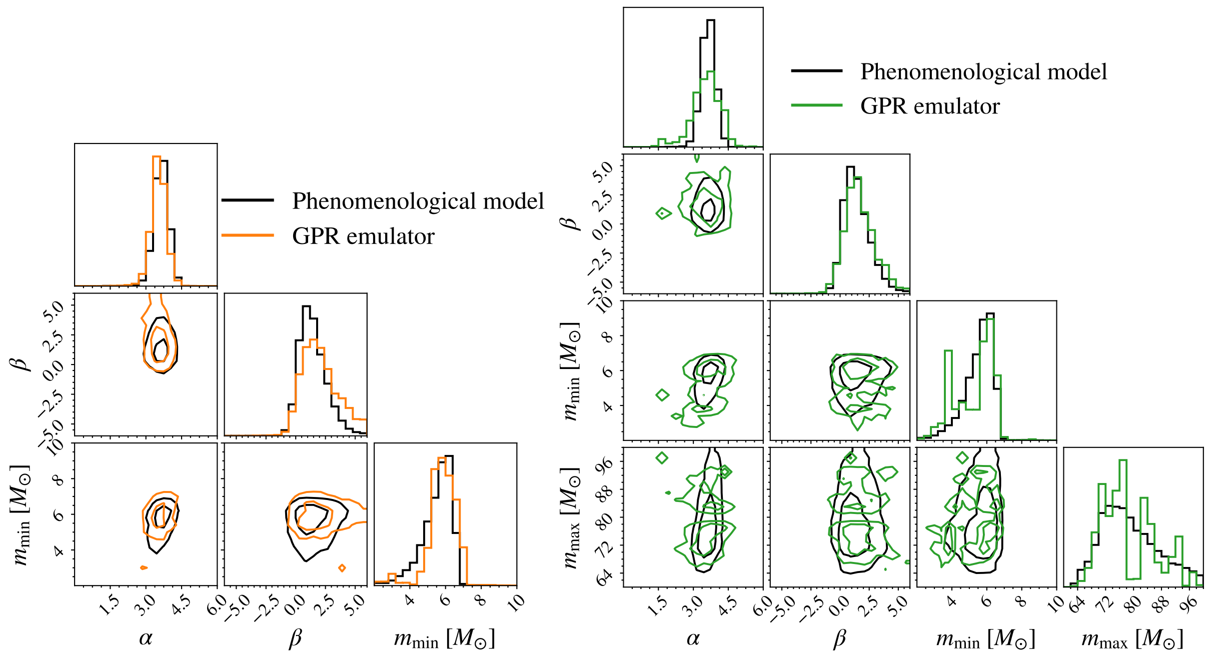


FIG. 6. Comparison of sampled posterior distributions on GWTC-2 data by using the phenomenological model and the GPR emulator. The black curves represent the sampled posterior distributions by using the phenomenological model, the orange curves (left) represent those by using the GPR emulator trained on only three hyperparameters $[\alpha, \beta, m_{\rm min}]$, while the green curves (right) are those by using the GPR emulator trained on all four hyperparameters:$[\alpha, \beta, m_{\rm min}, m_{\rm max}]$. The two contour levels represent the 50% and 90% credible regions of the sampled posterior distribution. For three hyperparameters, the GPR emulator can reconstruct the distribution using the phenomenological model, but not for four hyper-parameters. Even after training the GPR emulator with more data, the distributions persist to scatter and diverge.

theoretical joint probability density grows exponentially. To solve the problem, one possible approach is to divide the event parameter space into two regions—region 1 with plenty of samples and region 2 with fewer samples. Then we can continue drawing samples in region 2 until we have enough. After that, we can construct the GPR emulator with the reweighted samples. However, even if we used theoretical probability density, the number of bins still affects the resolution of the density estimation. The comparison shown in Fig. 6 demonstrates the inability of the GPR emulator on higher dimensions. We tried using more simulations as training data with the higher binning resolution, but the scatter and diverge problem persisted. The largest training dataset we tried took around a week to train using a 4-core CPU. Not to mention the time spent on generating the training data. As a result, training a good GPR emulator for population inference is unaffordable in terms of time and computational cost. The result may reveal GPR's limitations in estimating high-dimensional density [54,55]. GPR learns the model by inferring training data with a Gaussian prior. As a result, the predicted likelihood function will look like a sine curve that connects the training data; it has many local minimums and provides an explanation for the scattered posterior distribution in Figs. 3 and 6.

On the other hand, the NF emulator performs well except for underestimating the uncertainty for some hyper-parameter as shown in Fig. 7. But the uncertainty estimated by the emulator should be greater than the phenomenological model because of the limited training data. The problem of the underestimated uncertainty may come from the nature of the NF emulator [56]. NF is a series of continuous transformations so has relatively bad performance on learning truncation property. At the truncation of distribution, NF prefers a smooth change rather than a sharp truncation as shown in Fig. 8 at $m_1 = m_{\rm max}$. It requires infinitely many continuous transformations to get a sharp truncation. In addition, the training depends on the loss function which is the likelihood of the entire transformation. The loss function takes care of the entire training data at the same time so that it is unlikely to have binning and resolution problems. As a result, the uncertainty near the truncation will smooth out. Studies on controlling the uncertainty accumulated in the training should be carried out.

After understanding the characteristics of the emulator, we should use with caution when applying the techniques to those state-of-art models [16,57–59] for more sophisticated GW population studies. We can use the technique to eliminate the synthesis simulations which are not favored

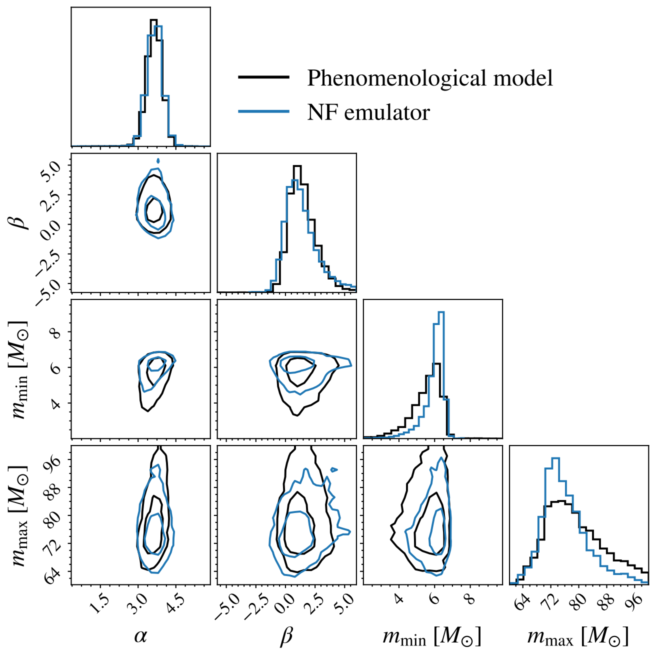


FIG. 7. Comparison of sampled posterior distributions on GWTC-2 data by using the phenomenological model and NF emulator. The black curves indicate the sampled posterior distributions by using the phenomenological model while the blue curves represent those by using the NF emulator. The two contour levels represent the 50% and 90% credible regions. The NF emulator can recover the distribution by using the phenomenological model except for the occurrence of underestimated uncertainty in the $m_{\rm min}$ and $m_{\rm max}$ distributions.

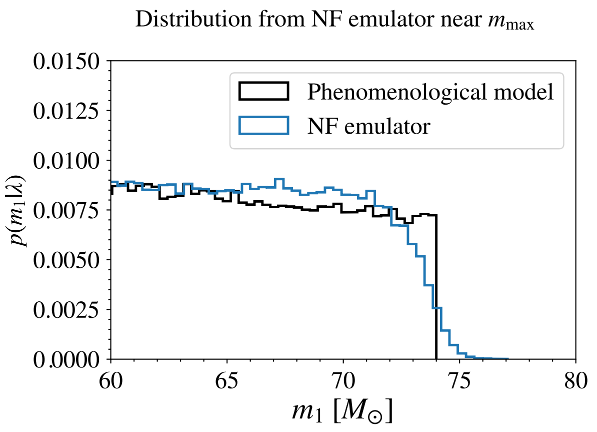


FIG. 8. The marginalized m_1 distributions near $m_{\rm max}$ on event sampling. The test hyperparameter are $[\alpha, \beta, m_{\rm min}, m_{\rm max}] = [3.0, 0.5, 6.0, 74.0]$ which approximately equal to the most probable inferred hyperparameters in Fig. 7. The black curves represent the marginalized probability density predicted by the phenomenological model while blue curves present those by the NF emulator. The NF emulator can not capture the truncation perfectly.

by the observation data by marginalizing the population posteriors from two models and computing the Bayes factor between the two models [24,60]. However, some simulations may be ruled out wrongly because of the underestimated uncertainty behavior of the NF emulator.

Furthermore, with the fast growth of the GW observed catalog, one potential research direction is to train an emulator using real GW data to sample the real GW distribution without employing any population model. However, each GW event is expressed as the posterior samples from the parameter estimation because of the measurement uncertainty. Machine learning frequently fails to handle such type of training data. In particular, the technique we used in the NF emulator is not regulated to train by using such type of data. One approach is to construct the Bayesian neural network [61] with NF. Studies and performance tests of this technique in GW population analysis should be carried out in the future.

ACKNOWLEDGMENTS

K. W. K. W. and S. H. are supported by the Simons Foundation. O. A. H. was partially supported by grants from the Research Grants Council of the Hong Kong (Project No. CUHK 14306218), T. G. F. L was partially supported by grants from the Research Grants Council of the Hong Kong (Project No. CUHK 14306419), The Croucher Foundation of Hong Kong and Research Committee of the Chinese University of Hong Kong. This project was conducted using computational resources at the Rusty cluster of the Flatiron Institute. This project made use of the following PYTHON packages: MATPLOTLIB [62], NumPy [63], SciPy [64], SCIKIT-LEARN [46], CORNER [65], EMCEE [65], and PyTorch [66]. This material is based upon work supported by NSF's LIGO Laboratory which is a major facility fully funded by the National Science Foundation. This research has made use of data, software and/or web tools obtained from the GW Open Science Center [67], a service of LIGO Laboratory, the LIGO Scientific Collaboration and the Virgo Collaboration. Virgo is funded by the French Centre National de Recherche Scientifique (CNRS), the Italian Istituto Nazionale della Fisica Nucleare (INFN) and the Dutch Nikhef, with contributions by Polish and Hungarian institutes.

APPENDIX: MOCK DATA CATALOG

Here is the detail about the formulas used to generate mock Gravitational-wave events catalog.

First, the distribution of m_1 follows a power-law distribution with spectral index α:

$$p(m_1|\alpha, m_{\rm min}, m_{\rm max}) \propto \begin{cases} {m_1}^{-\alpha} & m_{\rm min} < m_1 < m_{\rm max} \\ 0 & \text{otherwise.} \end{cases} \tag{A1}$$

Second, the mass ration $q = m_2/m_1$ follows a power-law distribution with spectral index β:

$$p(q|\beta, m_{\min}, m_1) \propto \begin{cases} q^\beta & m_{\min} < m_2 < m_1 \\ 0 & \text{otherwise.} \end{cases} \quad \text{(A2)}$$

And lastly, the red-shift distribution can be written as:

$$p(z) \propto (1+z)^{\kappa-1} \frac{dV_c}{dz} \quad z \in [0, 2.3], \quad \text{(A3)}$$

where κ is the redshift evolution parameter and is set to 1, and dV_c/dz is the differential comving volume.

TABLE I. Summary of truncated power-law hyper-parameters.

Hyper-parameter	Description	Prior
α	Power law index on m_1	$\mathrm{U}(-6, 6)$
β	Power law index on $q = m_1/m_2$	$\mathrm{U}(-6, 6)$
$m_{\min}$	Minimum mass for mass distribution m_1	$\mathrm{U}(2\ M_\odot, 10\ M_\odot)$
$m_{\max}$	Maximum mass for mass distribution m_1	$\mathrm{U}(60\ M_\odot, 100\ M_\odot)$

[1] B. Abbott, R. Abbott, T. Abbott, M. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. Adhikari *et al.*, Phys. Rev. D **93**, 122003 (2016).

[2] J. Aasi *et al.*, Classical Quantum Gravity **32**, 074001 (2015).

[3] F. Acernese *et al.*, Classical Quantum Gravity **32**, 024001 (2014).

[4] T. Akutsu *et al.* (KAGRA Collaboration), Nat. Astron. **3**, 35 (2019).

[5] R. Abbott, T. D. Abbott, F. Acernese, K. Ackley, C. Adams, N. Adhikari, R. X. Adhikari, V. B. Adya, C. Affeldt, D. Agarwal, M. Agathos, K. Agatsuma, N. Aggarwal, O. D. Aguiar, L. Aiello, A. Ain *et al.* (The LIGO Scientific, the Virgo, the KAGRA Collaborations), arXiv:2111.03606.

[6] B. Abbott, R. Abbott, T. Abbott, S. Abraham, F. Acernese, K. Ackley, C. Adams, R. Adhikari, V. Adya, C. Affeldt *et al.*, Phys. Rev. X **9**, 031040 (2019).

[7] R. Abbott, T. Abbott, S. Abraham, F. Acernese, K. Ackley, A. Adams, C. Adams, R. Adhikari, V. Adya, C. Affeldt *et al.*, Phys. Rev. X **11**, 021053 (2021).

[8] R. Abbott *et al.* (The LIGO Scientific, the Virgo, the KAGRA Collaborations), arXiv:2111.03634.

[9] M. Maggiore, C. V. D. Broeck, N. Bartolo, E. Belgacem, D. Bertacca, M. A. Bizouard, M. Branchesi, S. Clesse, S. Foffa, J. García-Bellido *et al.*, J. Cosmol. Astropart. Phys. 03 (2020) 050.

[10] D. Reitze *et al.*, Bull. Am. Astron. Soc. **51**, 035 (2019), https://baas.aas.org/pub/2020n7i035/release/1.

[11] B. P. Abbott, R. Abbott, T. D. Abbott, S. Abraham, F. Acernese, K. Ackley, C. Adams, V. B. Adya, C. Affeldt, M. Agathos *et al.*, Living Rev. Relativity **23**, 3 (2020).

[12] S. E. Perkins, N. Yunes, and E. Berti, Phys. Rev. D **103**, 044024 (2021).

[13] D. Wysocki, D. Gerosa, R. O'Shaughnessy, K. Belczynski, W. Gladysz, E. Berti, M. Kesden, and D. E. Holz, Phys. Rev. D **97**, 043014 (2018).

[14] D. Wysocki, J. Lange, and R. O'Shaughnessy, Phys. Rev. D **100**, 043012 (2019).

[15] G. Wiktorowicz, Ł. Wyrzykowski, M. Chruslinska, J. Klencki, K. A. Rybicki, and K. Belczynski, Astrophys. J. **885**, 1 (2019).

[16] J. W. Barrett, I. Mandel, C. J. Neijssel, S. Stevenson, and A. Vigna-Gómez, Proc. Int. Astron. Union **12**, 46 (2016).

[17] K. Belczynski, M. Dominik, T. Bulik, R. O'Shaughnessy, C. Fryer, and D. E. Holz, Astrophys. J. **715**, L138 (2010).

[18] S. R. Taylor and D. Gerosa, Phys. Rev. D **98**, 083017 (2018).

[19] K. W. K. Wong and D. Gerosa, Phys. Rev. D **100**, 083015 (2019).

[20] K. W. K. Wong, G. Contardo, and S. Ho, Phys. Rev. D **101**, 123005 (2020).

[21] R. Abbott, T. D. Abbott, S. Abraham, F. Acernese, K. Ackley, A. Adams, C. Adams, R. X. Adhikari, V. B. Adya, C. Affeldt *et al.*, Astrophys. J. Lett. **913**, L7 (2021).

[22] H.-Y. Chen, R. Essick, S. Vitale, D. E. Holz, and E. Katsavounidis, Astrophys. J. **835**, 31 (2017).

[23] S. Vitale, D. Gerosa, W. M. Farr, and S. R. Taylor, arXiv: 2007.05579.

[24] E. Thrane and C. Talbot, Pub. Astron. Soc. Aust. **36**, e010 (2019).

[25] S. Vitale, D. Gerosa, C.-J. Haster, K. Chatziioannou, and A. Zimmerman, Phys. Rev. Lett. **119**, 251103 (2017).

[26] C. Pankow, L. Sampson, L. Perri, E. Chase, S. Coughlin, M. Zevin, and V. Kalogera, Astrophys. J. **834**, 154 (2017).

[27] D. W. Hogg, A. D. Myers, and J. Bovy, Astrophys. J. **725**, 2166 (2010).

[28] I. Mandel, W. M. Farr, and J. R. Gair, Mon. Not. R. Astron. Soc. **486**, 1086 (2019).

[29] C. Talbot and E. Thrane, Astrophys. J. **856**, 173 (2018).

[30] Y. Himemoto, A. Nishizawa, and A. Taruya, Phys. Rev. D **104**, 044010 (2021).

[31] D. Martynov, E. Hall, B. Abbott, R. Abbott, T. Abbott, C. Adams, R. Adhikari, R. Anderson, S. Anderson, K. Arai *et al.*, Phys. Rev. D **93**, 112004 (2016).

[32] L. P. Singer, H.-Y. Chen, D. E. Holz, W. M. Farr, L. R. Price, V. Raymond, S. B. Cenko, N. Gehrels, J. Cannizzo, M. M. Kasliwal *et al.*, Astrophys. J. **829**, L15 (2016).

[33] H.-Y. Chen, D. E. Holz, J. Miller, M. Evans, S. Vitale, and J. Creighton, Classical Quantum Gravity **38**, 055010 (2021).

[34] G. Ashton *et al.*, Astrophys. J. **241**, 27 (2019).

[35] J. Veitch *et al.*, Phys. Rev. D **91**, 042003 (2015).

[36] W. M. Farr, Res. Notes AAS **3**, 66 (2019).

[37] D. Gerosa, dgerosa/gwdet: v0.1, 2017.

[38] K. Belczynski, V. Kalogera, F. A. Rasio, R. E. Taam, A. Zezas, T. Bulik, T. J. Maccarone, and N. Ivanova, Astrophys. J. Suppl. Ser. **174**, 223 (2008).

[39] J. R. Hurley, C. A. Tout, and O. R. Pols, Mon. Not. R. Astron. Soc. **329**, 897 (2002).

[40] M. Giersz, D. C. Heggie, J. R. Hurley, and A. Hypki, Mon. Not. R. Astron. Soc. **431**, 2184 (2013).
[41] N. Giacobbo and M. Mapelli, Mon. Not. R. Astron. Soc. **480**, 2011 (2018).
[42] D. Donovan, K. Burrage, P. Burrage, T. A. McCourt, H. B. Thompson, and E. S. Yazici, arXiv:1510.03502.
[43] M. Stein, Technometrics **29**, 143 (1987).
[44] M. Baudin, pyDOE, the experimental design package for PYTHON, 2013, https://github.com/tisimst/pyDOE.
[45] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning* (The MIT Press, 2016).
[46] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. Vanderplas, A. Joly, B. Holt, and G. Varoquaux, arXiv:1309.0238.
[47] G. Papamakarios, T. Pavlakou, and I. Murray, arXiv: 1705.07057.
[48] D. P. Kingma and M. Welling, arXiv:1312.6114.
[49] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, arXiv: 1406.2661.
[50] G. Papamakarios, T. Pavlakou, and I. Murray, arXiv:1705 .07057.
[51] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, arXiv:1605.02226.
[52] H. Liao and J. He, arXiv:2102.06539.
[53] S. Kullback and R. A. Leibler, Ann. Math. Stat. **22**, 79 (1951).
[54] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, arXiv:1807.01065.
[55] L. P. Swiler, M. Gulian, A. L. Frankel, C. Safta, and J. D. Jakeman, J. Mach. Learn. Model. Comput. **1**, 119 (2020).
[56] J. Hermans, A. Delaunoy, F. Rozet, A. Wehenkel, and G. Louppe, Averting a crisis in simulation-based inference (2021).
[57] N. Giacobbo, M. Mapelli, and M. Spera, Mon. Not. R. Astron. Soc. **474**, 2959 (2018).
[58] K. Breivik, S. Coughlin, M. Zevin, C. L. Rodriguez, K. Kremer, C. S. Ye, J. J. Andrews, M. Kurkowski, M. C. Digman, S. L. Larson *et al.*, Astrophys. J. **898**, 71 (2020).
[59] M. Dominik, E. Berti, R. O'Shaughnessy, I. Mandel, K. Belczynski, C. Fryer, D. E. Holz, T. Bulik, and F. Pannarale, **806**, 263 (2015).
[60] C. R. Jenkins and J. A. Peacock, Mon. Not. R. Astron. Soc. **413**, 2895 (2011).
[61] C. Louizos and M. Welling, arXiv:1703.01961.
[62] J. D. Hunter, Comput. Sci. Eng. **9**, 90 (2007).
[63] S. van der Walt, S. C. Colbert, and G. Varoquaux, Comput. Sci. Eng. **13**, 22 (2011).
[64] P. Virtanen, Nat. Methods **17**, 261 (2020).
[65] D. Foreman-Mackey, J. Open Source Software **1**, 24 (2016).
[66] A. Paszke *et al.*, in *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., Red Hook, NY, 2019), pp. 8024–8035, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
[67] https://www.gw-openscience.org.