University of Oviedo

Faculty of Sciences

Normalising flows for parameter inference on binary black hole mergers

Daniel Lanchares Álvarez

Master's Degree in Advanced Physics

Supervised by:

José Antonio Font & Joaquín Gonzalez-Nuevo

Oviedo

July 12, 2024

Abstract

This project follows the development of deep learning (DL) powered likelihood free inference of parameters from gravitational-wave data. Specifically, it focuses on the use of normalising flows conditioned by DL-extracted features of 2d (time-frequency) representations of gravitational waves from binary black hole (BBH) mergers. To train these estimators, datasets of simulated data are generated and fed to perform gradient descent. Since the project has required the custom implementation of much of its high level code infrastructure (the "Deep Tempest" library), its parallel development is also documented in this piece. The best of these models are presented in comparison to both simulation and real event data from all 3 currently available catalogues, against which it is found to have promising compatibility given its relatively low complexity and, most importantly, its speed both in training and sampling. This work makes a clear case for neural posterior estimation (NPE) alternatives to classic Bayesian estimation algorithms such as Markov-Chain Montecarlo (MCMC), at least in the realm of gravitational wave parameter inference.

Notes to the reader

The words that appear underlined at least once are defined or given more context on a glossary at the end of the document. One can access a word's entry by clicking on them. For the case of a web hyperlink it will appear specified (*e.g.* "this document" or "web page").

Following the conventions found in most general relativity textbooks, indices run from 0 to 3 denoting (t, x, y, z), with latin letters expressing purely spatial indices and greek letters for spacetime indices.

Since this work will feature in depth descriptions of object oriented code, I will colour-code this way:
`SampleDict` is my subclass of the `SamplesDict` class from PESummary, both shown in verbatim.

Bibliographic references appear color-coded as follows:
Scientific articles, textbooks and software presentation articles.

Every figure whose origin is not specified is of my own making.

Acknowledgements

The development of this work would have not been possible without the invaluable help and guidance of PhD student Osvaldo Freitas. Although administratively he may not appear as an official cosupervisor, he has effectively played that role.

Contents

1 A primer on gravitational waves

Although this phenomenon has taken a much more relevant place on the broader science scene since its direct detection on late 2015 by the LIGO detectors, gravitational radiation as a concept predates modern gravitation theories. A couple of decades before the advent of special relativity Oliver Heaviside (Heaviside 1893) had already entertained the idea of gravity propagating at a finite speed, even suggesting the possibility of it traveling at luminal speeds. Henri Poincaré would corroborate this idea (Poincaré 1905) as the Lorentzian nature of spacetime started to become apparent.

A decade later general relativity (GR from now on) would come about, and with it an explicit framework for gravitational waves. In order not veer too far off-topic I will gloss over the details, though comprehensive derivations can be found in most GR textbooks, such us Weinberg 1972; Carroll 2014, or Guidry 2019. By linearizing Einstein's equations on the weak field limit we can get the radiative perturbations of a background metric such that the combined metric is $g_{\mu\nu} = g^0_{\mu\nu} + h_{\mu\nu}$ with $\bar{h}_{\mu\nu} \equiv h_{\mu\nu} - \eta_{\mu\nu}h^\alpha_\alpha/2$ obeying:

$$\partial_\beta \partial^\beta \bar{h}_{\mu\nu} = -\frac{16\pi G}{c^4} T_{\mu\nu}. \tag{1.1}$$

As expected, it is a wave equation, though for now we will concern ourselves with flat vacumm solutions ($g^0_{\mu\nu} = \eta_{\mu\nu}$). Assuming propagation aligns with our z axis and choosing a specific gauge (transverse-traceless), the plane wave solution to the equation has the following form, which is then showed generalized:

$$h_{\mu\nu} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & A_+ & A_\times & 0 \\ 0 & A_\times & -A_+ & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} e^{i(\omega t - k_z z)} \qquad \Rightarrow \qquad \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & f_+(t-z) & f_\times(t-z) & 0 \\ 0 & f_\times(t-z) & -f_+(t-z) & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{1.2}$$

From equation 1.2 we can infer a surprising amount of information. Most notably it contains all polarization information. According to GR gravitational waves ought to have two and only two independent polarizations, both transverse and tensorial in nature. If we pair this expression with the "Lorenz condition" ($\partial^\mu h_{\mu\nu} = 0$) we get the imposition $k_\nu k^\nu = 0$, which is a rigorous way of stating that the wave vector has to be light-like and therefore that the speed of spacetime radiation is the speed of light. This is particularly relevant as many alternative theories of gravity introduce elements that alter these properties, but since this work pertains itself to better estimating based on current models I will not go into further detail (if curious see Ezquiaga and Zumalacárregui 2018; Will 2018; B. P. Abbott et al. 2018; B. P. Abbott et al. 2019(b); R. Abbott et al. 2021(b)).

To understand its behaviour more completely it would be wise to study its effect on point particles. By separating two point particles with a vector $\vec{x}$ on the XY plane, the wave will cause the distance between them to (as one might expect) undulate as:

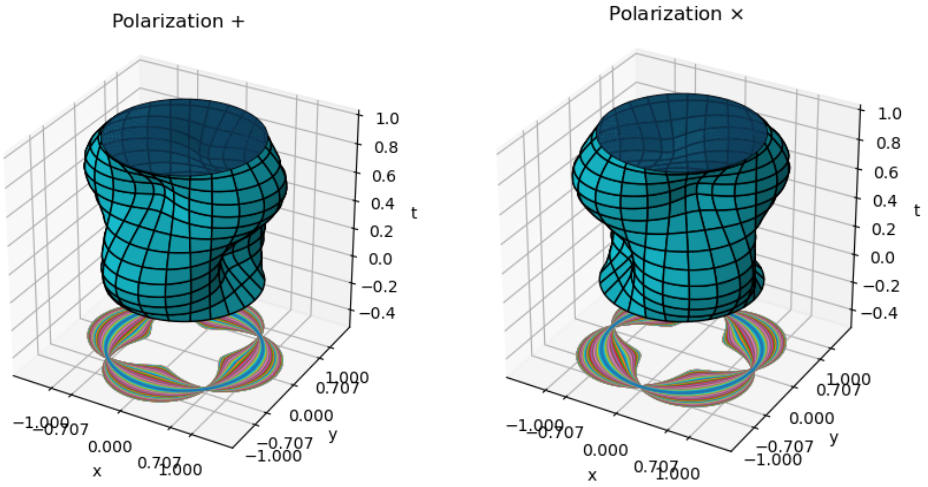
$$L_0^2 = \vec{v}\cdot\vec{v} = v^\mu v^\nu g_{\mu\nu} = v^\mu v^\nu \eta_{\mu\nu} + (v^x)^2 h_{xx} + 2v^x v^y h_{xy} + (v^y)^2 h_{yy}. \tag{1.3}$$

Substituting the metric perturbation of the plane wave and grouping by polarizations we arrive at:

$$L_0^2 = L^2_{0(\eta)} + (A_+((v^x)^2 - (v^y)^2) + A_\times 2v^x v^y) e^{i(\omega t - k_z z)} = L^2_{0(\eta)} + \delta L_0^2. \tag{1.4}$$

Now we can study the deviation of L_0^2 for $\vec{v} = L_{0(\eta)}(\cos(\theta), \sin(\theta))$, with $L_{0(\eta)} = 2$ to lie in the unit circle.

Dividing between the original proper length we obtain a dimensionless quantity, g-strain ($\delta L_0/L_0$), which will be of utmost importance to us, as it is the signal that comes out of a detector and into our pipeline.



(a) How a wave of amplitude 0.8 with '+' polarization deforms a ring of particles for a complete cycle ($\omega = 2\pi$).

(b) How a wave of amplitude 0.8 with '×' polarization deforms a ring of particles for a complete cycle ($\omega = 2\pi$).

Figure 1: Visualization of the canonical base of polarizations for a gravitational wave.

Discussions on whether these results were physical or detectable continued throughout the XXth century, involving the likes of Einstein, Rosen, Robertson... The community started reaching a consensus after Richard Feynman weighted in with his 'sticky beads' analogy. In it, he imagined to beads free to slide on a bar, though with a certain friction. When a gravitational wave passes, spacetime distortions cannot do much to compress the bar, but the beads will move slightly, dissipating energy through friction. With a sufficiently clean background the measure of that energy implies that we have detected a gravitational wave.

While hypothetical analogies are all well and good, we need more specifics to seriously consider detecting these waves. In particular, we need to understand how they are generated, which requires us to recover equation 1.1 and invert it using the D'Alambertian's Green function:

$$\bar{h}_{\mu\nu}(x^\sigma) = -\frac{16\pi G}{c^4}\int G(x^\sigma - y^\sigma)T_{\mu\nu}(y^\sigma)d^4y = \frac{4G}{c^4}\int \frac{T_{\mu\nu}(t-|\vec{x}-\vec{y}|/c,\vec{y})}{|\vec{x}-\vec{y}|}d^3y. \tag{1.5}$$

By assuming a few approximations (that the source of scale R is compact and far away: $|\vec{x}-\vec{y}| \approx \vec{x} \equiv r$, $\lambda \gg R$ and moving at non-relativistic speeds) and doing some Stoke's theorem sorcery we can leave the integrand as proportional to $\partial_t^2(T^{00}y_iy_j)$. Commuting integrals and derivatives we can define de quadrupole moment Q_{ij}. In the transverse-traceless gauge that we have been using all along we can rewrite Q in terms of the moment of inertia, I, leaving us with:

$$h_{ij} = -\frac{2G}{c^4}\frac{\ddot{I}_{ij}}{r}. \tag{1.6}$$

With this last equation we can get a ballpark estimation of what we should expect to be able to detect, keeping

in mind that g-strain is a dimensionless quantity, but related to a quotient of lengths. We already see that prospects are complicated since $G/c^4 \sim 10^{-45}\mathrm{s}^2\mathrm{kg}^{-1}\mathrm{m}^{-1}$, but we need an estimation for I and r. The efforts of the community at the moment are centered around CBCs of stellar mass black holes, so that is what we will tackle next.

1.1 Basic modeling of CBCs

The formulas we have seen currently are general enough as to allow both continuous and transitory emission of gravitational radiation. In fact, some of our protagonists (neutron stars) are thought to be capable of both, but as of writing no resolvable continuous gravitational waves have been observed. Early stages of a CBC can be considered mostly continuous, but our cutting-edge technology is yet to reach further than the last moments, when the peak in amplitude rises several orders of magnitude above previous stages.
We have been able to detect 'neutron star & neutron star' collisions (*BNS s*, *GW*170817 and *GW*190425: B. P. Abbott et al. 2017; B. P. Abbott et al. 2020(b)), as well as 'black hole & neutron star' (*BHNS s*, *GW*200105 y *GW*200115: R. Abbott et al. 2021(a)), but the overwhelming majority of detections comprise two black holes (*BBH*, for example: B. P. Abbott et al. 2016(a)). Mass gap events such as GW190814 (R. Abbott et al. 2020[a]) are also as rare as they are interesting.

Although resolving Einstein's equations in such a complex configuration is no easy feat even for the most advanced computer clusters, we can give a rough estimate of h with a few assumptions: By considering a circular orbit, which we will presume static (this violates energy conservation, but is acceptable on early stages of coalescence), of radius $l \gg (r_s)_n$, r_s being each objects Schwarzschild radius, we can parametrize:

$$x_1(t) = l\cos(\omega t) \qquad y_1(t) = l\sin(\omega t) \qquad z_1(t) = 0 \tag{1.7}$$

$$x_2(t) = l\cos(\omega t + \pi) \qquad y_2(t) = l\sin(\omega t + \pi) \qquad z_2(t) = 0 \tag{1.8}$$

With these conditions we can treat the compact objects as point particles when calculating the moment of inertia: $I_{ij} = (m_1 + m_2)x_i x_j$, which when substituted on equation 1.6:

$$h_{ij} = \frac{4G\omega^2\mu l^2}{rc^4}\begin{pmatrix} -\cos(2\omega(t-r/c)) & -\sin(2\omega(t-r/c)) & 0 \\ -\sin(2\omega(t-r/c)) & \cos(2\omega(t-r/c)) & 0 \\ 0 & 0 & 0 \end{pmatrix} = A \cdot P_{ij} \cdot e^{i(2\omega t - kr)}. \tag{1.9}$$

Where A is the wave's amplitude and P is the matrix that encodes polarization. Is interesting to note that the frecuency of the wave is twice that of the system that generated it. We can simplify even further if we assume equal masses and distances, which then evaluated on reasonable values ($f = 300\mathrm{Hz}$, $r = 500\mathrm{Mpc}$, $M = 35\mathrm{M}_\odot$, $l = 50000\mathrm{km}$) give:

$$A = \frac{8GMl^2\omega^2}{rc^4} \sim 2 \cdot 10^{-21}. \tag{1.10}$$

Here lies the main problem facing gw (gravitational wave) astronomy: Their amplitude is ridiculously small, so to record its passing we would require measuring objects thousands of kilometers apart with subatomic precision. This may lead some to think that the earth cannot hold such a facility, but in hindsight it is perfectly feasible (tough the specific challenges are outside the scope of this work and are therefore left as an exercise to the reader: See Saulson 2013; Aasi et al. 2015; Vajente et al. 2019; Fidecaro's Chapter on Coccia et al. 2020 and B. P. Abbott et al. 2020(a)). In this next Section we will delve deeper into what we are *actually* looking for and therefore what I had to simulate in order to train my estimators: Waveforms.

1.2 Advanced modeling: Waveforms

Although equation 1.10 gives a very useful back-of-the-envelope result, I am not going to adequately estimate any parameters from it, as it is full of approximations and only accounts for a handful of them. As we will see later the former is almost inevitable when dealing with strongly interacting GR, but the later can be taken care of with some better models.

We can divide a coalescence into three fairly distinct phases: inspiral, merger and ringdown, pictured in figure 2.

- The **inspiral** phase can take millions of years in which, as I've mentioned previously, the system is emitting almost continuously, though in frequencies and intensities far too low for our current detectors to hear anything but the last periods before merging, when frequency starts increasing rapidly. These can range from milliseconds to minutes depending on the mass of the system. This fact will condition our ability to estimate low mass coalesces.
- The actual **merger** sees both the frequency and luminosity peak, with the later reaching upwards of $\sim 10^{50} W$ or $\sim 10^{23} L_\odot$ (B. P. Abbott et al. 2016[b]). This increase in pitch is referred to as chirp, and it is indeed audible if the signal is played on speakers, as typical signals range from tens to hundreds of hertzs. It also gives its name to the chirp mass, a vital parameter that I will introduce shortly.
- Finally, on the **ringdown** phase the system returns to equilibrium, settling down into the final black hole. This process is of special interest to theoretical physicists specialized in black holes, as it provides an experimental test for perturbation theory in these yet so mysterious objects.

During the inspiral the gravitational field is sufficiently weak to afford a post-Newtonian approximation. As long as the object's speeds are non or low-relativistic we can expand as a series of powers of $(v/c)^2$ or other related parameters ($\alpha = GM/lc^2$ is particularly interesting as it relates mass and orbit radius). It is commonplace to denote terms as 'of order kPN', with $k \in \mathbb{N}/2$, when they are $O(1/c^{2k})$. Working at $3.5PN$ (Andersson 2019; Blanchet 2014) we end up at an expression for the evolution of the orbital frequency (remember that, even if I haven't proved it, it holds that $f_{GW} = 2f_{Orb.} = \Omega_{Orb.}/\pi$):

$$\frac{d\Omega_{Orb.}}{dt} = \frac{96}{5}\frac{GM\eta}{r^3}\alpha^{5/2}\left(1 + A(\eta)\alpha\right) + O\left(\alpha^4 \sim \frac{1}{c^8}\right). \tag{1.11}$$

Where $\eta \equiv \mu/M$ is the symmetric mass ratio, a possible parameter of ours. However, I have used the 'regular' mass ratio in this work ($q \equiv m_</m_>$) as it is the most often used and goes from 0 to 1, making it ideal for machine learning. With similar proceedings one can find a generalization of Kepler's third law: $\Omega^2_{Orb.} = GM/r^3(1+A'(\eta)\alpha + B'(\eta)\alpha^2 + C'(\eta,\eta^2,\eta^3)\alpha^3)$. By choosing a different perturbation parameter x depending on Ω instead of l we finally reach:

$$\frac{df_{GW}}{dt} = \frac{96}{5}\frac{G^{2/3}M^{5/3}\eta}{c^2}\pi^{8/3}f_{GW}^{11/3}\left(1 + \mathcal{A}(\eta)x\right) + O\left(x^4 \sim \frac{1}{c^8}\right). \tag{1.12}$$

The reason behind the employment of this formalism has just become apparent to the keen eye. We have shown that the frequency evolution is proportional to $(M\eta^{3/5})^{5/3}f_{GW}^{11/3}$, with $M\eta^{3/5}$ being interpretable as a pseudo-mass. Concretely, we have defined the **chirp mass**:

$$\mathcal{M} \equiv \eta^{3/5} M = \mu^{3/5} M^{2/5} = \sqrt[5]{\frac{(m_1 m_2)^3}{m_1 + m_2}}. \tag{1.13}$$

Using $\mathcal{M}$ in conjunction with q or η is generally a good idea, even though successful estimations are possible with m_i as well, because the shape of the signal is easily matched to this parameter by equation 1.12. This fact will prove incredibly useful in this work in particular, since we will be learning from the spectrogram itself, where the chirp is most visible.
Figure 2 shows a noiseless waveform with its associated spectrogram [1]

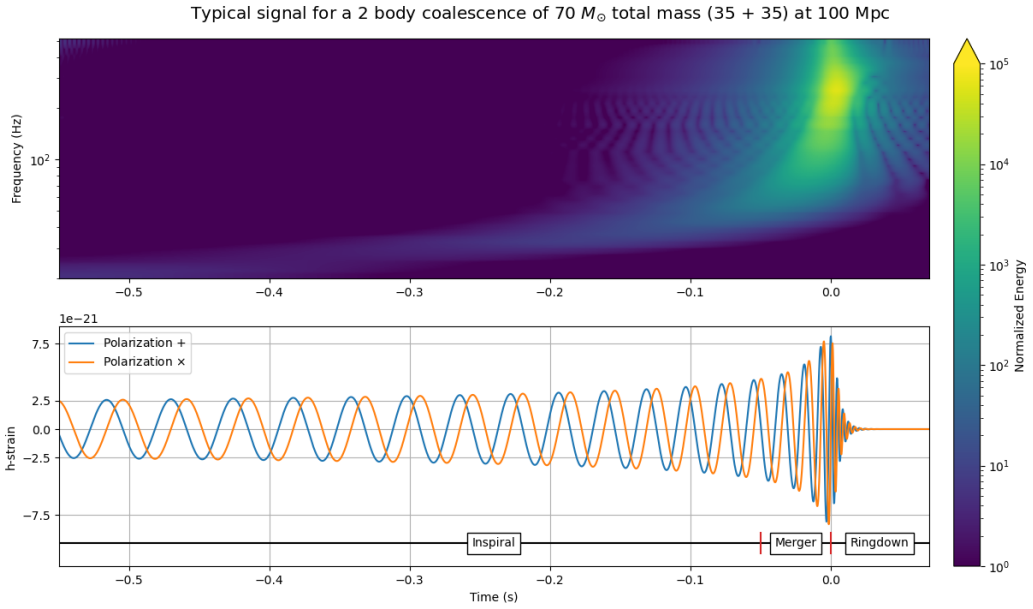


Figure 2: Signal to be expected from a CBC coalescence (a BBH system of 35 solar masses each, to be precise, which is compatible with the first ever detection from LIGO). Waveform generated by the *SEOBNRv4* approximant (more on it shortly) through the PyCBC library. The lower plot shows timeseries data for both plus and cross polarizations. Since both masses are equal they only differ on phase.
The upper one shows the Q transform representation for the '+' polarization. Both show an incremental rise in both amplitude and frequency respectively as the objects get ever closer, followed by short merger and ringdown phases, demarcated below.

Although we arrived at a vital result (equation 1.13) through perturbative methods, modeling CBCs will eventually require the whole theory, and GR is not an easy theory to work with. We lack analytical solutions to this problem and numerical simulations can, in the worst cases, run on timescales similar to current observation campaigns. For this reason Phenomenological models have been developed. Even though for the purpose of this work approximants can be thought of as black boxes that spit out a waveform given a set of parameters, I consider it worth our time to delve deeper on these miracles of ingenuity.
When using the word "approximant" I refer to a model that returns a waveform faster than numerical solutions of general relativity (by a few orders of magnitude) at the cost of accuracy.

[1] Throughout this work I will be using the word spectrogram to refer to a generic frecuency vs time representation of a signal, which will most times be a Q transform, but could also be a fourier-based spectrogram, a wavelet transform or similar.

There are two main families of approximant models:

- EOBNR: ***E**ffective **O**ne **B**ody + **N**umerical **R**elativity*.
  As the name suggest, they generate waveforms within the effective one body formalism (that is, approximating the system by a test particle and an effective metric) calibrated to numerical relativity simulations (Taracchini et al. 2014). The latest as of writing is `SEONNRv5` (Pompili et al. 2023), calibrated to 442 NR simulations and 13 black hole perturbation theory waveforms.
- IMRPhenom: ***I**nspiral-**M**erger-**R**ingdown **Phenom**enological*.
  Unlike the previous family, these are phenomenological models that clearly separate between stages of coalescence and incorporate behaviours "by hand". By the time the first detections took place production-ready models were only apt for non-precessing black holes (Khan, Husa, et al. 2016), but since then precessing (Hannam et al. 2014; Khan, Chatziioannou, et al. 2019) as well as high-multipole (García-Quirós et al. 2020; Pratten et al. 2021) and eccentric (Ramos-Buades et al. 2021) approximants have been developed. Throughout this work I used `IMRPhenomPv2` (Bohé et al. 2016), which is the most widely used at the moment (it is employed by most parameter inference references).
- Although these are the most common, other solutions are being explored, such as surrogate models for numerical relativity (Field et al. 2014; Tiglio and Villanueva 2022).

Now that we have a tool to generate data (vital if we were hoping to train an ML model with there being only 90 events published at the time of writing) we can finally start describing CBCs with all possible parameters.

1.3 The full parameter space

The parameter space of a CBC will depend on the assumptions one is willing to make (Can masses be approximated as equal? Or are spins supposed to be aligned? Is the orbit circular? Etcetera). A comprehensive table of possible parameters can be found on appendix A, table 9, though an intuitive rundown of the most common is given here, keeping track of the number os parameters describing each phenomenon:

- The masses of the two bodies: M_1 & M_2, or a combination of the two such as $\mathcal{M}$ & q (**2p**).
- Angular momentum of spin of both: $\vec{S}_1$ & $\vec{S}_2$ or angular renditions a_1, θ_1 & a_2, θ_2 + ϕ_{12}, ϕ_{JL} (**2p+4p**). If spins are presupposed to be aligned only a_1 & a_2 are needed to encode spin, hence the 2+4 notation. Effective and precession spins (χ_{eff} & χ_{p}) are also popular choices to be discussed later.
- Sky position and luminosity distance: α & δ + d_{L} (**3p**). Luminosity distance can be related to redshift and comoving distance through the Hubble parameter as $d_{\mathrm{L}} = (1+z)\int_0^z dz'/H(z')$. This, however, forces us to choose a particular cosmology.
- Inclination of the orbital plane and polarization angle: θ_{JN} & Ψ (**2p**). If spins are aligned then θ_{JN} is equivalent to ι, understandable as θ_{LN}.
- Time and phase at coalescence: t_{c} & ϕ_{c} (**2p**). The latter defined such that $\max(h) \sim |e^{i\phi_{\mathrm{c}}}|$.
- If we aspire to model BNSs we also need tidal distortion parameters Λ_1 & Λ_2 or if preferred $\tilde{\Lambda}$ & $\delta\tilde{\Lambda}$ (**2p**).
- Lastly, one could drop the circular orbit assumption with eccentricity and argument of periapsis, e & ω. This front is very much at its early stages, as no approximant can model precession and eccentricity at the same time. Despite this, literature is starting to appear (Huerta et al. 2018; I. Romero-Shaw et al. 2021; Wagner and O'Shaughnessy 2024) (**2p**).

This makes for a grand total of nineteen parameters. However, as mentioned above, it is currently impossible to model them all at the same time. Most "full" estimations tend to have 17 parameters for BNSs and 15 for BBHs. Coalescence time can prove tricky to implement (see Dax et al. 2021(a) for Dingo's solution), so I will avoid it entirely and settle for a maximum of 14 parameters for estimation of BBHs. As we will se in Section 4.2, an estimator of much lower dimensionality can also achieve solid results.
Aside from having 2 fewer parameters, a reason to avoid non-BBH events is data storage. Since frequency evolution is tied to chirp mass (equation 1.12) lower mass CBCs will take longer, and therefore occupy more diskspace, reducing dataset size. This is reflected in the chosen generation ranges in table 7, and it also means we would expect a generally worse performance for lower mass events.

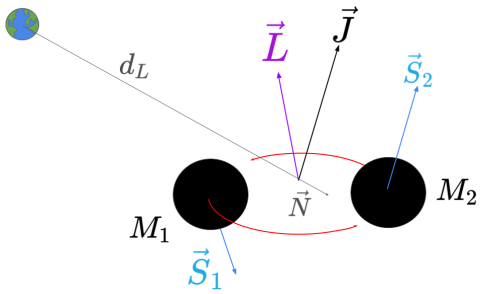


Figure 3: Diagram showing some of the parameters of a CBC.

To finish off this Chapter I want to give a few clarifications on various parameters.
We have already introduced the chirp mass, $\mathcal{M}$, but further points in its favor are made in B. P. Abbott et al. 2016(c). q is not as easily retrievable, and confusion can arise as some define it as $m_>/m_<$ and therefore bigger than one, which does not suit our needs. η, the symmetric version, goes from 0 to 0.25, which for ML purposes has the exact opposite problem: it would require to be upscaled.
Because gravitational waves are reaching us from cosmological distances and detectors are getting ever more sensitive the masses we recover on the detector frame are not the actual masses. To sample them we need the event's redshift (and therefore a specific cosmology) and the relations on table 9.
Most parameter-oriented libraries use the *Plank15 Cosmology*[2] (Planck Collaboration 2016) even if more modern results (*Plank18*, Planck Collaboration 2020) exist, probably for consistency between observation campaigns.

Regarding spins, a_i are defined as $|\vec{S}_i|c/GM_i^2$ to be dimensionless and between 0 and 1, as from the Reissner-Nördstrom metric one can deduce the inequality 1.14. By assuming a neutral object (generally reasonable to do) we obtain the [0, 1] restriction.

$$\frac{Q^2}{4\pi\epsilon_0} + \frac{c^2|\vec{S}_i|^2}{GM^2} \leq GM^2. \tag{1.14}$$

If one wants to have a basic modelling of spin that still gives a complex picture, effective and precession spin parameters are great.
As the name suggests, the former expresses the 'amount' of spin of the entire system through a mass-weighted average of orbital-momentum-aligned spin while the later gives a normalized idea of how well would the system be approximated by aligned spins, with 0 being equivalent to complete alignment and 1 to maximum precession (More information on Section 9.7 of Filipović and Tothill 2021). They are defined from χ_i and $\chi_i^\perp$, which are formed from a_i and θ_i (See table 9).

$$\chi_{\text{eff}} = \frac{(m_1\chi_1 + m_2\chi_2)}{m_1 + m_2}, \qquad \chi_{\text{p}} = \max\left(\chi_1^\perp, q\frac{3q+4}{4q+3}\chi_2^\perp\right). \tag{1.15}$$

Apart from being problematic to model, there is another reason to discard eccentricity when modeling a CBC. Gravitational radiation has the effect of circularizing orbits (Peters 1964) over long periods of time, so low eccentricities are to be expected (which makes for an interesting test of GR once approximants catch up). As previously alluded, the difficulties are not physical, but purely technical. More information can be found on Appendix E of I. M. Romero-Shaw et al. 2020.

[2]Deduced from bilby's source code

2 The problem at hand: (Neural) Posterior estimation

Now that we are comfortable with gravitational waves we should move on to the other main topic described in this work's abstract: NPE. The formal definition can be found in the glossary. This Chapter is aimed instead at giving a glimpse into why it is so relevant and the current state of the art, which I hope may introduce the reader to the main technical concepts that will be discussed in Section 3 and indirectly throughout the rest of the document.

Parameter inference boils down to the following:
Before we pick up any signal, we already know that, for example, masses and distances will be defined positive, or that angular parameters should be within $[0, 2\pi]$ or similar and have periodic bounds. We even have some information about the expected distribution of these parameters over a population. If CBCs occur uniformly in the comoving volume (which is reasonable given we have cosmological detection ranges) then this imposes certain constraints in luminosity distance and stellar position, such as declination being uniform in cosine instead of directly. We can encode this **prior** knowledge as a probability distribution of how likely it is for a generic event to have each possible value[3] for each parameter, $p(\theta)$.

On a first approach to the problem one could try a few parameter combinations and see which produces a signal closest to what we measured[4]. What we are technically doing is maximizing the likelihood, $p(x|\theta)$, which is the probability of measuring the signal given a set of parameters, and while it will return the best fit, we won't have many clues regarding uncertainty.
We can use this information to "update" our prior knowledge in the form of a posterior distribution, $p(\theta|x)$, related with prior and likelihood through Bayes' theorem[5]:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} \tag{2.1}$$

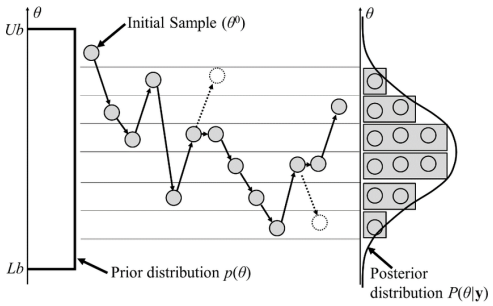


Figure 4: The Metropolis-Hastings algorithm sampling a normal one-dimensional posterior probability distribution (Lee et al. 2015).

Up until the late 2010s and into the 2020s, the landscape has been dominated by this equation and algorithms implementing it. The most successful has been MCMC, and in particular the Metropolis-Hastings algorithm (Metropolis et al. 1953; Hastings 1970. Figure 4).
This methodology powered LALInference, the original GW estimation pipeline (Veitch et al. 2015; Berry et al. 2015) and its complementary fast sky localization tools (Singer and Price 2016), as well as more user-friendly software such as bilby (Ashton et al. 2019) or PyCBC's inference module (Biwer et al. 2019). For thorough review of GW-oriented parameter estimation see Christensen and Meyer 2022.

Nonetheless, its characteristic slowness[6] makes it a weak link in discovery pipelines, as multi-messenger events are set to become increasingly common and therefore fast localization may be required in large numbers. For that reason much of the discussion surrounding advances in parameter estimation has revolved around machine learning and in particular the concept of normalizing flows.

[3] Since we are dealing with continious parameters, a range of values is what is well-defined, the probability of any unique value is 0.

[4] When a signal is detected through match-filtering the parameters of the closest templates can give an initial guess as to the parameters of the signal.

[5] The denominator of equation 2.1, $p(x) = \int p(x|\theta)p(\theta)d\theta$, is often omitted in actual calculations by constraining the posterior to be normalized, and it is therefore not discussed in detail.

[6] a 15 parameter estimation with bilby is set to take on the order of the tens of hours according to Section 3.3.4 of I. M. Romero-Shaw et al. 2020, and producing $\sim 10^4$ independent samples requires $\sim 10^6$ waveform evaluations (Dax et al. 2021[b])

2.1 Normalizing flows amongst the wider machine-learning revolution

When one thinks of machine learning usually two things come to mind: ChatGPT misinterpreting your words and a picture of a cat labeled as "boat". Neither of these are parameter estimation, but the second one, classification, is close enough to be a useful start. In that case a neural network (usually convolutional) learns to transform a given input through successive filters and a final softmax into a vector of 0 to 1 floating point values. The position of the highest value corresponds to the label the model thinks should classify the input, and the difference between the fist few classes gives an idea of the model confidence, a sort of uncertainty.
However, that isn't the kind of uncertainty an estimation needs. An estimation requires samples, a number of them large enough to obtain meaningful median and deviation information. This is a problem because every step of this classificator model is deterministic. If we rerun the image it will give the exact same output, so if we use it as parameter values it will simply return Dirac delta distributions, which unless we really are that confident in our algorithms it is something we ought to avoid.

With that said, there are a multitude of ways of incorporating uncertainty into machine learning models. The most simple one would be to just train more networks, each giving you a sample to build your distribution. If you are performing a relatively simple classifying task -say, for instance, b-tagging- you could get away with training a few hundred simple models and averaging the results. You are going to need more robust methods to get > 98% accuracy (see table 1 of Butter et al. 2019), but with a simple forest of models I have been able to achieve up to 81% on a class lab.

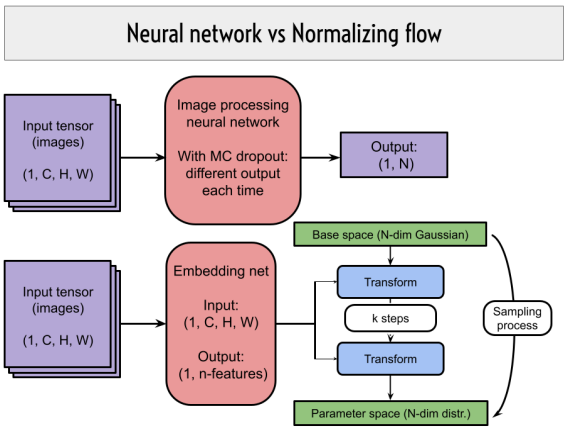


Figure 5: Diagram showing the difference in the forward passes of an MC dropout network (upper) and a normalizing flow (lower).

Of those more robust methods the most common is Montecarlo Dropout on a neural network (Gal and Ghahramani 2015; Gal and Ghahramani 2016). It requires, on a first approximation, turning on and off arbitrary connections between neurons of the network with a certain degree of randomness both during training and especially during sampling. The aforementioned papers showed that this technique could be used to approximate Bayesian inference in a deep learning setting. However, it does have difficulties approximating complex distributions (in GW inference we may expect single parameters with up to 3 modes) and requires optimization of its hyperparameters: dropout rate and weight regularization.

An example of MC Dropout applied to gravitational wave estimation can be found in Álvares et al. 2021, the publication from which I have based much of my methodology and which I will discuss further in Section 2.2.
Unlike on that paper, I was suggested to use a different approach to machine-learning estimation: Normalizing flows.

The typical neural network can be thought of as a function that, given an input, transforms it to return an output. Introducing dropout randomizes the output such that each time you introduce an identical input it transforms it into a new sample.
In a flow-based architecture, however, the input is not transformed into any particular sample, but it is instead used as context to transform a base distribution (usually Gaussian) into the desired distribution (figure 5). Once the input has set the transformation settings, the base -simple- distribution is sampled n times and each sample is transformed and then given to the user.

The key to make this work is the invertibility of the transforms. A composition of invertible functions is

invertible and therefore some variation of backpropagation algorithms can be performed. In practise what we want is to minimize the difference between the distribution we want to emulate and what our model returns. This is known as the Kullback–Leibler divergence, and with some work (see Section 2.3 of Papamakarios, Nalisnick, et al. 2021) it can be shown that minimizing the KL-divergence is equivalent to maximizing log-probability. To enable the use of gradient descent or any of its counterparts we can instead minimize the negative log-probability.

Thanks to this equivalence, training a normalizing flow is no different from training a conventional neural network. In fact, many of the transforms used in this work are controlled by their own smaller neural networks. Let's go over some of the transforms used, all of which are implemented in my `flow_utils` module from the building blocks found in nflows, a dependency of this project.

- `random_perm_and_lulinear` is the basic middle and final transform of my models (see Subsection 3.1.2). As the name suggests it consists of a random permutation (random, but fixed once drawn) followed by a LULinear transform initialized as the identity. It does not take context into account.
- `mask_affine_autoreg` is a combination of a reverse permutation and the titular autoregressive transform. **Autoregressive** implies each stage of the transform depends only on previous stages. Since the typical neural network is fully connected this property is enforced by applying a **mask** to the weight matrices to erase unwanted connections (see the MADE nets of Germain et al. 2015, implemented in nflows). Finally, it is called **affine** for the affine transform it performs (equation 18 of Green, Simpson, et al. 2020) For more information on MAFs in general see Papamakarios, Pavlakou, et al. 2017.
- `mask_piece_rq_autoreg` uses more complex transformations: monotonic piecewise functions, and in particular, rational-quadratic splines (equation 4 of Durkan et al. 2019) to approximate more complex distributions.
- `dingo_rq_coupling` is a fork of Dingo's implementation of a rational-quadratic coupling transform (Durkan et al. 2019). Coupling transforms are elementwise, they split the data in two and use one half to act on the other half, as pictured in figure 6.

All the models presented in this piece are composed of `random_perm_and_lulinear` secondaries with the primaries being affine transforms either on their own or interlaced between rq-couplings. Various other spline-based transforms exist: linear, quadratic, cubic... polynomial in general (Müller et al. 2019). However, these require their input domain (parameter ranges) to be $[0, 1]$, so without a fool-proof normalization they are not feasible.

Most of the design philosophies of the tools reviewed so far are "likelihood-free", meaning they do not rely on calculations of the likelihoods as equation 2.1 would suggest. This is not the case for MCMC, which does require evaluations. Prior information is implicitly recorded in the parameter ranges of the training data, which for this work are available in table 7.

These more advanced tools are what is often referred to as ***Neural Posterior Estimation***, or NPE for short, and many of developments in the field have come about from within the gravitational wave community (e.g. Kolmus et al. 2024 and Leyde et al. 2024, with the latter building upon Dingo, which will serve as a comparison point in the next Section).

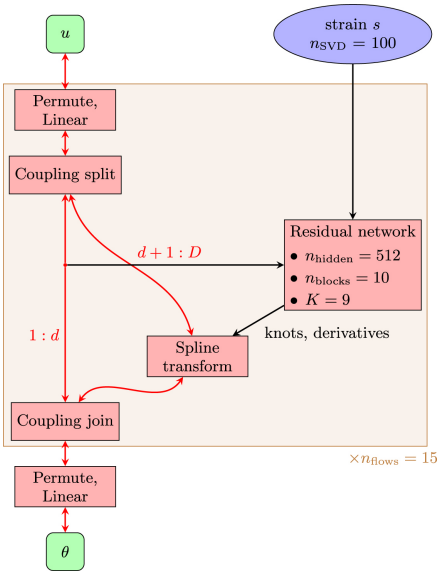


Figure 6: Diagram showing Dingo's implementation of the rational-quadratic spline coupling transform (Green and Gair 2021).

2.2 NPE in gravitational wave astrophysics

The field is widening fairly quickly in various directions (Ruhe et al. 2022; Langendorff et al. 2023; De Santi et al. 2024), but the most developed project in the field has been the Dingo library. Born as a machine-learning adaptation of bilby (up to an understandable love of Australian fauna), it is based on publications such as Green, Simpson, et al. 2020.
Since then, they have demonstrated the validity of these techniques by matching Markov-Montecarlo samples for GWTC-1 (Dax et al. 2021[b]), giving a complete estimation for GW150914 (Green and Gair 2021) or more recently applying their models to population studies (Leyde et al. 2024). They have also developed a more sophisticated algorithm for implementing coalescence time (or really any parameter with some special symmetry) in Dax et al. 2021(a). Since that level of complexity was far beyond the scope of a master's thesis I opted to ignore said parameter and hope to, at most, present a 14 parameter model.

Their main difference to this work is the data format used. Dingo opts to infer based on timeseries input, a more conventional approach akin to previous software such as bilby. Representing data as a vector allows dataset compression through a Singular Value Decomposition of the whole set. To properly capture the detectable portion of a signal's dynamic range (up to the hundreds of Hz) a sampling rate of 512 or 1024 Hz is recommended.

On the contrary, image-based estimation relies on an extra step of transformation, so the strain may be sampled at the finest rate with weight-saving compromises taken afterwards. A similar approach could have been taken regarding compression but for each individual image, though convincing results have been achieved with resolutions as low as 48 by 72 pixels(see page 53). Therefore, in an image-based scheme each event is also fairly lightweight in terms of diskspace without the need of compression algorithms such as SVD, which facilitates large datasets and batch-sizes while avoiding complex data-loading processes.
Besides, the chirp signature of the signal is easily recognizable, as well as noise patterns that could potentially be run through the Gravity Spy glitch-detection pipeline (Soni et al. 2021) beforehand. Computer vision, and in particular improvements in image recognition and characterization, is one of the main drivers of funding in the machine learning community , so it is expected that tools and embedding architectures will greatly improve in the coming years.

During the development of this project we have only had 3 detectors operating at peak capacity (the two LIGO detectors and Virgo), but this may soon change as KAGRA improves its detection range and LIGO India starts operations. The current embedding structure was meant to perform classification tasks on RGB images, so channel number is therefore hardcoded to 3. A more flexible alternative would have to be implemented for future observation networks.

3 A possible solution: The Deep Tempest library

Now that we have established the state of the art we should focus our attention on the main objective of this work: The Deep Tempest library (dtempest namespace).
Being an acronym for **Deep** **T**ransform-**E**xchangeable **M**odels for **P**osterior **Est**imation, it aims to provide a framework for developing highly customizable models for NPE based on normalizing flows.

Following on the footsteps of most software suites in the field, this library aims to be usable for non-gw oriented purposes, as most of what I have developed can be applied to any image-based NPE problem. In fact, I have worked hard to ensure that the core functionalities can work as its own library even on operating systems not supported by LALSuite This implies one could theoretically expand this software to study all different kinds of estimation problems.

3.1 The envisioned pipeline

This software is meant to be as self-contained as possible.

- It handles dataset generation and/or adaptation (more on that on the following Section), as well as its study.
- It allows problem specific parameter conversion. In the case of gravitational waves this is achieved with a mix of in-house solutions and the PESummary `conversion` package of its `gw` module.
- The user can define the architectures of flow and embedding of the model as well as the various stages of training needed through dictionaries (which would be easily extendable to json/yaml file systems).
- Once the model has been trained (or at any time, really), it can be saved and reloaded to be stored and used at a latter date.
- Estimations are handled by the `Estimator` object, which then creates a `SampleDict` or `SampleSet` that contains the results, which can then be plotted or studied further.

3.1.1 Generating our dataset

Now that machine learning is more in vogue than ever, it is becoming increasingly clear where the power of these algorithms lies: They are at most as good as their training data, usually worse. Large language models, for instance, have the entire public internet at their disposal (and oftentimes copyrighted data too), but this field is apparently not so lucky. As of writing there are only 90 confirmed detections of gravitational waves, orders of magnitude less than needed to train a mildly accurate model. Is there anything that can be done, then?

Well, if one has no data, one ought to invent data. Thankfully, as we saw in page 8, we have various models to create waveforms based on a given set of parameters. It is for that reason that a flexible yet robust data generation pipeline is implemented first of all. And it was, indeed, the first thing I worked on in this project, while reading up on machine learning (as at this point of development I had never tinkered with the technology myself).

From a users point of view all that is needed is to import the `Injector` object and initialize it, then save it:

```
from pathlib import Path
from dtempest.gw.generation.parallel import Injector

seed = 0
files_path = Path('path_to_file')
zero_pad = 3
inj = Injector(2000, seed=seed, config={'log_file': files_path / 'log.txt',
                                        'seed_zero_pad': zero_pad})
inj.save(files_path / f'Raw_Dataset_{seed:0{zero_pad}}.pt')
```

Listing 1: Dataset generation code. With default priors acceptance rate is a little over 50%, so this code produces about 1000 valid events, which constitutes each individual chunck of the full dataset.

When created, the `Injector` samples the parameters from the given prior and then generates each injection. In 1 none are given, so it takes the default ones (table 7), which are the ones used throughout the work.

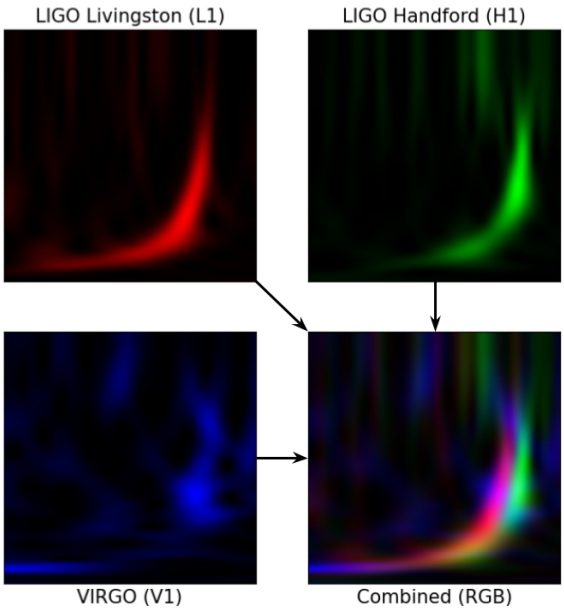


Figure 7: Depiction of each separate channel as well as the combined image for injection 1 of the validation set.

Generation is done in two steps: First one has to properly generate the waveform with the intrinsic parameters[7] as well as some extrinsic ones (distance, inclination and phase). Once the basic signal has been created we project it onto the desired detectors (LIGO and Virgo in most cases) and then keep a non-whitened copy (see whitening for more details) for matched filtering and a whitened one for creating the image through a Q transform.
The current network of three detectors is perfect for an RGB image, as shown in figure 7.

Finally, injection data is stored along with the parameters and some extra info such as the SNR on each detector, the Network Antenna Pattern, or NAP (more about it on page 30 and Section B.1), and the ids of the injection. Since there is a minimum SNR not all injections are kept, which tends to reject injections that have a high luminosity distance. That is why we have two ids: the actual id tells us the injection number among the accepted and the generation id (gen_id) informs us about which number it occupied among the entire population, useful for debugging and studying dataset generation.

Just as relevant as generation are data exploration and characterization utilities, as we will explore on the beginning of Chapter 4. The main tools at our disposal are image plots and parameter histograms, for which an example is provided in listing 2. In fact, this very code produced the histograms of figure 11.

```
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt

from dtempest.core.common_utils import load_rawsets, seeds2names
from dtempest.gw.conversion import plot_images, plot_hists
```

[7] That is, those that are related to the CBC itself: masses and spins

```
7
8     files_dir = Path('/to/files')
9     rawdat_dir = files_dir / 'Raw Datasets'
10    figsize = (12, 8)
11    params = np.transpose(np.array([['chirp_mass', 'mass_ratio'],
12                                    ['luminosity_distance', 'theta_jn'],
13                                    ['ra', 'dec']]))
14
15    seeds = range(50)
16    dataset = load_rawsets(rawdat_dir, seeds2names(seeds))
17    dataset.change_parameter_name('d_L', to='luminosity_distance')
18
19    indices = np.arange(12).reshape((3, 4))
20    fig_images = plot_images(dataset, indices, figsize=figsize)
21    fig_images.savefig('First_12_injections.png')
22
23    fig_hist = plot_hists(dataset, params, figsize=figsize, bins=50, title='')
24    fig_hist.suptitle('Parameter distributions on datasets 0 through 49')
25    plt.tight_layout()
26    fig_hist.savefig('Distribution_0_through_49.png')
```

Listing 2: Dataset exploration code.

This pipeline, as useful as it has been to me, has a few glaring setbacks:

- It is slow: Although through some multiprocessing magic I was able to get about five minutes per one thousand simulations, changes to PyCBC slowed everything down by a factor of 3.
- The whitening process doesn't fully convince me: Having to whiten the data and then do a blank pass on a copy seems to be overengineering the problem and was likely the cause of the slowdown.
- It isn't as flexible as it could be. Separating waveform creation from image creation would allow multiple basis for constructing the spectrogram (Wavelets, for example).
- Not getting a defined number of injections. With my settings acceptance is a bit over 50%, but a method that recalculates distance to fit SNR range would be much more efficient.

For these reasons I have since overhauled this Section to be bilby-based, which allows for more realistic priors, while retaining the ability to model detector acceptance (see figure 11 and discussion on appendix B.2). With that being said, this generator has produced the data behind this work's main results, so we shall move on and start training.

3.1.2 Training our models

Before training can begin we need a model to train. And to decide on a model we would need to decide on which parameters to perform estimations. For illustrative purpose lets settle for seven parameters; enough to show the functionality without the big plots of full estimations. This parameters will be the same as those used by models on Section 4.2.

First we need a parameter list of the aforementioned, and then a configuration dictionary for both the embedding net and flow.

The net is fairly straight forward: You specify whether it is a PyTorch net, which means it will use its `ResNet` class to build it. Then you lay out its structure (table 1 of He et al. 2015), number of output features and, if needed, type of block (`BasicBlock` by default, `BottleNeck` for deeper nets). More information can be found in PyTorch's resnet page and in its source code.

As for the flow configuration, here are the main parameters:

- **Scales** are definitely the feature whose introduction improved results the most. In spite of the various normalization layers of the flow, training on parameters with smaller magnitudes improves training stability and ease. The model is completely oblivious to the actual extent of the distributions. The data is reduced ($\theta / s_\theta \forall \theta \in \Theta$) before creating the dataloader structure and then the samples are appropriately expanded before returning them to us.
- The **input dimension** and **context dimension** should never be set manually (speaking from experience), as they are determined by your parameter space and by your embedding net: `input_dim` should always be the number of parameters you have and `context_dim` the number of features your net returns.
- With the **number of flow steps**, one is able to control how many repetitions of everything that will follow the model has. The more blocks, the more complex distributions you will achieve, but the model grows and deepens, so a careful balance has to be struck, as with most parameters.
- Now to the differentiating aspect of this library: the "exchangeable transforms". The architecture of an individual block consists of two pairs of arguments:
  - A **middle transform** which, despite its name, executes at the beginning of each block (its sandwiched in the middle of the base transforms, hence the name). It is supposed to be a simple transform, like a random permutation. It has to have the following signature: `func(param_dim: int, **kwargs)`.
  - And the hearth of the model: the **base transform**. This is meant to be a neural-network controlled transform, oftentimes based on neural spline flows (Durkan et al. 2019). We went over the particular transforms used in this work in page 12.
- At the end of the model we have the **final transform**. It is supposed to be akin to the middle transform, but it is allowed to be defined differently.

```
import numpy as np
from pathlib import Path

from dtempest.gw import CBCEstimator
import dtempest.core.flow_utils as trans  # Import as a transform drawer

params_list = [  # Order of parameters will be preserved
'chirp_mass',
'mass_ratio',
'chi_eff',
'luminosity_distance',
'theta_jn',
'ra',
'dec'
]

pre_process = None  # Some models had preprocessing applied to the images

net_config = {
'pytorch_net': True,
'depths': [2, 2, 2, 2],  # Resnet18 Structure
'output_features': 128
}

```

```
    flow_config = {
    'scales': {  # The model will train on data labeled: label/scale['param'] and then
    samples taken will be rescaled
        'chirp_mass': 80,
        'luminosity_distance': 2000,
        'theta_jn': 2*np.pi,
        'ra': 2*np.pi,
        'dec': np.pi},
    'input_dim': len(params_list),
    'context_dim': net_config['output_features'],
    'num_flow_steps': 5,

    'base_transform': trans.d_rq_coupling_and_affine,
    'base_transform_kwargs': {
        'hidden_dim': 64,
        'num_transform_blocks': 5,
        'use_batch_norm': True
    },
    'middle_transform': trans.random_perm_and_lulinear,
    'middle_transform_kwargs': {},
    'final_transform': trans.random_perm_and_lulinear,
    'final_transform_kwargs': {}
    }

    '''Flow creation'''
    flow = CBCEstimator(params_list, flow_config, net_config,
                        workdir='directory/to/save/stuff', mode='net+flow',
                        name='Model Name', preprocess=pre_process)
    flow.save_to_file(Path('not/necessarily/workdir') / f'{flow.name}.pt')
```

Listing 3: Model creation code.

Then we load up the data (both train and validation) and we convert it based on the parameter list of the model and start training. I will go in more detail about conversion algorithms at the end of Section 3.2, but for external use one has to keep in mind that it is an application dependent Section. Even if the core module implements the fast (default, faster for gravitational waves) conversion algorithm, it needs a non default `Jargon` dictionary to convert properly. Conversion can be done right after model creation, only separated here to better adapt to written format.

```
    from pathlib import Path

    from dtempest.core.common_utils import load_rawsets, seeds2names
    from dtempest.gw import CBCEstimator
    from dtempest.gw.conversion import convert_dataset

    pre_process = None
    n_epochs = 10
    train_config = {
        'num_epochs': n_epochs,
        'checkpoint_every_x_epochs': None,  # Not yet implemented
        'batch_size': 64,
        'optim_type': 'Adam',  # Stochastic Gradient Descent ('SGD') also an option
        'learning_rate': 0.001,
        'grad_clip': None,
        'sched_kwargs': {
            'type': 'cosine',  # Takes learning rate to 0 over T_max epochs
            'T_max': n_epochs,
            'verbose': True
            }
    }

    flow = CBCEstimator.load_from_file(Path('not/necessarily/workdir') / f'Model Name.pt')
    flow.rename(f'Trained model')

```

```
    '''We load training data onto memory'''
    seeds = range(60)  # Datasets from 000 to 059
    dataset = load_rawsets('dataset/directory', seeds2names(seeds))
    dataset.change_parameter_name('d_L', to='luminosity_distance')
    trainset = convert_dataset(dataset, flow.params_list)


    '''We load validation data onto memory'''
    valiseeds = 999  # Could be range of seeds, as in the dataset
    valiset = load_rawsets('validation/directory', seeds2names(valiseeds))
    valiset.change_parameter_name('d_L', to='luminosity_distance')
    vali_trainset = convert_dataset(valiset, flow.params_list)

    traindir = Path('train/directory')
    flow.train(dataset, traindir, train_config,valiset)
    flow.save_to_file(Path(traindir) / f'{flow.name}.pt')
```

Listing 4: Training routine code.

Training is, by far, the biggest hurdle in any machine learning journey, and I will go over my adventures in Section 3.2.

3.1.3 Estimations with our models

Finally comes the most interesting part for a scientist working on a machine learning project: Getting to play with your models. The sampling process is fairly straight forward. We load up sampling data the same way we loaded the training data in the previous listing. Although we can sample directly from the model it is better to place them in a suitable container, and that is where `SampleSet` and `SampleDict` (or its subclasses, such as `CBCSampleSet`) come in.
They hold our samples and implement most of the analysis functionality, such as statistical tests and plotting routines.

```
    from pathlib import Path

    from dtempest.core.common_utils import load_rawsets, seeds2names

    from dtempest.gw import CBCEstimator
    from dtempest.gw.conversion import convert_dataset


    traindir = Path('where/you/left/it')
    flow = CBCEstimator.load_from_file(traindir / f'Trained Model.pt')

    seed = 999
    event = f'{seed}.00001'

    dataset = load_rawsets('data/directory', seeds2names(seed))
    dataset.change_parameter_name('d_L', to='luminosity_distance')
    trainset = convert_dataset(dataset, flow.param_list, name=f'Dataset {seed}')

    sset = flow.sample_set(3000, trainset, name=f'flow {flow.name}')

    full = sset.full_test()  # Absolute accuracy + precision statistical test
    full_rel = sset.full_test(relative=True)  # Relative accuracy + precision test

    image = trainset['images'][event]
    label = trainset['labels'][event]
    sdict = flow.sample_dict(10000, context=image, reference=label)

```

```
    sset.save() #Not implemented
    sdict.save() #Not implemented

    # We perform a per-parameter mean and print the results as latex code
    print(full.pp_mean().to_latex(float_format="%.3f"))
    print(full_rel.pp_mean().to_latex(float_format="%.3f"))
```

Listing 5: Sampling example code.

This code produces tables like the ones shown: Absolute accuracy and precision give an overarching performance of a model over a given dataset, though comparing parameters with different units can be complicated.

parameters	median	truth	accuracy (MSE)	precision_left (1.0σ)	precision_right (1.0σ)	units
chirp_mass	47.058	46.853	6.099	7.209	7.225	$M_\odot$
mass_ratio	0.596	0.613	0.167	0.195	0.228	ø
chi_eff	0.023	0.011	0.152	0.189	0.185	ø
luminosity_distance	1445.094	1502.022	508.717	578.854	698.108	Mpc
theta_jn	1.580	1.546	0.630	0.790	0.806	rad
ra	3.078	3.176	1.135	1.375	1.285	rad
dec	0.049	0.017	0.423	0.501	0.509	rad

Table 1: Example of absolute summary table of a model over a dataset.

Relative magnitudes make comparisons much easier, and yet, as demonstrated with the χ_{eff} and α (ra) parameters, can give you a misleading idea. For the former, the almost 1000% uncertainty is related to the fact that effective spin tends to hover around very small values (0.011 being the average over the dataset), which causes the relative values to blow up. A similar thing occurs with δ (dec). As for the latter, the relative table would suggest massive overfitting, yet none of that is seen in the absolute table. This again has to do with small values. In this case a spurious event for which the true α is close to 0 and the predicted value must be quite large , heavily biasing the average.

parameters	median	truth	accuracy (MSE)	precision_left (1.0σ)	precision_right (1.0σ)	units
chirp_mass	47.058	46.853	16.251	16.851	17.964	%
mass_ratio	0.596	0.613	40.187	35.178	44.625	%
chi_eff	0.023	0.011	214.077	946.265	924.850	%
luminosity_distance	1445.094	1502.022	43.205	38.146	48.774	%
theta_jn	1.580	1.546	81.520	50.019	52.472	%
ra	3.078	3.176	542.423	46.704	64.900	%
dec	0.049	0.017	242.582	571.724	587.219	%

Table 2: Example of relative summary table of a model over a dataset.

Mean tables allows us to evaluate at a glance our model's performance, but the best metric is to actually show the samples. There are a huge a variety of ways to visualize high dimensional histograms, but the most flexible is by far the corner plot. Given the time limitation of this work I have focused my efforts on producing good-looking corners such as figure 8 by partially overriding the plot functionality from PESummary.

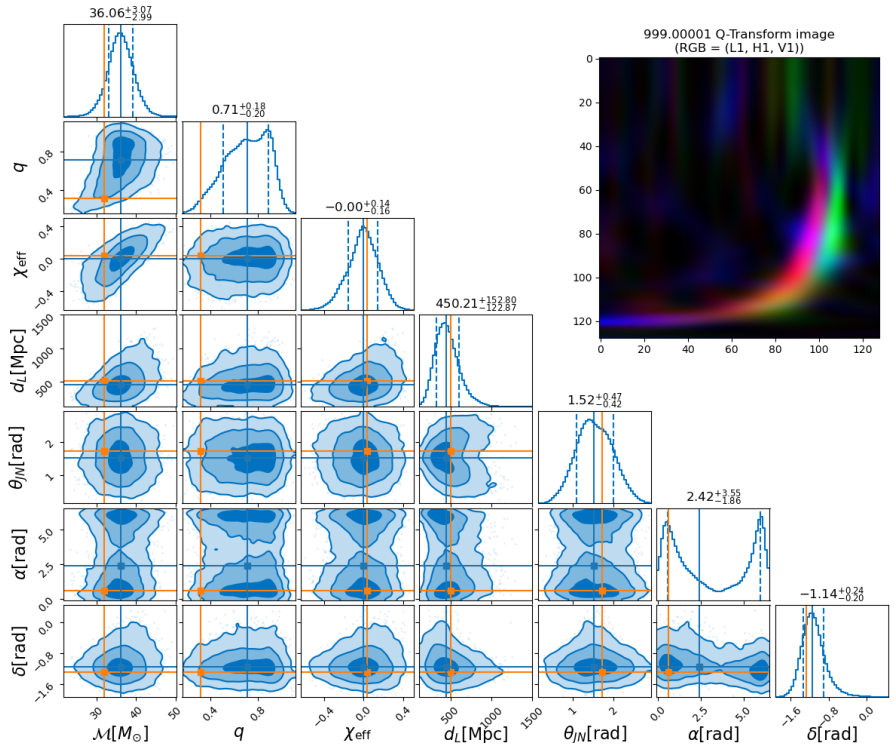


Figure 8: Example of smoothed corner plot of 10000 samples. Blue solid lines represent the model's median, with the dotted lines demarkating a 1σ deviation, which follow the darked shaded regions of the 2D histograms. Lighter regions denote 2 and 3σ regions (86.5 and 98.9% confidence intervals, all of them being top-level configurable). Orange lines represent injection parameters. The image on the top right is the raw image that produced the samples (in this case also the actual one, as there was no preprocessing).

3.2 Deep Tempest's development journey

So far this Section we have focused our attention on the more "traditionally interesting" aspect of my work; the talking points that one would go over when writing a publication or software presentation. Yet, since this is a Master's Thesis and a substantial part of the focus is to demonstrate my validity as a researcher I thought it would be not just appropriated but necessary to briefly detail my journey and some of the problems I faced to get this library to where it stands. If you indulge me for a few pages I think a more personal and informal tone may better communicate the ideas contained in this Section.

Although not quite in chronological order, I will start with the so-called "epiphanies"; the little adjustments that you wish you had thought of months prior and are often caused by wrong assumptions. As I discussed with Joaquín at some point during the year, they are what tends to be left out of scientific papers, making it seem as though everything worked first try or with little effort, when in reality nothing could be further from the truth. As I have said, they are not relevant for the proper results of my work, but are akin to what I would discuss at a lab report, and so I consider them relevant.

The first of these was in part due to being new to the machine learning scene, as at this point last year I had no knowledge nor experience on the subject. Common sense would indicate that the least components one ought to train, the easier the process. And in a post GPT-induced social craze about Artificial "Intelligence" finding robust pre-trained nets was easier than ever. For that reason my early attempts relied on a pre-trained resnet18 architecture (He et al. 2015) trained on the ImageNet dataset (Russakovsky et al. 2015) that could be obtained directly from PyTorch. ImageNet is highly varied (1000 different classes) and great success has been found modifying the final layer to retrofit these nets in fields as distant as software safety (Rezende et al. 2017) or art classification (Gao et al. 2023). Surely having one of these as a feature extractor to feed into the flow would be accurate from the get go and improve convergence by training only about half (depending of course on the configuration of each subassembly) of the model's parameters, right?

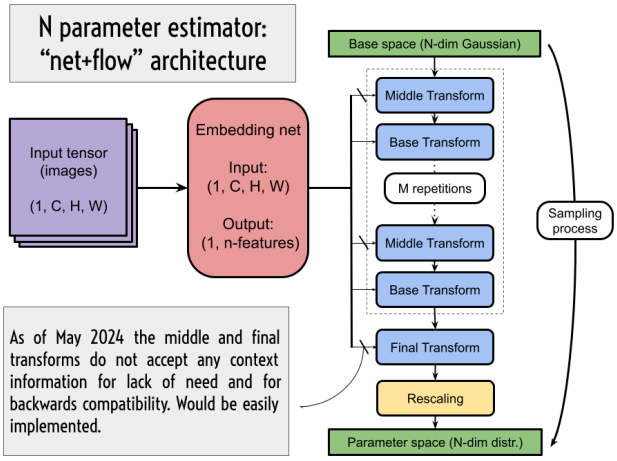


Figure 9: Diagram of the basic architecture behind the dtempest models. Unlike traditional networks, you don't transform the input into the output; You use the input to transform (white) noise into the output.

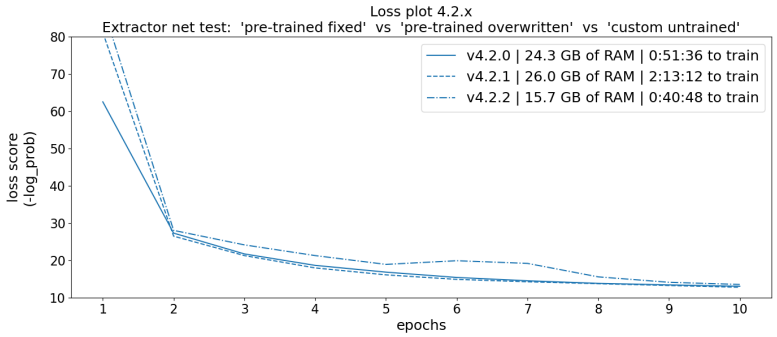


Figure 10: Comparison between various feature-extracting philosophies on a small 7 parameter model. Pre-trained networks behave fairly similarly whether overwritten or not, which is still desireble given convergence was expected to worsen. On deeper models a fixed stage followed by an overwriting stage was found to improve loss targets. As expected, the untrained net has a more chaotic curve, but the massive saving on RAM allows for bigger datasets and therefore better performance on a comparable amount of time.

Well, as it turns out, not really. When I tested whether overriding the parameters of the extractor would slow convergence down, it did the opposite. Now the system had many more degrees of freedom to optimize, which resulted in substantial improvements. After a while it became increasingly clear that the initial weights were not really giving me any advantage, so I opted to remove them. This had another benefit I wasn't aware of. The object in which PyTorch stores the pre-trained parameters is also full of other information about them and the training conditions. It also contains the necessary transforms that pre-process the images, which, among other things, set the size as `224 X 224` pixels, almost twice the size (so four times the pixel count) of my original images. This entails that datasets will consume unnecessarily large amounts of memory. The initial tests presented in figure 10 were taking around 25 Gigabytes of RAM, as opposed to the 15 a comparable model with a resnet18 of my own making would be occupying during an identical training session. Removing this excess weight allowed much bigger datasets, which combined with deeper flows thanks to better convergence improved the models significantly.

In spite of this improvement, I was struggling with deeper models, as they started to overfit too early to be useful. I was not considering training with more than 8 parameters, as initial loss diverged and gradients exploded. This is a common problem and has a fairly easy solution: normalize your parameters. It is so common, in fact, that many architectures implement off-the-self normalization layers. My flows employ them, as evidenced by the `'use_batch_norm':True` argument on listing 3. For this reason I was convinced that I was going to need something fancier. At a meeting Osvaldo suggested that I rescale data to bring large numbered parameters (mostly chirp mass and luminosity distance) closer to a 0 to 1 interval. I wanted to ensure I could be flexible enough to test different scales affecting different parameters, so I implemented it as the `'scales'` sub-dictionary to be passed at model creation time. As it is probably evident by its inclusion in this Section, it worked wonderfully. Flows trained faster and more reliably. The results obtained for train data where now achievable by validation data as well, and jumping to high parameter models didn't seem to impact convergence. The jump in quality was such that I was finally ready to test my models on actual recorded events and against official estimations (GWTC publicly available data).

parameters	median	truth	accuracy (MSE)	precision_left (1.0σ)	precision_right (1.0σ)	units
chirp_mass	47.277	46.853	7.034	2.637	2.609	$M_\odot$
mass_ratio	0.609	0.613	0.187	0.082	0.086	ø
chi_eff	0.037	0.011	0.178	0.060	0.059	ø
luminosity_distance	1553.703	1502.022	574.074	255.214	253.163	Mpc
theta_jn	1.523	1.546	0.693	0.472	0.551	rad
ra	2.965	3.176	1.287	0.542	0.551	rad
dec	0.044	0.017	0.461	0.182	0.186	rad

Table 3: Validation statistics of model 2.11.0c, trained on 60 thousand images for 30 epochs (around 9 hours). Average acurracy fluctuates between 1.4 and 3 sigmas among parameters, which is a clear sign of overfittig. Despite this, plots show interesting structures (see bottom panel of figure 18).

parameters	median	truth	accuracy (MSE)	precision_left (1.0σ)	precision_right (1.0σ)	units
chirp_mass	46.725	46.853	6.025	6.805	6.766	$M_\odot$
mass_ratio	0.590	0.613	0.168	0.193	0.215	ø
chi_eff	0.017	0.011	0.151	0.185	0.181	ø
luminosity_distance	1427.974	1502.022	505.952	545.038	639.602	Mpc
theta_jn	1.577	1.546	0.627	0.806	0.759	rad
ra	3.078	3.176	1.135	1.357	1.206	rad
dec	0.007	0.017	0.412	0.493	0.535	rad

Table 4: Statistics of model 2.12.0c, trained on 60 thousand images for 10 epochs (around 3 hours).

Outside of struggles that directly improved results there were a myriad of other technical issues since, after all, this is a software project.
To illustrate this point I will give an example that gave me headaches at various points of development: the `TrainSet` class (source code on listing 6). I am particularly proud of this class as when asked about extending `ndarray`-like objects the online forum communities tend to give a patronizing "don't bother".

Its functionality is fairly simple to describe, it is an extension of the pandas `DataFrame` object. It is supposed to hold the converted datasets, so rows of image-label pairs to be accessed through a key-value system. The sampling code has an example of this in action (listing 5: lines 24, 25). A `DataFrame` is a great choice as it comes with a huge variety of utilities, such as data localization support, insertion and deletion of both rows and columns, rng functions, etc.

However, I needed some extra attributes and custom implementations of saving, loading and concatenation methods. As I alluded to before, subclassing these objects is extremely cumbersome, so a wrapper was a much more appealing option. For those unfamiliar with object-oriented programing -OOP-, a wrapper contains an object in such a way that the high-level user interacts with the wrapper, but the actual task is performed within the wrapped object. A middleman, in a way. For instance, to operate on two python arrays (numpy `ndarray` objects) we don't need to know any C++, but the actual operation is done in C++ though a cross-language wrapper for performance reasons.

The standard way of implementing object wrappers is to catch the interaction and pass it to the inner object (which I will from now refer to as `self._df`). That is what `__getitem__` and `__setitem__` functions do on 6 regarding the accessing / setting of data, while handling the objects methods is where the previous "don't bother" starts to be a reasonable option.
Passing the method is perfectly fine for operations that return specific data or even modify `self._df`, but it becomes a problem when `self._df.method()` returns a `DataFrame` that should replace `self._df`.

To keep things concrete I will give an example: Lets imagine that we don't know the exact length of our dataset (only that is over 10000 instances) and we want to train over exactly 10000 images. `DataFrame` has the perfect solution: the `sample` method returns a `DataFrame` with n randomly selected entries of the parent `DataFrame`. The randomness can be seeded with a random number generator and could come in handy as we can remix it between stages without losing reproducibility. if we have our `trainset` of type `TrainSet` and we do "`trainset = trainset.sample(n=10000, random_state=rng)`" we are left with a `trainset` that is now of type `DataFrame`. And if we keep pretending it is a `TrainSet` we will eventually break something.

So to fix this we need to not only catch the method in `__getattr__`[8], but also identify which of them return another class. Doing this in python is fairly unorthodox, as it is a dynamically typed language and therefore return types need not be known before runtime. Thankfully most big libraries are sensible enough to include them, as it facilitates automatic documentation generation (something I would actually like to tinker with eventually).
Those that return a new `DataFrame`, `new_df`, need to return "`TrainSet(data=new_df, name='old_name')`" to the user, which is a problem because we are still in the trainset's `__getattr__` function. As I explained in the OOP glossary entry, this function gets executed on any "`trainset.attribute`" expression. It returns the requested attribute, but if it is a function it has not been given any arguments yet, so in our case `sample` has not been called, and therefore we don't have `new_df`. How are we supposed to pass it to `TrainSet` if it hasn't been created?

[8] python objects treat methods and attributes the same way, thus they are both handled in `__getattr__`

```python
from pathlib import Path
import pandas as pd
import torch

from dtempest.core.common_utils import RawSet

class TrainSet:
    """
    Wrapper class for pandas DataFrame. Allows me to name it.
    When applying any funtion that returns knows what to return
     (Array-like objects are not easily extended or subclassed)
    """
    def __init__(self, name: str = None, *args, **kwargs):
        if 'data' in kwargs.keys() and isinstance(kwargs['data'], pd.DataFrame):
            self._df = kwargs['data']  # Initialize from a DataFrame
        else:
            # Initialize from a pandas-recognized data structure
            self._df = pd.DataFrame(*args, **kwargs)
        if name is None:  # If name is not given, take it from the data...
            for arg in args:
                if isinstance(arg, RawSet):
                    name = arg.name
                    break
            else:  # ... Or have a generic name
                name = type(self).__name__
        self.name = name

    def __getattr__(self, attr):
        if attr in self.__dict__:
            # If it is in the TrainSet __dict__, just pass it along
            # Otherwise it better be a DataFrame attribute
            return getattr(self, attr)
        elif hasattr(self._df, attr):
            # First let's deal with some special, often used, cases
            if attr == 'T':
                return self.transpose()
            if attr == 'sample':
                # Transform the internal dataframe and return a new TrainSet
                return Overarcher(getattr(self._df, attr), self)
            try:  # Classify attributes depending on their return types
                if hasattr(type(self._df), attr) and hasattr(getattr(self._df, attr),  '__annotations__'):
                    returns = getattr(self._df, attr).__annotations__['return'].split( ' | ')
                    if 'DataFrame' in returns:
                        return Overarcher(getattr(self._df, attr), self)
                    elif 'Series' in returns:
                        # Not using it yet, but interesting for other classes
                        return getattr(self._df, attr)
                    elif 'str' in returns:
                        return getattr(self._df, attr)  # can catch formatting calls
                    else:
                        return getattr(self._df, attr)
                else:
                    return getattr(self._df, attr)
            except KeyError:
                return getattr(self._df, attr)
        else:
            raise AttributeError(f"{type(self)} has no attribute '{attr}'")

```

```
59          def __getitem__(self, item):
60              if isinstance(self._df[item], pd.DataFrame):
61                  return type(self)(data=self._df[item], name=self.name)
62              return self._df[item]
63
64          def __setitem__(self, item, data):
65              self._df[item] = data
66
67          @classmethod
68          def concat(cls, trainsets: list, name: str = None, *args, **kwargs):
69              if name is None:
70                  name = trainsets[0].name
71              dfs = [tset._df for tset in trainsets]
72              return TrainSet(data=pd.concat(dfs, *args, **kwargs), name=name)
73
74          @classmethod
75          def load(cls, path: Path | list[Path], name: str = None, verbose: bool = True):
76              # Trainset loads and saves its internal DataFrame, not the TrainSet itself
77              if type(path) is list:
78                  return TrainSet.concat([TrainSet.load(_path) for _path in path], name=name)
79              try:
80                  _name, df = torch.load(path)
81              except ValueError:
82                  # Then is a lightweight save
83                  df = pd.read_pickle(path)
84                  _name = path.parts[-1][:-3]
85
86              if name is None:
87                  name = _name
88
89              if verbose:
90                  print(f'Loaded TrainSet {_name}')
91              return TrainSet(data=df, name=name)
92
93          def save(self, path: Path, lightweight: bool = False):
94              # Trainset loads and saves its internal DataFrame, not the TrainSet itself
95              # lightweight saves the dataframe contents only. No name but more compact
96              # If it isn't given a pickle file it will name it after itself
97              if path.parts[-1][-3:] != '.pt':
98                  path = path / f'{self.name}.pt'
99              if lightweight:
100                 self._df.to_pickle(path)
101             else:
102                 torch.save((self.name, self._df), path)
103
104         def __len__(self):
105             return self._df.shape[0]
106
107         def __repr__(self):
108             return self._df.__repr__()
```

Listing 6: `TrainSet` class source code.

This need to get the result before calling the function is the reason I suspect most people online give up and move on, but I came up with a trick. Instead of returning "`TrainSet(data=new_df, name='old_name')`", why not return "`Overarcher(getattr(self._df, attr), self)`", where `Overarcher` is a small custom class whose code can be found in listing 7. It is basically a method-wrapper that, when called by an unsuspecting user, does not return the results of the method, as the user might expect, but an object of the same type as `self` (in this case a `TrainSet`) created from said results, which achieves our purposes.

If you paid close attention you may have noticed that I wrote "the `sample` method returns a `DataFrame`...", and not "the `DataFrame.sample` method returns...". Though it could have been for simplicity on expressing complex and abstract ideas, it was a fully conscious decision and the reason why sample is actually handled separately, as it is actually not an attribute of the `DataFrame` class. Luckily it only needs to be caught with a dedicated "`if`" statement and served to an `Overarcher` object (lines 37–39 of listing 6).

```
    class Overarcher:
        def __init__(self, method, wrapper_object: TrainSet):
            self.method = method
            self.wrapper = wrapper_object

        def __call__(self, *args, **kwargs):
            return type(self.wrapper)(data=self.method(*args, **kwargs),
                                      name=self.wrapper.name)
```

Listing 7: `Overarcher` class source code.

There were a few similar struggles throughout development. Related to dataframes was getting the table formatting to suit my needs. Similarly with plotting routines (mainly that of corner plots). Another one that comes to mind and that I think is worth reflecting on is the "pesum_deps" package. Although, as mentioned earlier in this Chapter, gravitational wave software is separated in core and specific functionalities, that does not mean that they can live independently, let alone be installed separately. Since I was determined to make mine fully independent I needed to be careful with what dependencies I allowed for my `core` package. The only gw software that wasn't really avoidable was PESummary, so I had to copy the classes and functions I needed making sure there was no dependence on LALSuite or any other gw-oriented software. For my `gw` package I import PESummary directly, but gw objects that inherit from core objects (such as `CBCSampleDict` from `SampleDict`) will still depend on my copy of the package, something to keep in mind as PESummary receives updates.

This was not an issue for parameter conversion, tough, as I could depend directly on PESummary's conversion algorithm whenever mine failed. Mine converts through a tree of dependencies based on the fact that I know which parameters I chose in generation (table 7), whereas the aforementioned generates all possible parameters given the samples, which is general and robust but slow and overkill in most cases. The only use I currently make of it is when dealing with catalogue data, as they obviously were not fitted to my particular parameter choice.

As hinted by the last paragraph, there will be comparisons between my increasingly powerful models and official MCMC samples conditioned on real world data taken at both LIGO and Virgo.

4 Results so far

We have reached the crux of this work. I have spent this past few months working on what is essentially a simple conceptual question: Can a machine tell us the characteristics of a CBC based on images of its signal?

First of all let us take a look at the distribution of parameters over our dataset. This information will be crucial to understand many of the later results, as it gives us a concise picture of what our models are seeing on training. The first 50 dataset chunks contain what most models trained on, and it is big enough to avoid statistical flukes.

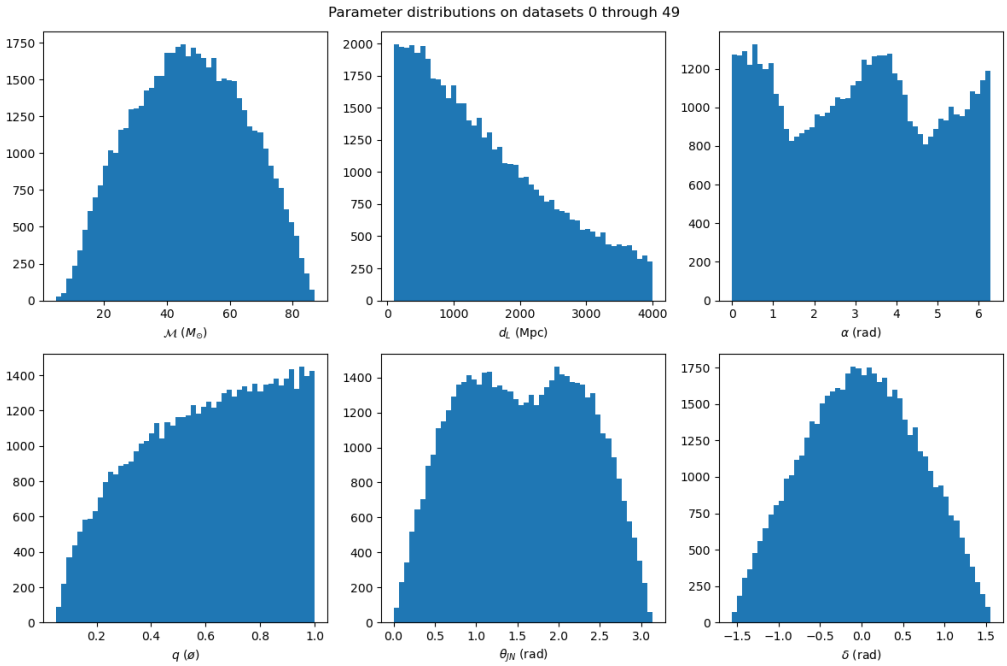


Figure 11: Distribution of parameters over the first 50 datasets. These histograms are a combination of the priors on the second column table 7 and minimum SNR conditions.

From figure 11 we can gather that low mass ratio and large distances are disfavored. Distance limitations should become an issue especially in catalogues 2 and 3 (GWTC-2.1 and GWTC-3), as sensitivity and therefore detection range increases. Chirp mass should be most accurate for events falling on between 35 and 60 solar masses, and models that estimate inclination should be able to predict a certain degree of bimodality. Declination preserves its prior after discarding, but right ascension bins periodically lose up to a third of their events. This is related to the NAP pattern of the L1+H1+V1 configuration (see a more in depth discussion on Appendix B.1).

All datasets, including those in figure 11, used IMRPhenomV2 (see discussion on page 8). No comparison between approximants is shown given the current accuracy, but works such as Álvares et al. 2021 indicate that results shouldn't particularly deviate. Still, it is something to check for production-ready models.

4.1 Low dimensionality models

As one would expect for a task as complex as this one, starting out with the simplest possible scenario is always the better option. For this reason I set myself the objective of producing a chirp mass estimator model and build from there. Because there was only one parameter there were no fancy corner plots or parameter interrelations to worry about. At the same time, there is so much information we lose by restricting ourselves to only one parameter that we cannot really guarantee that the model will learn. It will understand that a strong signal is probably massive, but it could also be very close, or inclined towards us, or in a location of the sky to which the detectors were particularly sensitive at that instance (distance-inclination degeneracy is pictured in figure 17).

Here I present two sets of results: First, the models from the latter months of 2023 (fig. 12), followed by a more modern one made with the same architecture as the ones discussed in the following Sections (fig. 13(b)). Looking back, these first attempts were actually quite on point considering the limitations of reducing a high dimensional space to just a single degree of freedom. Masses tended to be within one standard deviation of the GWTC median, but precision was their main handicap. Uncertainties where often so large that the 1 sigma region contained all GWTC samples, and my samples would cover the entire prior space even in the best of cases (see B. P. Abbott et al. 2019(c) and I. M. Romero-Shaw et al. 2020 for more information on all GWTC-1 events).

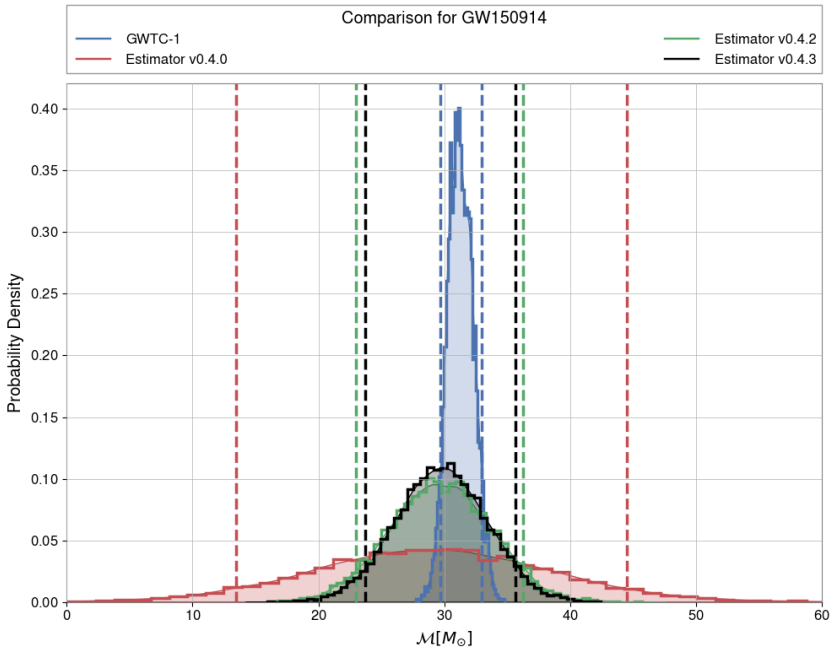
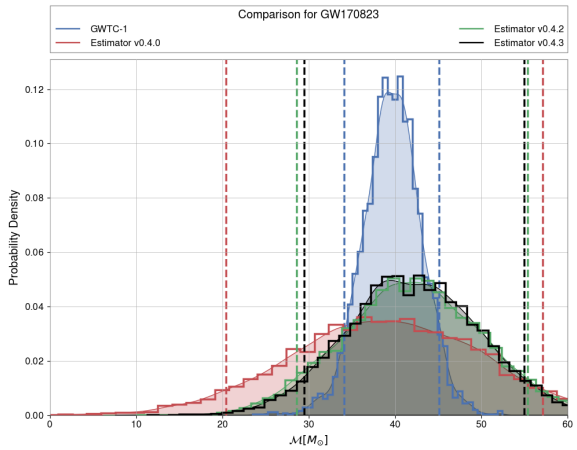


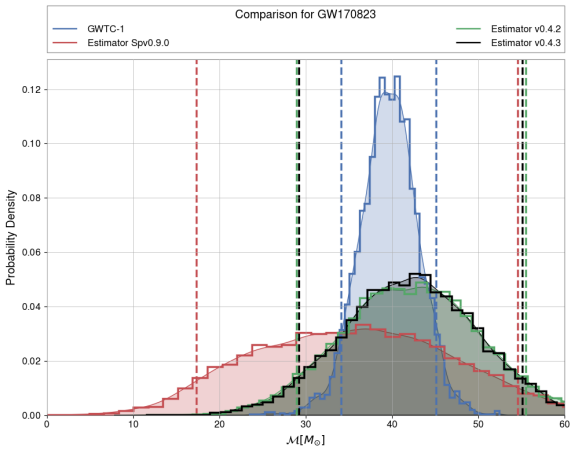
Figure 12: Comparison between various chirp mass estimations and the nominal distribution acompaning B. P. Abbott et al. 2019(c) for the first recorded event, GW150914. Although the models are far from perfect, it is interesting that relatively simple models can achive such resemblence.

Best case scenario for chirp mass estimators (and arguably for most models) has been GW170823, the last BBH from the first catalogue (B. P. Abbott et al. 2019[c]), as evidenced in figure 13. It is on the higher end of mass and distance for said catalogue, with average spin properties.

Before moving on, it is curious to see that trying out the architectures that gave the results we will see on Sections 4.2 and 4.3 with much more data actually resulted in mediocre models, shown in figure 13(b). Table 5 shows v0.9.0 is the better model, yet the overfitted models appear to be making a better estimation.



(a) Comparison of various early models on GW170823. The limits have been set for more clarity due to important diferences in precision between estimations.



(b) Here the first model (red) has been substituted by a more modern architecture. The limits have been artificially cut off once again for clarity.

Figure 13

To be fair, v0.4.2 and v0.4.3 were trained for a total of 110 epochs each on a smaller dataset. Since no validation information was available for these early models I cannot say for sure to which extent they are overfitted, but they are clearly being less accurate than precise, and although their estimations seem to mostly agree with the catalogue, they ought to be taken with a grain of salt.

chirp_mass	median	truth	accuracy (MSE)	precision_left (1.0σ)	precision_right (1.0σ)	units
v0.4.0	40.953	46.853	13.533	14.716	14.701	$M_\odot$
v0.4.1	40.395	46.853	13.303	13.996	14.008	$M_\odot$
v0.4.2	46.505	46.853	12.011	10.929	10.913	$M_\odot$
v0.4.3	46.245	46.853	12.234	9.978	9.980	$M_\odot$
v0.9.0	46.433	46.853	12.080	13.871	13.958	$M_\odot$

Table 5: Validation statistics of the various chirp mass estimators. One thing that isn't shown and yet remains vital is sampling time. The more modern model produced its `SampleSet` in just under a minute and a half, almost twice as fast as the others. This is quite counterintuitive, as it is actually the most complex flow for an identical net structure (though the elders use pre-trained extractors).

After these tempting results I upgraded the degrees of freedom to four, as was the case in Álvares et al. 2021. The aim is to model the main phenomenology of a BBH in as few parameters as possible. Because of this, stellar coordinates were reduced to a single parameter: ***N**etwork **A**ntenna **P**ower*. It encapsulates the sensitivity of a given network of detectors (See appendix B).
Alongside NAP we also have chirp mass, effective spin and luminosity distance, all of them discussed in detail in Section 1.3. Although a neat idea for economizing parameter dimensionality, the lack of support of this custom parameter in gw-software meant I spend most of my time fighting with code errors, so I substituted NAP for right ascension and declination without much cost. I hadn't got the corner-plot utilities working at that point, but revisions showed that no meaningful learning occurred.

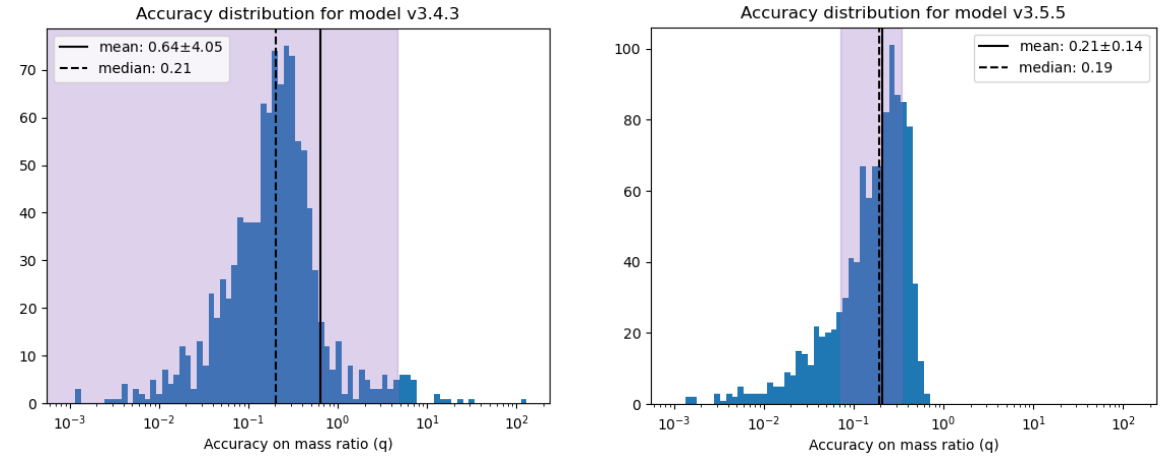
Since I could get a complete description of the mass by adding one more parameter, I included the mass ratio as well. The following results correspond to the final stages of the last two configurations I attempted. I did not feel the need to change the model configuration aside from accommodating the growing number of parameters, focusing mostly on training routines. I trained the first one, v3.4.3, for four stages of ten epochs each. I slowly increased the batch size from 96 to 256 and cut learning rate in half by the last two stages. This was motivated due to the proportionality found between batch size and initial loss. As we discussed in page 23, a large initial loss will difficult or outright impede training, and reducing batch size is a simple solution, though I eventually came up with better alternatives (page 34).
The other, v3.5.5, has had two extra stages and the batch size was kept constant at a size of 256. This was possible thanks to the large spread in initial loss values that occurs with this current configuration[9] , so it was down to the draw of the dice.
Training times for both hovered around 45 minutes for a little over 15000 images, with the last two stages of v3.5.5 taking a whole hour, probably due personal use of computer during training. Each epoch the 15 datasets were redrawn from a pool of 45.

v3.4.3	median	truth	accuracy (MSE)	precision_left (1.0σ)	precision_right (1.0σ)	units
chirp_mass	47.171	46.853	11.787	13.801	13.986	$M_\odot$
mass_ratio	0.699	0.613	0.649	0.250	0.233	ø
chi_eff	0.070	0.011	0.208	0.254	0.251	ø
luminosity_distance	1342.582	1502.022	645.798	627.534	816.265	Mpc
ra	3.016	3.176	1.580	1.737	1.747	rad
dec	-0.076	0.017	0.556	0.663	0.663	rad
v3.5.5	median	truth	accuracy (MSE)	precision_left (1.0σ)	precision_right (1.0σ)	units
chirp_mass	46.979	46.853	13.918	16.725	18.259	$M_\odot$
mass_ratio	0.578	0.613	0.208	0.245	0.264	ø
chi_eff	0.015	0.011	0.191	0.290	0.269	ø
luminosity_distance	1379.870	1502.022	699.671	614.409	1040.480	Mpc
ra	2.955	3.176	1.629	2.049	1.999	rad
dec	-0.049	0.017	0.557	0.655	0.713	rad

Table 6: Validation statistics for a couple of 6 parameter models.

Something interesting occurs with these two models, which is why I thought it was appropriate to discuss both. v3.4.3 overfits on only a single parameter (mass ratio), but it does so quite importantly. Or so we might think. In actuality, it could be due to spurious samples skewing the distribution, which we can test by plotting the mean squared errors of all samples of q on a histogram (figure 14).
As we suspected, a handful of outliers are contaminating the result of the first model (figure 14(a)), which suggest this model is useless, as nobody should trust a model that can miss not just the value, but the entire prior by a couple of orders of magnitude.
Model v3.5.5 is much better on that regard, always staying within the prior.

[9]This randomness is mostly due to the affine transmorm that forms the basis of these early models. Later we will se that not all transforms behave this way, for better and for worse.



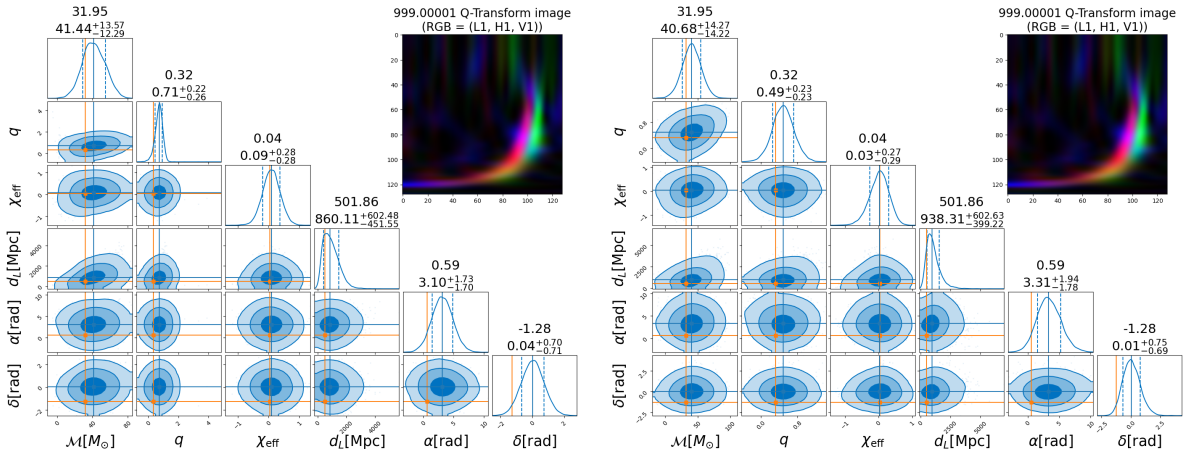
(a) Model v3.4.3. Since $q \in (0, 1]$ by definition, data whose MSE is bigger than one are clearly non-physical outlirers

(b) Model v3.5.5. Non-physical outliers have dissapeared with the two extra training stages.

Figure 14: Histogram depicting the distribution of accuracy (mean squared error, MSE) of two models on validation data (1064 injections). Shaded region depicts 1 sigma range.

We have confirmed our suspicions, but to actually see how they fare we need to show an estimation. The best way to illustrate a high dimensional histogram a corner plot, which will be most of the results going forward. I will presume a certain degree of familiarity with these tools, but an important note is that throughout this work the shaded regions of 2D histograms always represent the 1, 2 and 3 sigma confidence levels, which respectively contain a $1 - \exp(-a^2/2)$ fraction of the population, not to be confused with the 68% common in 1D[10].

As expected given the previous discussion, the mass ratio is acceptably estimated but the outliers drag the distribution away, even if they are not numerous enough to be inside confidence levels of the plot (figure 15(a)). On the newer model (see 15(b)), now without outliers in q, they appear in d_L. Apart from that, behaviour is still very much Gaussian, which indicates these models are still "raw", if we allow a cooking analogy.



(a) Corner plot of model v3.4.3 on injection 999.00001.

(b) Corner plot of model v3.5.5 on injection 999.00001.

Figure 15

[10]This latter one is obtained with $c = \int_{-a}^{a} e^{-x^2/2} / \sqrt{2\pi}$. A brief note on the subject is available on the plotting package's documentation

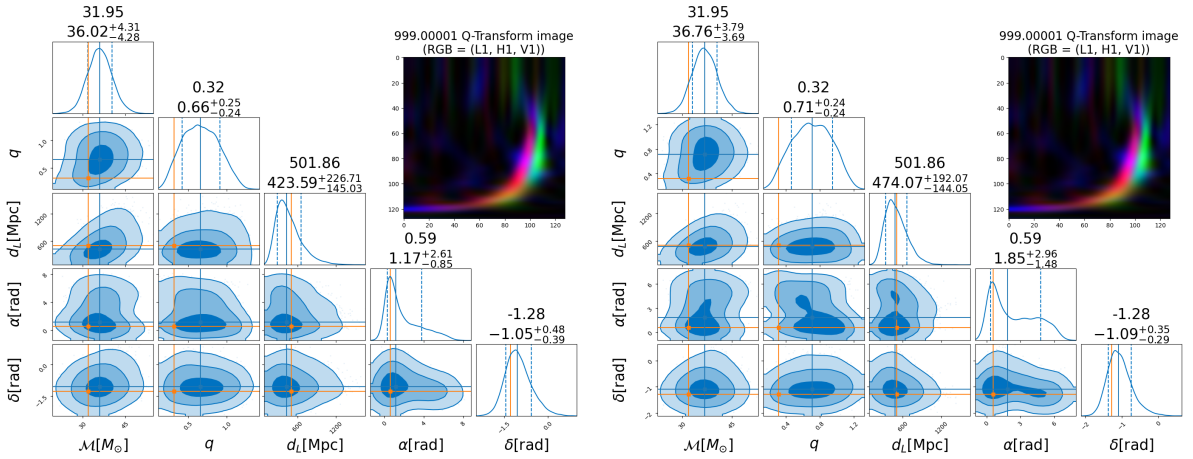
We have seen vaguely accurate results even with relatively low parameter space dimensionality (later on we will see how they fare against real events). However, the change in extractor philosophy I mentioned in the Section 3.2 allowed for much bigger datasets than before, which in turn enable the introduction of more parameters. Though we will start with similar dimensionality, these plots will soon become too large to fit side by side.

4.2 More interesting toy models

After doing away with pre-trained nets there was enough memory to increase dataset size by 400%, from 15 thousands to 60. But since computing power was being spent on generation I started testing what I call "partition training": circumvent the memory limitations of your system by training on your data in chunks.
If we were to train on each partition for a few epochs the model would forget what it had learned from the previous partition, so training time is significantly slower do to extra loading and converting of injections, which then led me to experiment with pre-converted data to be loaded directly onto a `TrainSet`. I generated a total of 120k images to be fed in batches of 30k. Batches of 60 thousand were technically possible and theoretically desirable, but required the use of local swap memory, which is slower, and tended to crash if the change of guard (unload/reload) took more memory than expected.
It is at this stage where I finally decided to include validation data during training, as I felt models where starting to learn enough to actually overfit to all this data (contrary to my belief at the time some of the 1 parameter models had actually overfitted).

Given that effective spin tended to be of near zero magnitude along the dataset, I decided to drop it for the time being, hoping it could concentrate training on the more variable parameters. Of these models the Spv1.4.2 family was the most successful, with Spv1.4.2.B3 the best performing underfitted model up until the introduction of scaling. The letter "B" denotes the batch number. It means that the model was trained with the whole 120k dataset for 3 epochs (So 12 actual stages). The model could still be trained further, it started overfitting at the beginning of the 20th stage, but mean performance had dropped slightly. Even so, pictures in figure 16 show that non-Gaussian behaviour is starting to appear.
For the first model, the estimation is within 1σ of the true value in all parameters except q. It is not as worrying as it may seem at first glance, as low mass ratios are notoriously hard to estimate. We will test our models in low q events such as that of figure 24. Despite being on the verge of overfitting[11] , the second model (figure 16(b)) seems generally better for this concrete injection. Sky position in particular is now almost within the bounds of the prior (see table 7), and the peak is still on the correct place.

[11] Closeness to overfitting can be known by producing a table like table 6, which would be expected to show the average accuracy almost at the 1 sigma precision level. In this case it was due to validation scores dropping the very next epoch during training.



(a) Corner plot of model Spv1.4.2.B3 on injection 999.00001.

(b) Corner plot of model Spv1.4.2.B6 on injection 999.00001.

Figure 16

As for the technical side, this family also implemented a much deeper model than previously attempted, taking full advantage of the custom construction.
A [2, 3, 4, 2] layout resnet (compared to the [2, 2, 2, 2] of resnet18, see table 1 of He et al. 2015) for 128 features, feeding into a 5-step flow with affine transforms controlled by nets with 64 hidden dims (as opposed a handful previously).

All of this extra depth forced an extra layer of complexity: weight checks.
As disused on the previous Section, affine autoregressive transforms have a sizeable spread on the loss associated to the first batch of data. Although it is generally a trait that will make training more difficult and comparison of models less reliable, this could be tuned in our favor by forcing the model to reset its parameters if initial loss did not fall under a certain threshold, essentially redrawing the dice. Since I was no longer using pre-trained networks this wasn't and issue, and with a bit more care I could have specified only to reset flow parameters.

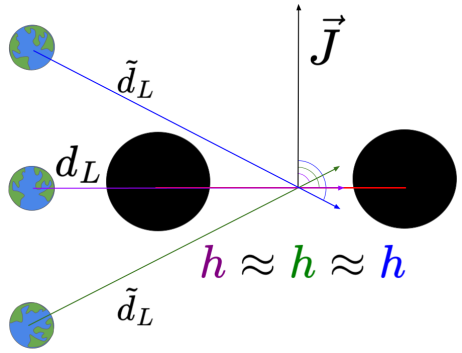


Figure 17: Diagram showing the apparent degeneracy between inclination and distance and between different inclinations. These can be resolved at least partially at higher multipoles, but can be a challenge for NPE models.

Results were slowly improving, so perhaps adding more parameters would result in better overall estimations. I decided to bring effective spin back and also add inclination, as a signal's strength is highly affected by it.

Initial tests were actually quite disappointing. By the time models overfitted the distributions were quite accurate, yet still highly guassian. Model family Spv2.3.x showed some improvement on the distributions, even if only slightly, but it wasn't until the 12th test (models 2.11.x) that plots started looking vaguely interesting.
These were starting to capture inclination bimodality (a reflection of the symmetry characteristic of the parameter, see diagram 17). A good reason for it would be the change in transform used for the flow. As opposed to only the affine I was using previously I incorporated a more complex transform: a rational quadratic coupling as seen in Section 2.1. This had been my initial choice when building the flow, but various technical difficulties made me opt for the affine instead.

The transform had an extra caveat. The initial loss estimation was now highly consistent, which meant that if the model was too deep it wouldn't train regardless of how many times I "turned it off and on again". To make the weight check effective once again I interlaced the rq couplings with affine transforms (see page 12 for more information), as my naive assumption that they would combine to produce a low but useful spread turn out to be spot on.

The following family of models implemented scales on its chirp mass and luminosity distance, and as I alluded to before, the results were very convincing (see figure 18).

This is the first time we see bimodality in a parameter, inclination, which could be mirroring the distribution of the parameter across the dataset, as seen in figure 11. Declination is spot on, but the distribution still spills from the prior. This was further improved by introducing scales, which I detailed in Section 3.2.

Mass ratio was still being stubborn, but for the first time since the initial chirp mass test I was ready to use these models with actual events, comparing them to actual GWTC samples (figure 19, on the next page).

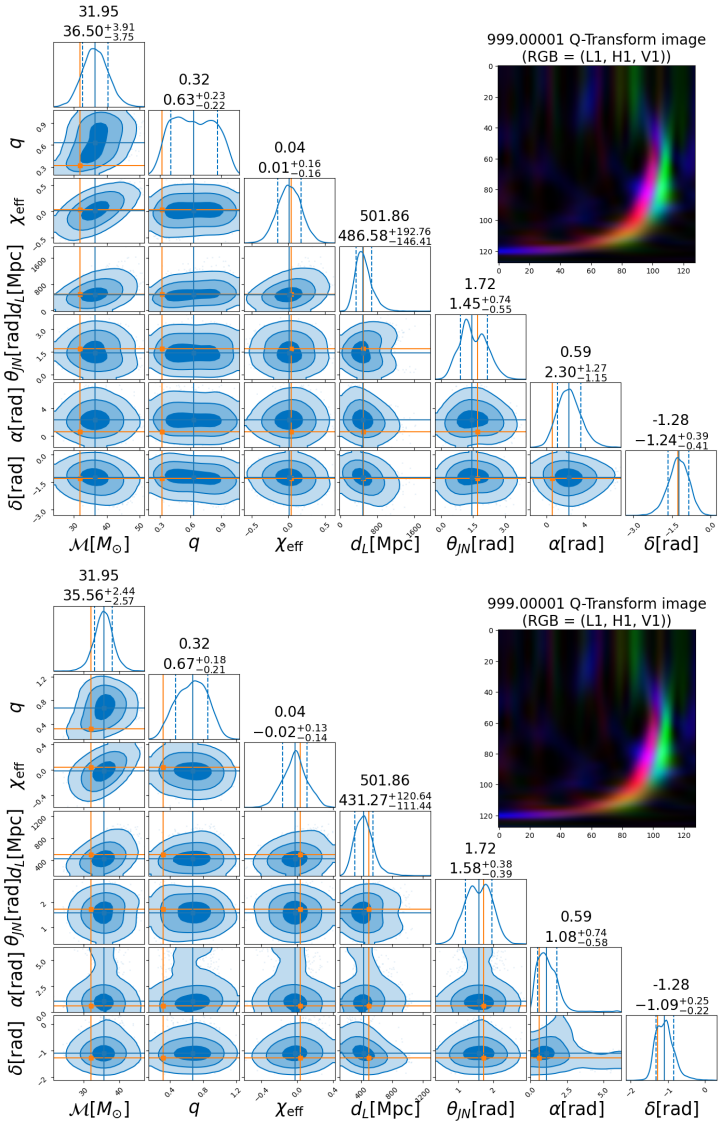


Figure 18: Corner plots showing the improvement derived from the introduction of scales. Models 2.11.2b and 2.12.0c.

In the analysis of said figure, chirp mass is accurate up to a 0.075%, though it is slightly less precise. Mass ratio and effective spin are also accurately estimated but more importantly, their distributions closely match that of the catalogue data. Luminosity distance is not as accurate but remains compatible. The d_L vs θ_{JN} plot shows the same cardioid-shaped features as the reference samples, albeit not as pronounced. Similarly, the "cathedral" structure typical of inclination bimodality is on full display. The main weakness of this estimation is clearly stellar position, in spite of the official data being already highly disperse.

In all fairness this event seems to have the right conditions to be accurately estimated. It makes a lot of sense given that the generation pipeline uses noise segments from August the 18th, 2017, just a few days before the event was recorded , so the detector's noise on the event may closely match that of the simulation data. That and the fact that it is fairly strong make it the ideal optimistic benchmark. Soon we will see how a bigger model performs on all public detections (figures 25 to 29).

On the other hand, GW170823 was only recorded by two detectors (notice the lack of blue in the image of figure 19), which is an excellent sign, since all training data consisted of L1, H1 and V1 detections. The model seems to understand how to infer a signal in spite of a missing detector, represented with a zero-filled array in the corresponding channel.

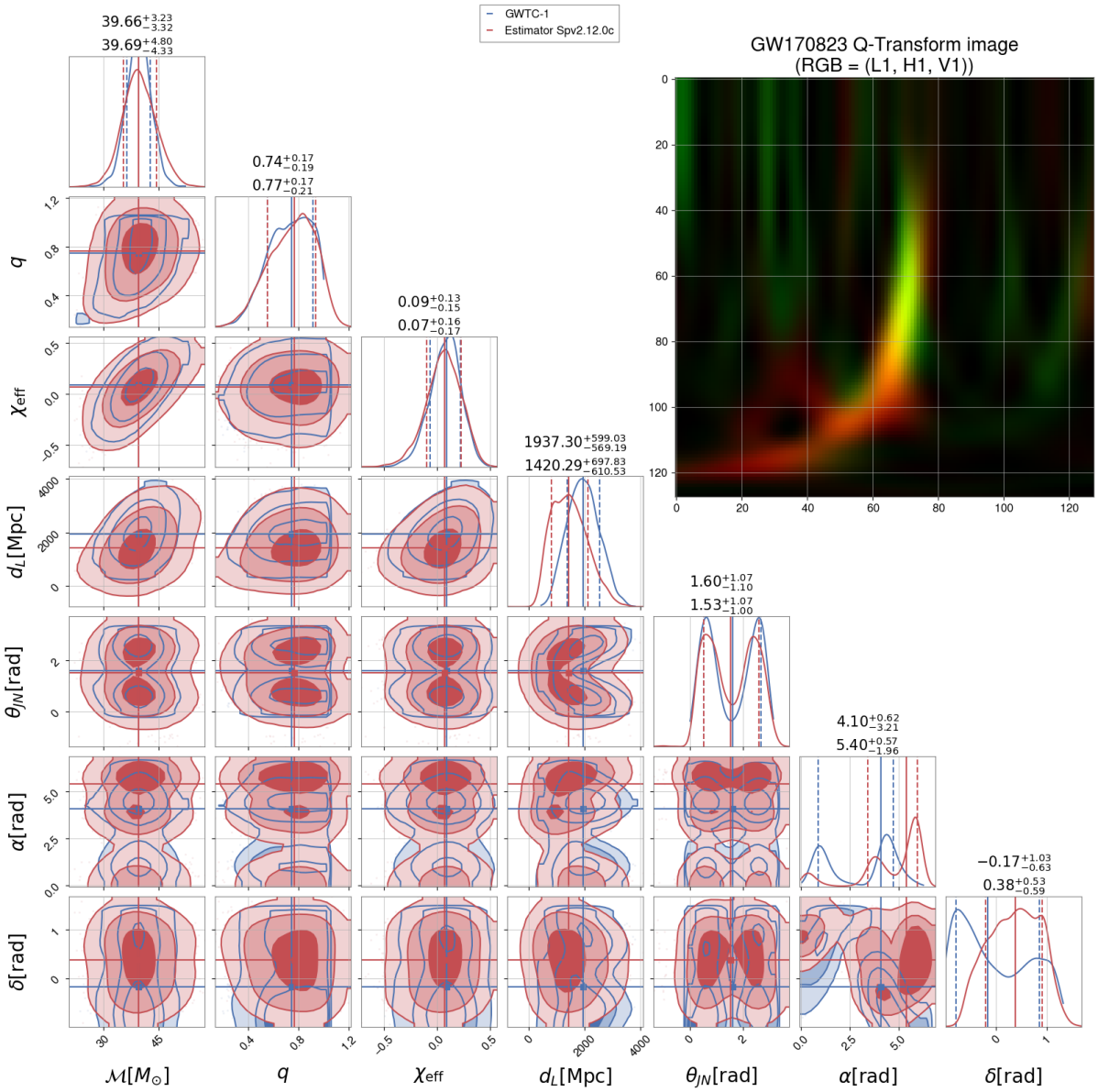


Figure 19: Comparison between one of my toy models (Spv2.12.0c) and GWTC (B. P. Abbott et al. 2019[c]) samples on event GW170823. 10000 samples generated with the image shown used unprocessed as context.

After these results I started wondering if perhaps higher dimensional models that up until this point seem unreachable could be attempted. I quickly trained a 10 parameter estimator and, upon witnessing similar results, trained a bit further. It overfitted, but as figure 20 shows an overfitted model may not be conceptually ideal, but it seems to be working just as fine if not better in a few places. The most interesting behaviour comes from right ascension. It drops the degeneracy of the stellar position almost completely. Normally I would think a model that gives lower uncertainties than reference estimations is probably not working correctly, but it is remarkably in line with GWTC median value predictions. It does, however, falter in declination, where it amplifies its predecessor's bet on a more centered distribution. Nonetheless, it is a relief to learn that overfitting models are not nan-returning garbage and that I therefore shouldn't cancel an hours-long stage if I see it starting to lose validation performance.

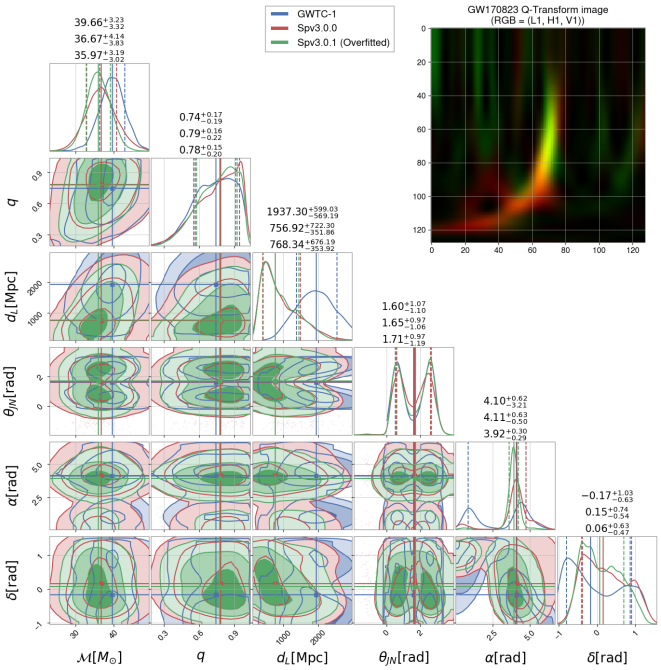


Figure 20: Comparison between a "normal" and an overfitted 10 parameter model (the latter being an extra 10 epochs of training added to the previous one). Results have clearly not degraded, but chirp mass shows that as a model overfits it becomes increasingly confident on its estimation, regardless of its actual accuracy. Missing 4 parameters for lack of certain spin parameters, phase and polarization angle in GWTC-1 samples.

With that tested, I decided to go a bit further and slowly increased parameter count to 12. Given that 12 parameters preliminary behaved just as well as all post-scaling flows, I went all in for a 14 parameter estimators:

4.3 Potential for competitive models

You may remember from our discussion on parameters back on Section 1.3 that, for BBHs with circular orbits, the full parameter space had 15 dimensions, but this work would only concern itself with 14. So, why is that? The answer is fairly simple: Not all parameters are born equal. Some (all but one, thankfully) can be implemented with no more than declaring it when creating the model, but there is one that breaks the mold when dealing with timeseries, whether directly or through a spectrogram like we are doing: Time itself. This is related to the fact there exists a certain symmetry regarding time translations.
Instead of estimating time of coalescence, we could estimate a nuance parameter that then helps us locate the merger time within the signal (bilby does something similar to this with its *time_jitter* parameter when marginalizing over it), but this would require alterations to existing datasets and generation routines and performance is far from guaranteed. As I pointed out in Section 1.3, some strategies to solve this are starting to appear (Dax et al. 2021[a])

With that said, a 14 parameter model should be a perfectly valid place to stop developing and start doing proper analysis, which is after all what will tell us if this endeavour is really worth pursuing. Another advantage of stopping now is that corner plots were really starting to get out of hand. A couple more parameters and they would no longer fit on the page, given figure 21.

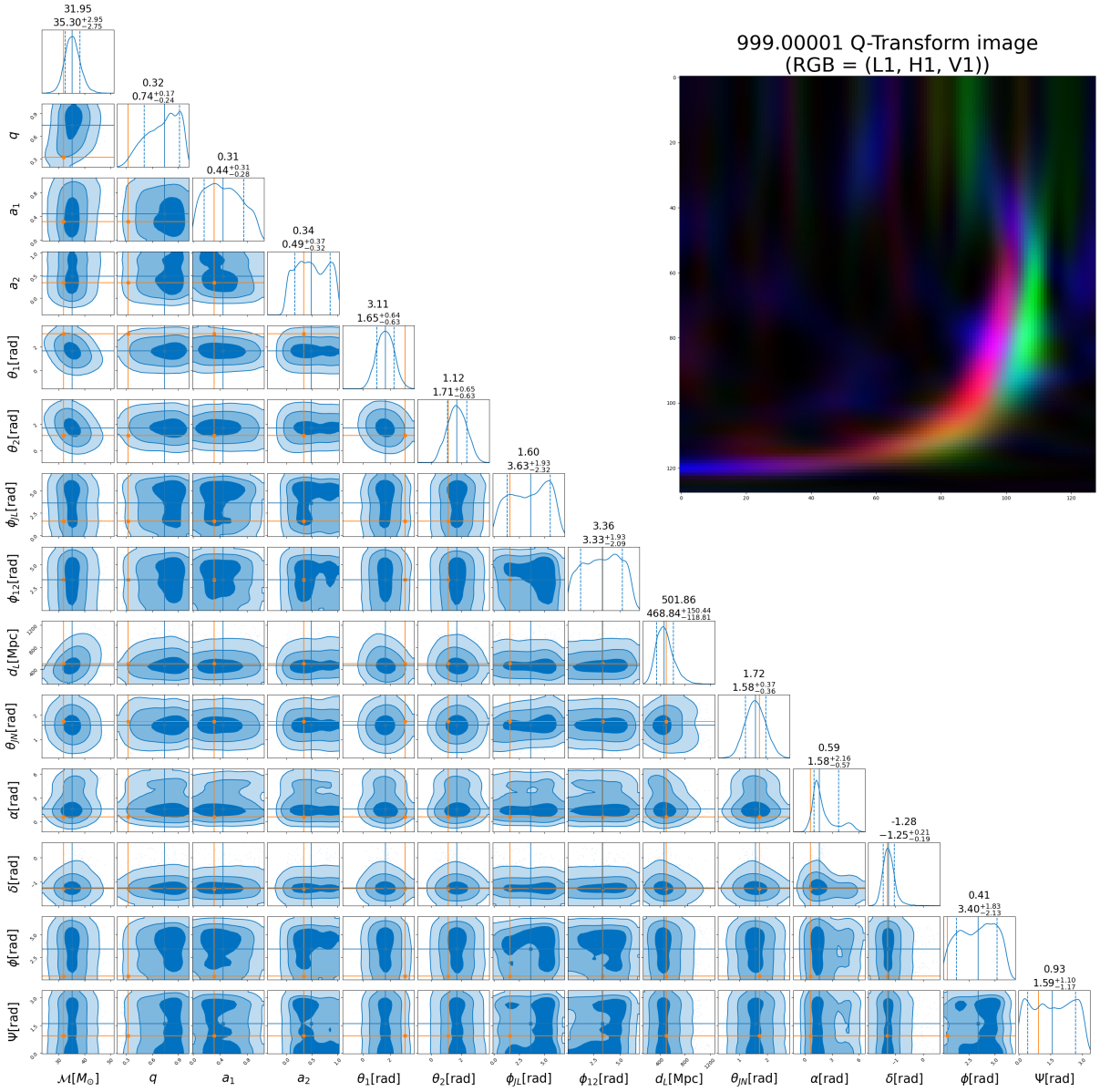


Figure 21: Example estimation of model Spv5.0.0 on the first injection of the validation dataset. 10000 samples generated with the image shown used unprocessed as reference context.

On that plot we can see that chirp mass is accurate, though on the edge of the 1σ level. Mass ratio is a disaster, but given the estimation of figure 19 it may be a common problem among estimation algorithms (an event suspected to be of low mass ratio is analyzed on figure 24). Most spin parameters are basically giving us our prior back, with θ_1 being somewhat worrisome. Luminosity distance and inclination are decent. Stellar position is better than expected but still too large to be of any use. Phase and polarization angle give us the prior back.

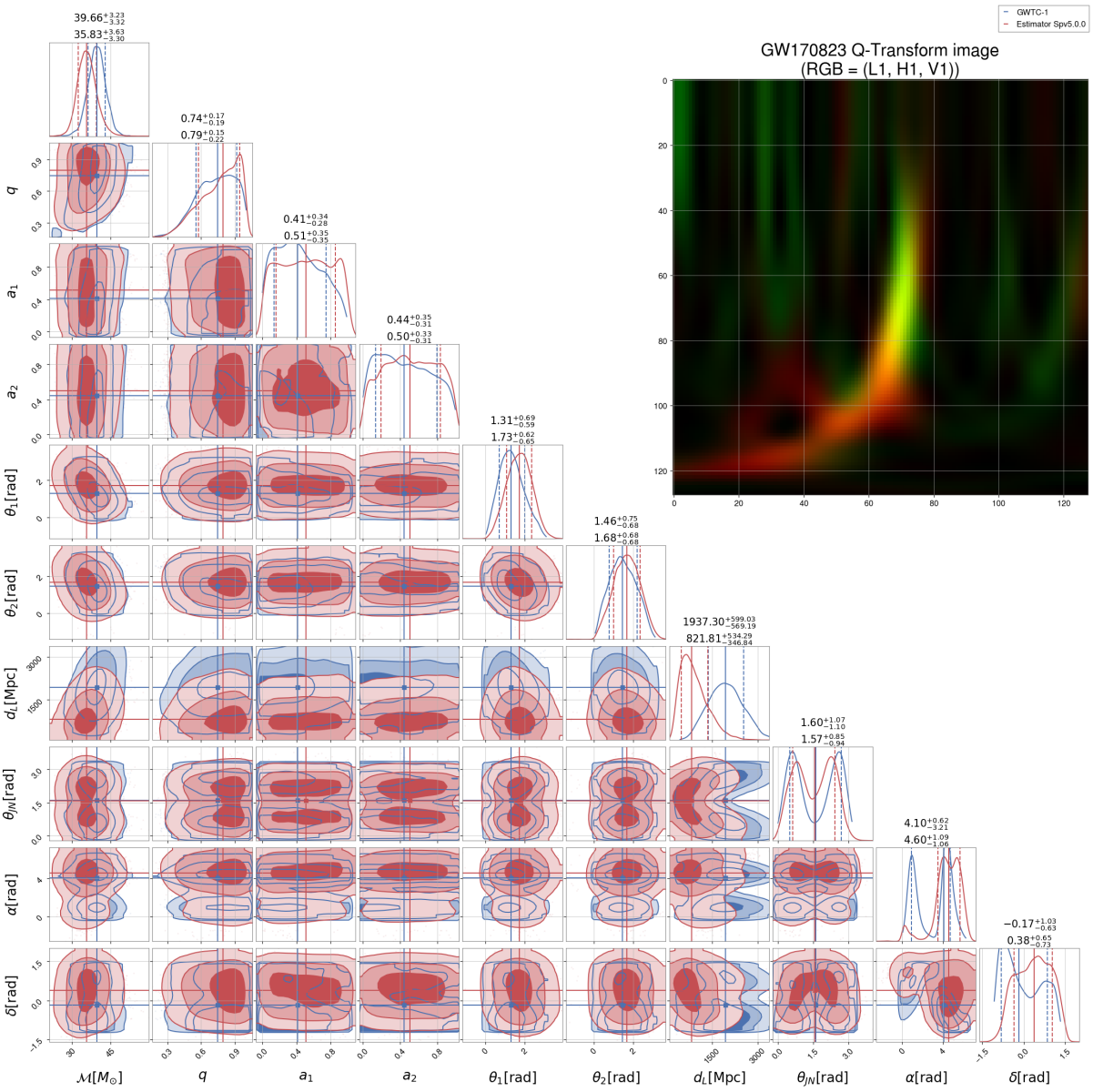


Figure 22: Comparison between my current flagship model (Spv5.0.0, 14 parameters) and GWTC-1 (B. P. Abbott et al. 2019[c]) MCMC samples on event GW170823. 20000 samples generated with the image shown used unprocessed as reference context.

On figure 22, chirp mass is on the border of compatibility, as was the case with the injection of figure 21, but it is underestimated compared to reference data as opposed to overestimated in simulations. Mass ratio is not as closely matched as what Spv2.12.0c achieved, yet it is still successful in capturing the overall tendency. Spin parameters are also generally matched, though the model remains agnostic on regards to a_i, while GWTC data slightly favors low spin magnitudes. ϕ_{JL} and ϕ_{12} are missing on any analysis of GWTC-1 data due to them missing on reference data and the inability of the PESummary conversion algorithms to generate them from existing samples. Luminosity distance is severely underestimated, which is to be expected given the distance limitations imposed on the training data by the minimum SNR criteria. It seems to have interpreted the signal by compensating the increase in power derived from it being closer by lowering its mass. Despite this, the characteristic features of d_L vs θ_{JN} and singular θ_{JN} are clearly visible. Like with the 7 parameter case, stellar position is considerably less precise than the already uncertain MCMC estimation. Phase and polarization data is also missing from the Montecarlo.

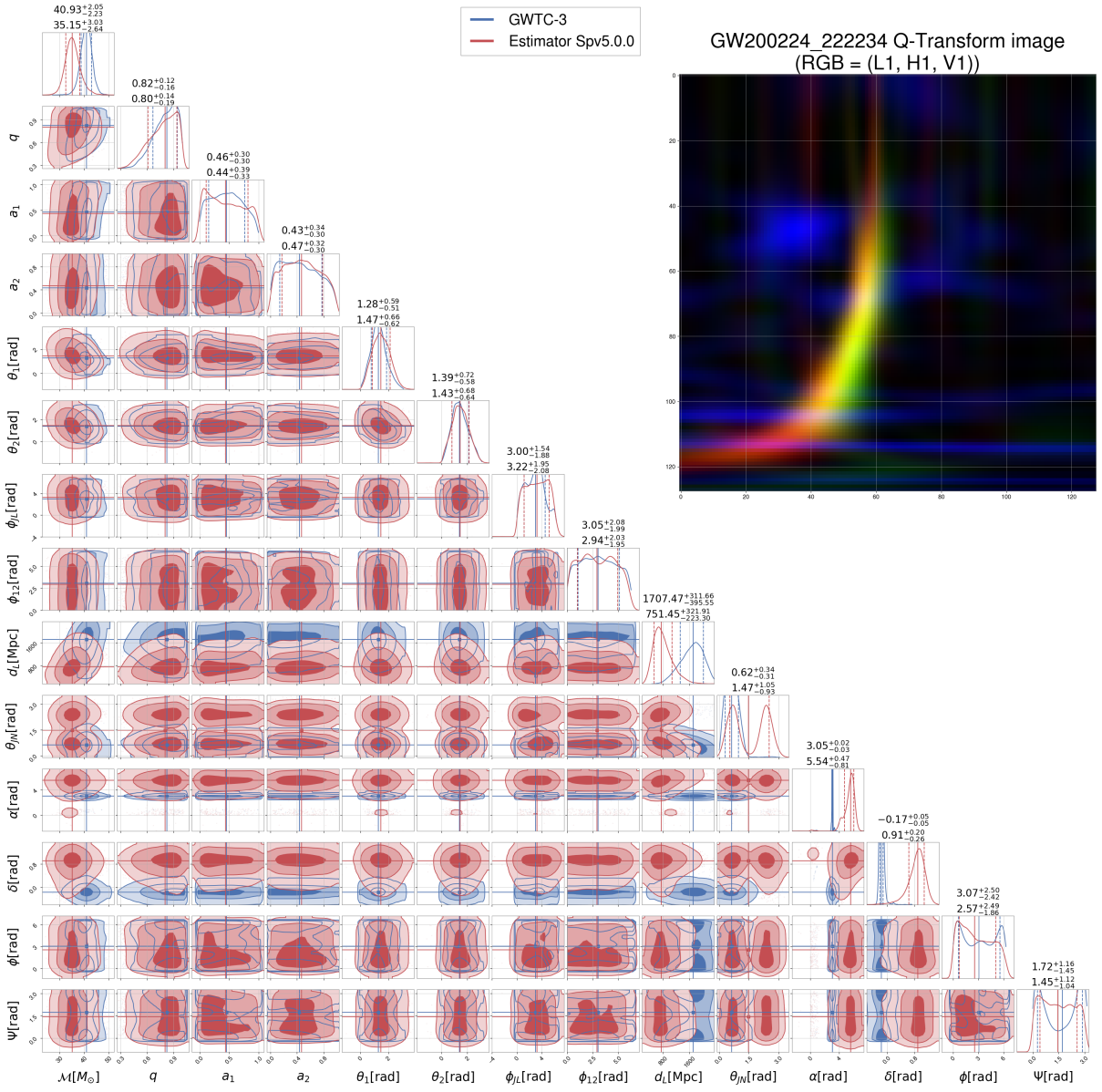


Figure 23: Comparison between my current flagship model (Spv5.0.0, 14 parameters) and GWTC-3 (R. Abbott et al. 2023) MCMC samples on event GW200224. 20000 samples generated with the image shown used unprocessed as reference context.

On figure 23, chirp mass is in this case incompatible, though still close at about 2 to 2.5σ difference. Mass ratio and spin parameters (all six this time) are more closely matched to Montecarlo than in 22, even if it is curious that now reference data is agnostic on a_1 and my model slightly prefers low values. Luminosity distance has the same issue, which again seems to be correlated with chirp mass underestimation. Inclination is interesting: My model has an even split between $\sim \pi/4$ and $\sim 3\pi/4$, while the GWTC data is certain it is $\sim \pi/5$. The degeneracy on inclination could have been broken by a better estimation of α and δ with information on coalescence time. The introduction of Virgo reduces sky position uncertainty massively. My model has also improved, but the difference between estimations is staggering, and it is something that will clearly need some more work. Phase and polarization angle are both favored at the extremes of the prior, which my model does capture to an extent. Nonetheless, polarization is also an area that could be improved.

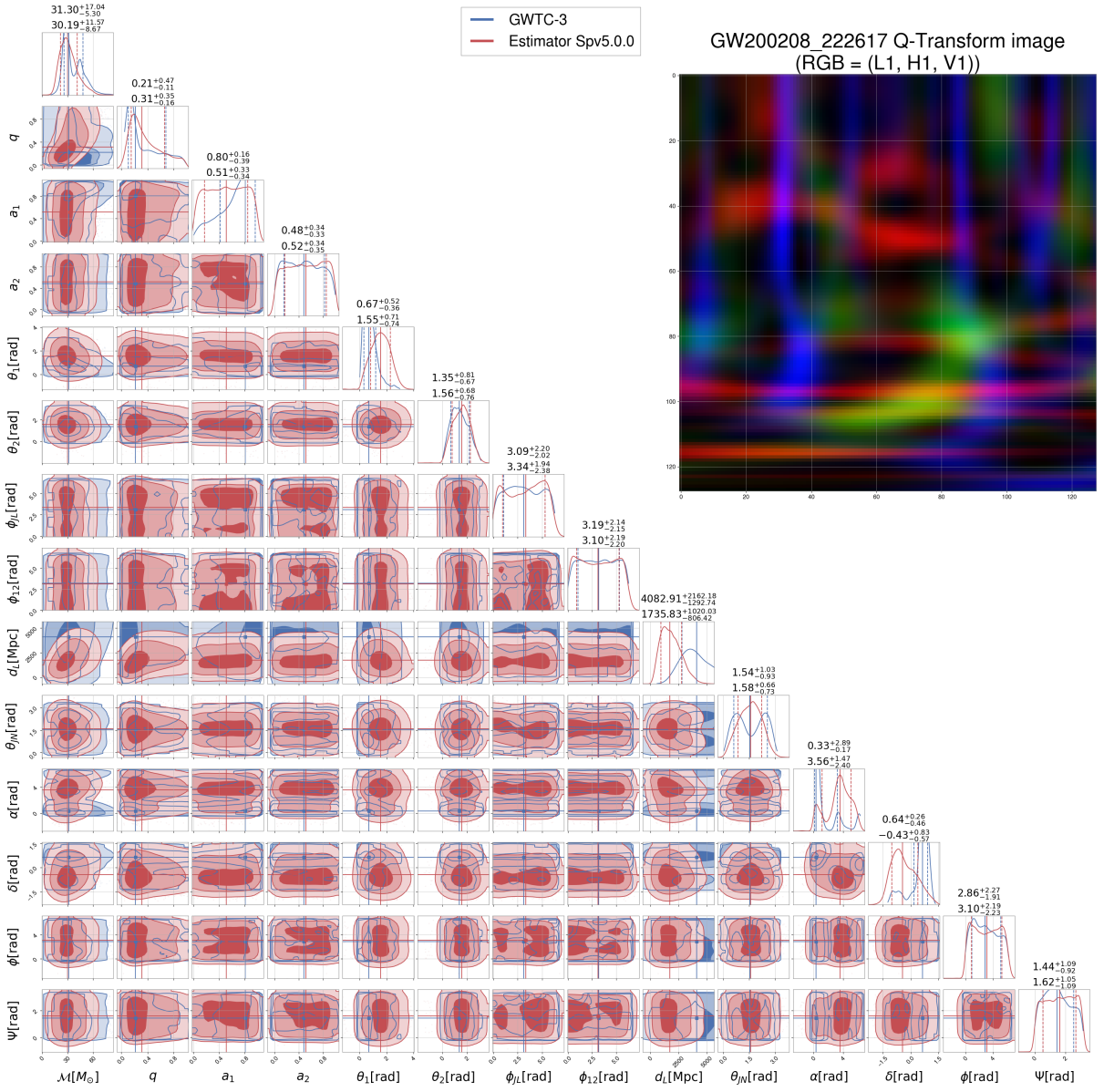


Figure 24: Comparison between my current flagship model (Spv5.0.0, 14 parameters) and GWTC-3 (R. Abbott et al. 2023) MCMC samples on event GW200208. 20000 samples generated with the image shown used unprocessed as reference context.

And on figure 24, chirp mass is closely matched, though it is interesting to see bimodality in this parameter. The model's ability to understand events with high mass asymmetry is a pleasant surprise. Spin parameters are generally great (though Montecarlo has a preference for high a_1). Luminosity distance was estimated to be over my actual distance prior by GWTC-3, so I wasn't expecting much. Still, the model is able to infer that the signal is distant, even if it underestimates it again. θ_{JN} is now opposite to that of figure 23: Montecarlo estimates a degeneracy and I do not, although the fact that my distribution is centered between the peaks leads me to believe that it is not choosing one over the other, but rather just averaging them. Stellar position is not great but at least it focuses on one of the three peaks of th reference data, albeit not the one the MCMC favours.

Given the page limitations on Master's Theses, showing sample comparisons for all cataloged events would be highly impractical. Standard deviation comparisons are shown instead. They allow us to get an idea of performance over a catalogue at a glance, but we ought to be careful, as these distributions are clearly not Gaussian.

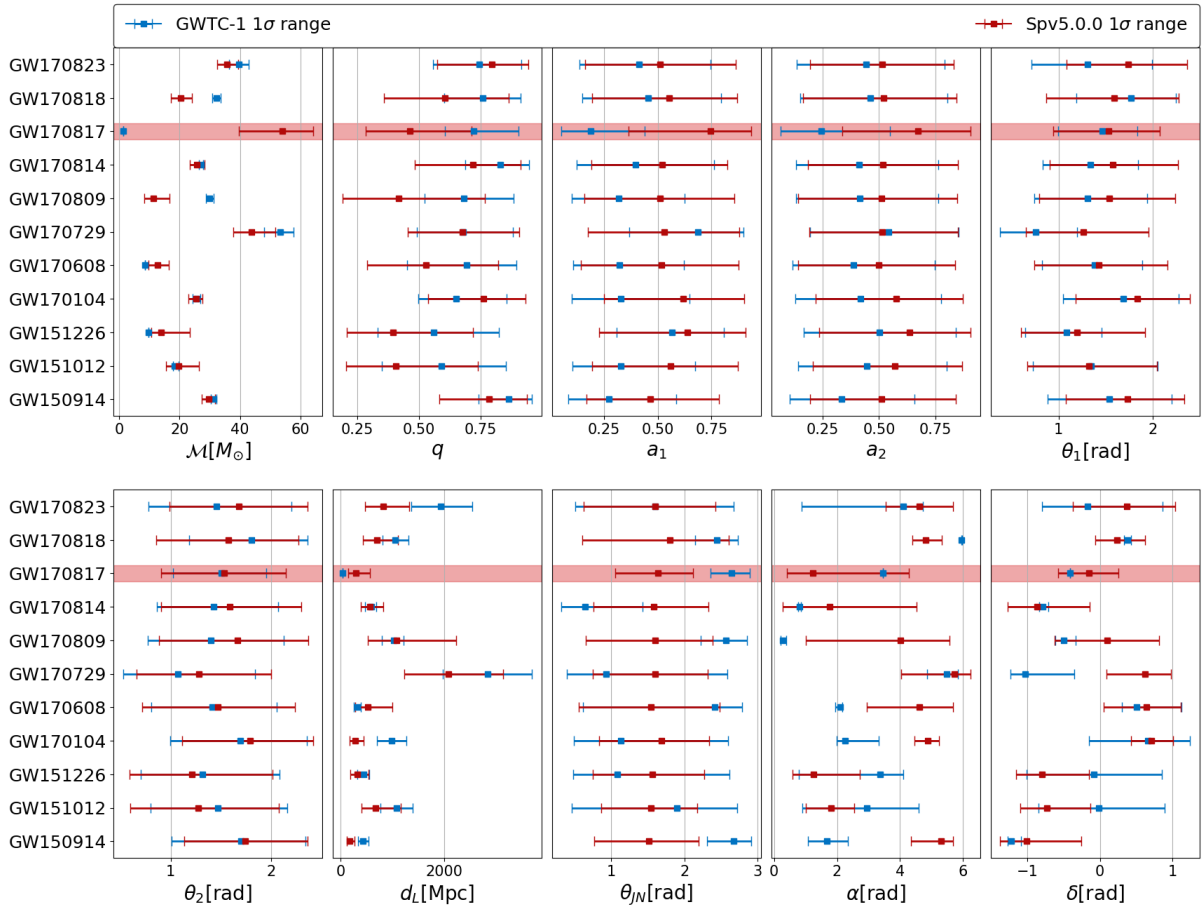


Figure 25: Comparison of 1 standard deviation ranges for all 11 events of GWTC-1 (B. P. Abbott et al. 2019[c]) for the 10 available parameters from publicly released samples. The event marked in red, GW170817 (B. P. Abbott et al. 2017; B. P. Abbott et al. 2019[a]), is a BNS, and therefore has completely different mass priors. For that reason, our model should not be expected to perform adequately on this or any event that isn't a BBH. In fact, performance on low mass systems ($\mathcal{M} \lesssim 20 M_\odot$) is surprisingly accurate as no extra low-mass datasets were used, as was the case in Álvares et al. 2021.

Figure 25 shows that performance is successful, though uneven. As hinted in figure 22 sky position is its biggest failure, though luminosity distance is more compatible catalogue-wise than it was looking only at GW170823. Chirp mass is either compatible or completely off (again, to be expected in GW170817, but not for example in GW170818), while spin parameters are estimated with similar levels of uncertainty as official data.
One important thing to note is that this analysis have not been made individually but on a semi-automatic manner. This could lead to cutoffs of the signal if it were to fall outside the region of the Q transform window. No complete cutoff occurred and no significant degradation of estimation quality was observed on partial cutoffs, but a more definitive analysis would implement custom time windows for each event to ensure the signal is properly estimated. A comparison with a more recent model is presented on appendix A, figure 34.

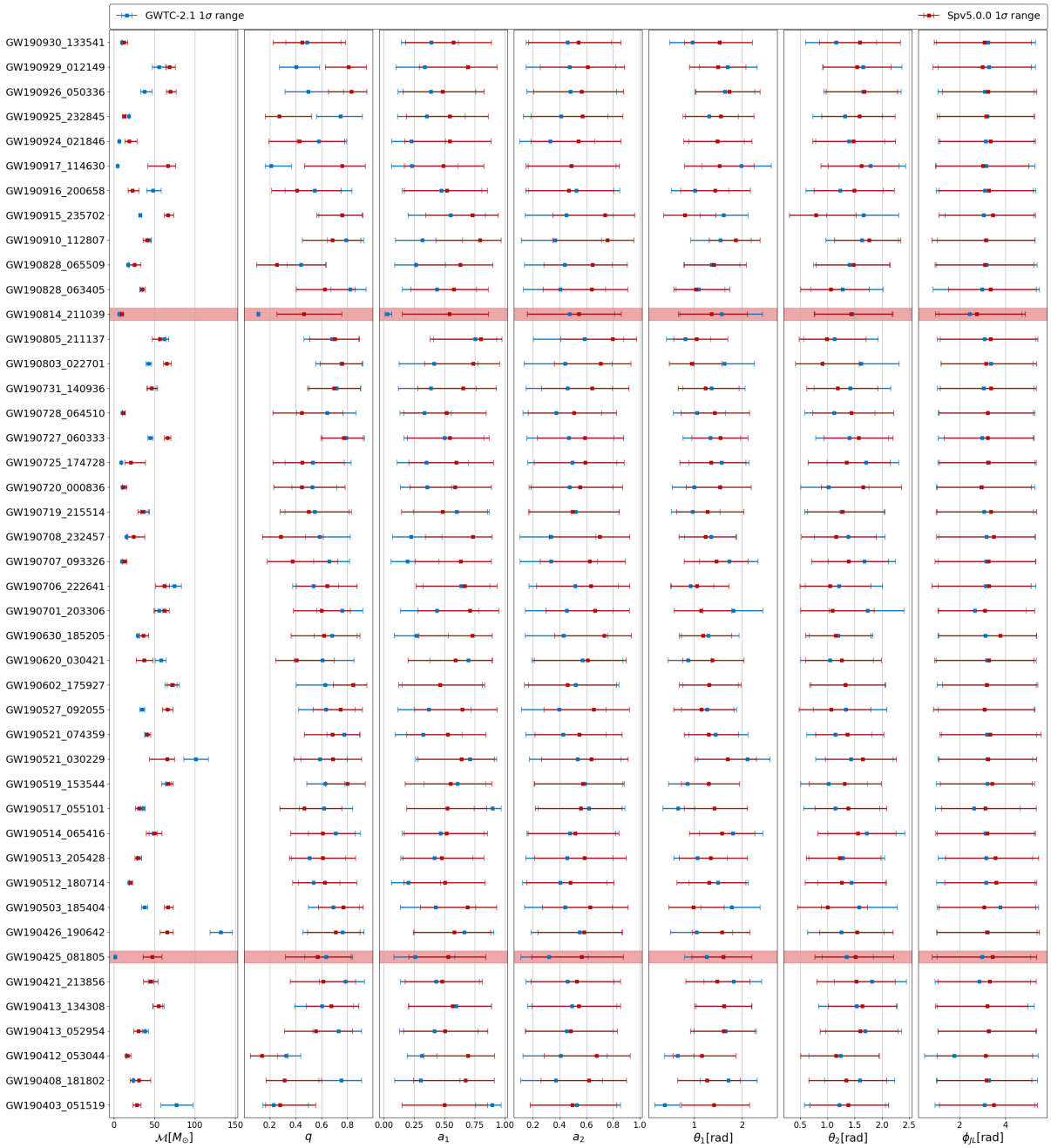


Figure 26: Comparison of 1 standard deviation ranges for all 44 events of GWTC-2.1 (R. Abbott et al. 2024) for the first seven parameters of my model. Again, events GW190425 (B. P. Abbott et al. 2020[b]) and GW190814 (R. Abbott et al. 2020[a]) are BNS and mass gap events respectively, so are included mainly for completeness and for taking the model to its limits.

Figure 26 implies the model seems eager to not trespass the 70 $M_\odot$ mark (inline with figure 11), which becomes a problem in 03-based catalogues with more massive events, such as GW190521 (R. Abbott et al. 2020[b]). Mass ratio remains compatible, even in some extremal cases (GW190403, GW190727). Failure to correctly estimate one mass parameter does not seem to be correlated with missing the other. Spin parameters appear to keep performing, although in some cases the model fails at high spin magnitude (GW190403, GW190517). Underestimating a_1 might be correlated with overestimating θ_1, given these two events.

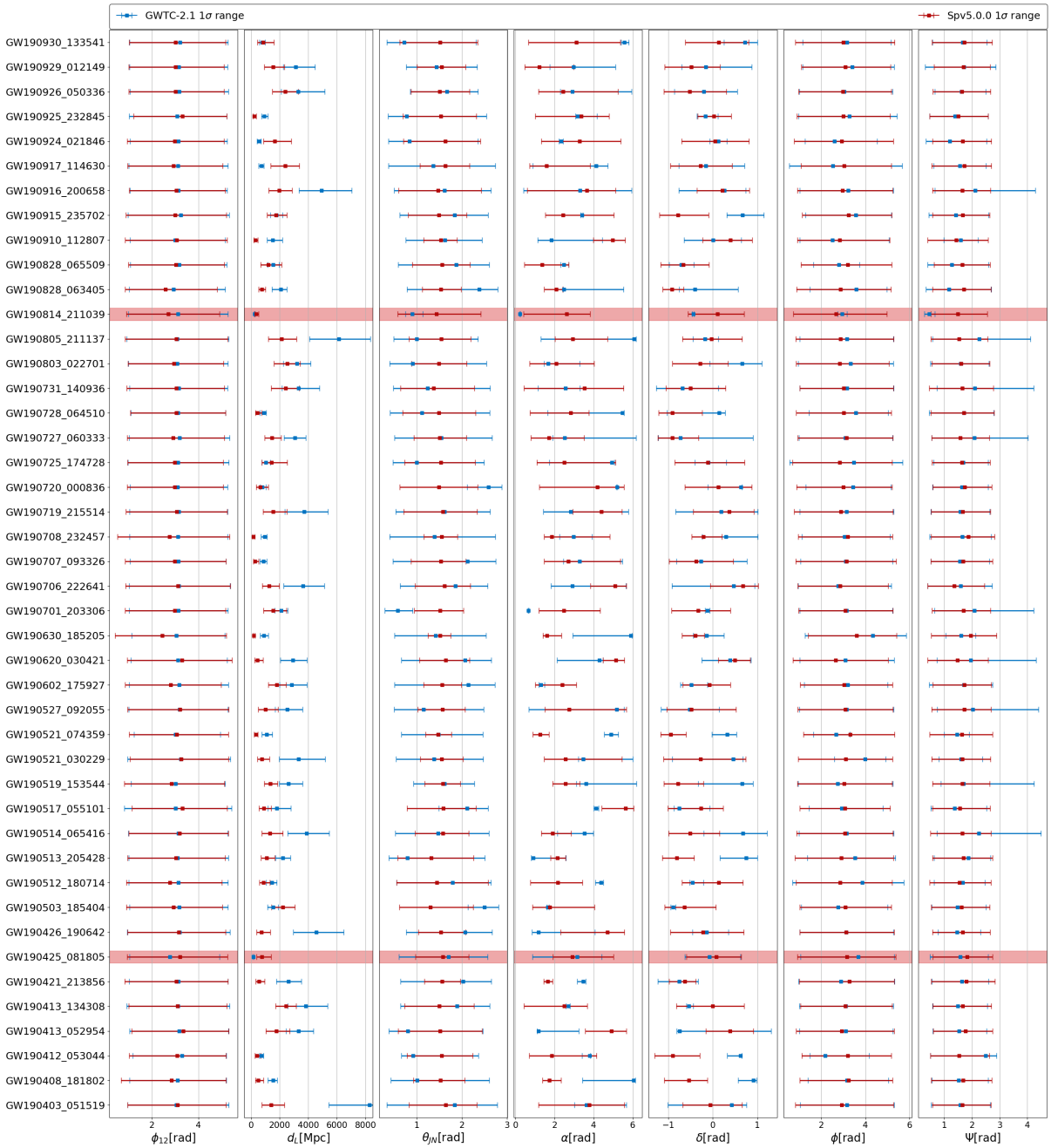


Figure 27: Continuation of figure 26, including the other 7 parameters of the model.

To properly study ϕ_{12}, ϕ and ψ one needs at the very least the 1D histograms to observe possible favoring of lower or higher values within the relatively uniform posteriors, especially in ψ, as can be seen in the right-most plot of figure 23. From figure 27, performance on luminosity distance has differed the most, as now it continuously underestimates it (though not consistently, which could have been corrected). This is a clear sign that future models need a higher and more uniform distance prior and a wider range of noise patters, not just from different moments of a continuous timeseries, but from different years. Sky position needs a lot of work in terms of both accuracy and precision.

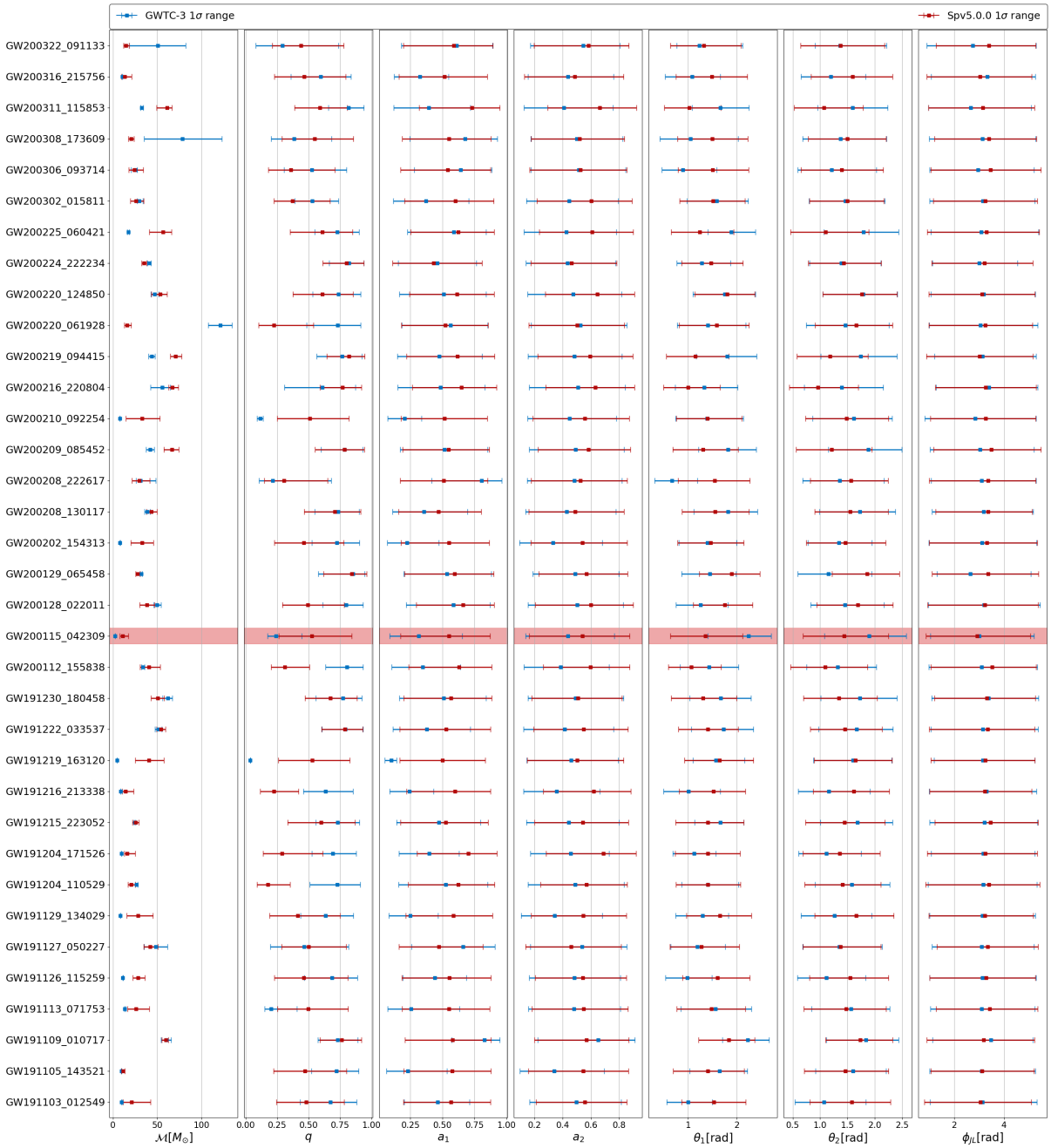


Figure 28: Comparison of 1 standard deviation ranges for all 35 events of GWTC-3 (R. Abbott et al. 2023). As in previous catalogues, non-BBH events are marked in red. This time it is GW200115 (R. Abbott et al. 2021[a]), the confident BHNS of the two detected in early 2020.

The model overestimates a_1 in one occasion (GW191219), but there are no high-spin cases in figure 28 to test against figure 26. An important caveat is that this analysis have been taken with a predetermined Q transform window for all events. CBCs such as GW200112 or GW200311 may have been affected by having part of their signal cut off. As previously discussed, a manually adjusted window for those events would be something to consider in the future.

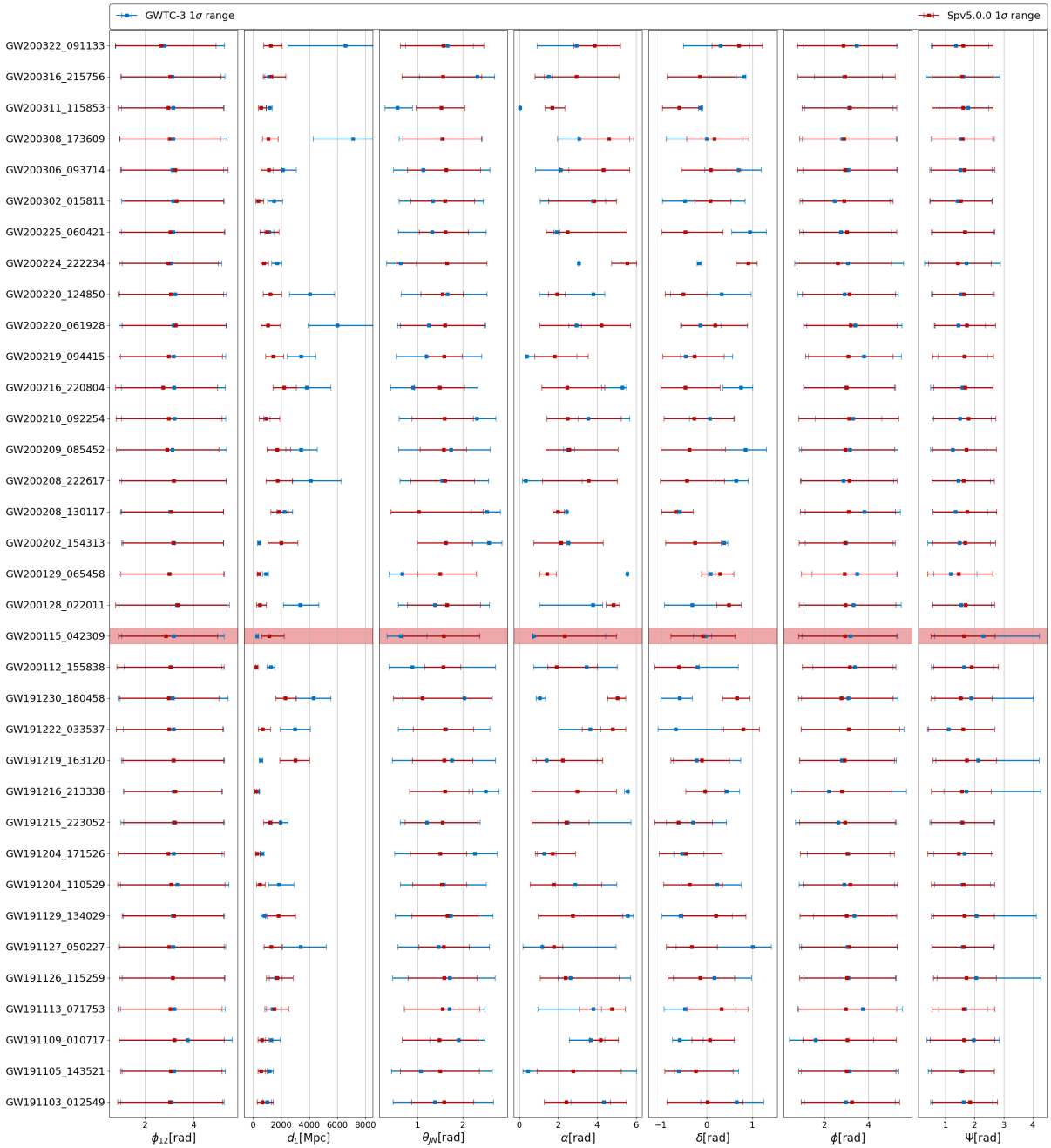


Figure 29: Continuation of figure 28, including the other 7 parameters of the model.

Luminosity distance appears to be performing slightly better on figure 29, but still far from useful, and unable to go over 4000 Mpc. This is normal as the dataset it was trained on went that far and showed a clear bias for shorter distances (figure 11). Other than that it seems to have the exact same weaknesses as in GWTC-2.1.

5 Conclusions

This work has merged two machine learning philosophies, residual networks and normalizing flows, into a general-purpose image-based estimator. A multitude of different models have then been trained on spectrograms made from data simulated in-house and then tested on publicly available data (GWOSC) from LIGO and Virgo's first three observing runs.
Following Álvares et al. 2021, each of the three detectors were adjudicated a color channel for the resnets to then transform into a feature vector for the flows to use as context when sampling.

Given that increasing parameter space carried no substantial increase in complexity or difficulty of training once parameters were scaled down, most of the focus went into two configurations: A 7 parameter model meant to characterize the most relevant information of a coalescence and a 14 parameter model to give an (almost) full description of the event. Both of these were tested for the complete GWTC catalogs, and showed remarkable agreement over the majority of parameters, with chirp mass, luminosity distance and sky position (right ascension and declination) being the only parameters with a non-anecdotal rate of incompatibility.

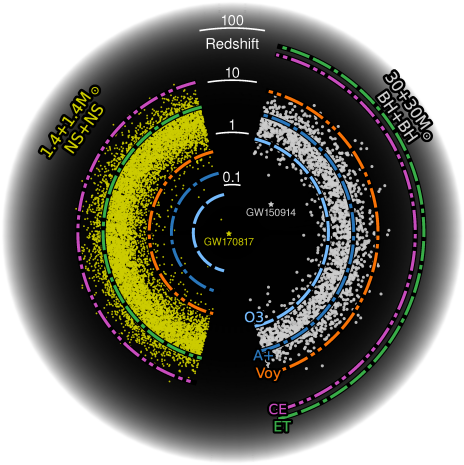


Figure 30: Detection range of present and future detectors along with distributions of BNS and BBH events as a function of redshift. One can see the relative luck we had observing GW170817, yet it is also a hopeful look into the era of multi-messenger astronomy, since BNS detections ought to become a normality. Image obtained from Hall 2022.

Moreover, the initial goal of proving that the sampling velocity of machine learning systems made their training overhead an inconvenient yet necessary trade-off has been not only achieved but surpassed, demonstrating that a model that produces samples comparable to official MCMC results for an arbitrary BBH not only samples faster, but trains in less time that the average Montecarlo takes to sample a single event. This speed will be a decisive factor for future observation campaigns (figure 30), as the probability of finding candidates of interest to multi-messenger astronomy will continue to rise as detector range improves and detection-characterisation-alert pipelines of the LVK Collaboration will have to function at an ever-increasing frequency.

With these results I am confident that a more robust model could be trained in proper hardware in the $O(10)$ hours often cited for these events (Section 3.3.4 of I. M. Romero-Shaw et al. 2020). Even so, it is an especially substantial achievement given the hardware limitations of my system, restricting the size of the datasets. By the time this work is presented further progress may have been achieved with the use of the Artemisa computing cluster. Other improvements could include a proper normalization of the images, which was dropped along with pre-trained networks temporarily and never recovered, or slightly downscaling images (to 96 by 96 pixels, for example) and experiment with rectangular formats.

To collect this work's findings:

- These technologies can accurately model the high dimensional parameter space of a BBH.
- Given that estimations remain compatible on most parameters when tackling other types of CBCs, it is reasonable to suggest that the library could be applied to BNSs and BHNSs with similar results on the future by generating appropriated datasets.
- They also remain stable when analyzing data captured in only 2 detectors (and independently of which one was off-line), showing meaningful learning of the physics behind CBCs.

- Estimations tend to match MCMC best when the latter is unsure (usually q, a_i, ϕ_{JL}, ϕ_{12}, ϕ and ψ), but can also give matching results in more confidently regressed parameters such as $\mathcal{M}$ or θ_i. Considering that works such as Green and Gair 2021 have produced near-identical distributions compared to MCMC with deeper models and using machine learning techniques similar to mine, it is appropriate to take GWTC samples as truth values, although we have to keep in mind that these are not injections after all and therefore deviations are not necessarily mistakes. A more robust model could challenge GWTC results if they were to present serious incompatibilities.

- One of the main weakness is luminosity distance (d_L), which is often underestimated due to minimum SNR requirements leaving large distances disfavoured or entirely out of the training data. Use of O2 noise in training data could also explain bias against more distant events. A runtime-based sampling of extrinsic parameters and an extra channel for PSD data (such as that of Dax et al. 2021(b)) may prove helpful, though abandoning a 3 channel structure could restrict embedding architectures.

- The other main weakness is sky position (α, δ): Performance on these two parameters ranges from accurate but imprecise to highly "erroneous" (again, when compared to MCMC samples). Regardless, subsequent training of models suggest an increase in dataset size may solve these inaccuracies.

- Overfitting seems to degrade estimation quality slower than expected, meaning overfitted models could still be used. However, an unsure model is preferable over a confidently incorrect one, so overfitting should be avoided.

- As with any machine learning task, training data quality plays a crucial role in the final results. The purpose of this work was to explore how a general purpose estimator could fare based on the dataset generation techniques described in Álvares et al. 2021, but future works are likely to require specialization of the simulation data. Early attempts of this concept can be seen in figures 36 and 37, where the model is trained on various O2 noise samples and with LIGO interferometers only.

On a final and more personal note I want to stress the generality of these systems. This architecture can easily be molded into a variety of use cases. In fact, if you need image-based parameter estimation and your data is either plentiful or "easily" simulated, this software is likely to be of use.

As I stated before, these results are far more meaningful than anything I was hoping to achieve. When my advisors suggested that a high dimensionality model could be attempted I thought it was an interesting prospect, yet far-fetched. By late April this was still the case, and I was focusing on over-training a model to conclude that this technique was promising because I had proven that it **could** work, even if it didn't yet.

Now I can confidently say that the model works. It is not perfect, of course, but considering not just the aforementioned hardware limitations but also that it was done by someone with no prior machine learning experience, it is quite a start.

A Parameter tables for discussed events

Physical parameter	Prior: Uniform except told otherwise	Units	Technical parameter	Value(s)
mass_i	[5,100]	$M_\odot$		
a_i	$[0, 1]$	$\varnothing$	approximant	IMRPhenomPv2
tilt_i	$[0, \pi]$	*rad*	minSNR	5
phi_jl	$[0, 2\pi]$	*rad*	filt_order	512
phi_12	$[0, 2\pi]$	*rad*	q_interval	$(-0.15, 0.1)$
ra	$[0, 2\pi]$	*rad*	num_workers	2
dec	$[-\pi/2, \pi/2]$*	*rad*	chunksize	5
d_L	$[100, 4000]$	Mpc	seed_zero_pad	3
theta_jn	$[0, \pi]$**	*rad*	log_file	None
psi	$[0, \pi]$	*rad*		
phase	$[0, 2\pi]$	*rad*		
tGPS	1187058342	*s*		
tau	$[8, 492]$	*s*		

Table 7: Default priors for generation of datasets. Notice that luminosity distance is referred to as 'd_L'. I renamed the parameter before conversion and was changed for subsequent generation pipelines. Non-uniform priors are uniform in cosine (*) or sine (**).

Physical parameter	Prior: Uniform except told otherwise	Units	Technical parameter	Value(s)
chirp_mass	$[20, 60]^\dagger$	$M_\odot$		
mass_ratio	$[0.125, 1]^\dagger$	$M_\odot$		
a_i	$[0, 1]$	$\varnothing$	approximant	IMRPhenomXPHM
tilt_i	$[0, \pi]$**	*rad*	minSNR	5
phi_jl	$[0, 2\pi]$	*rad*	filt_order	None
phi_12	$[0, 2\pi]$	*rad*	q_interval	$(-0.1, 0.1)$
ra	$[0, 2\pi]$	*rad*	num_workers	12
dec	$[-\pi/2, \pi/2]$*	*rad*	chunksize	None
luminosity_distance	$[100, 5000]^\ddagger$	Mpc	seed_zero_pad	4
theta_jn	$[0, \pi]$**	*rad*	log_file	None
psi	$[0, \pi]$	*rad*		
phase	$[0, 2\pi]$	*rad*		
tGPS	(1185366342, 1187058342, 1187490342)	*s*		

Table 8: Priors for generation of the O2-based dataset using the new generation pipeline. Non-uniform priors are uniform in mass components (†), source frame (‡), cosine (*) or sine (**).

	Name (bilby)	Description	Latex	Units
Relatives to mass	mass_i	ith object's mass on the detector's frame	m_i	$M_\odot$
	chirp_mass	chirp mass on the detector's frame	$\mathcal{M}$	$M_\odot$
	total_mass	total mass on the detector's frame	M	$M_\odot$
	mass_ratio	ratio $m_</m_>$, ≤ 1	q	$\varnothing$
	symmetric_mass_ratio	redefinition independent of which mass is biggest $(q/(1+q)^2)$	η	$\varnothing$
	mass_i_source	$m_i/(1+z)$: m_i on the source's frame	m_i^{source}	$M_\odot$
	chirp_mass_source	$\mathcal{M}/(1+z)$: $\mathcal{M}$ on the source's frame	$\mathcal{M}^{source}$	$M_\odot$
	total_mass_source	$M/(1+z)$: M on the source's frame	M^{source}	$M_\odot$
Relatives to angular momenta	a_i	ith object's dimensionless spin	a_i	$\varnothing$
	tilt_i*	zenith angle between vectors $\vec{l}_i$ and $\vec{s}_i$ of ith object's	θ_i	rad
	cos_tilt_i*	cosine of angle θ_i	$\cos(\theta_i)$	$\varnothing$
	phi_jl*	diference between azimutal angles from vectors $\vec{J}$ and $\vec{L}$	ϕ_{JL}	rad
	phi_12*	diference between azimutal angles of the projection of vectors $\vec{s}_i$ on to the orbital plane	ϕ_{12} o $\Delta\phi$	rad
	chi_i*	ith object's dimensionless spin aligned with $\vec{l}_i$: $\chi_i = a_i \cos(\theta_i)$	χ_i	$\varnothing$
	chi_i_in_plane*†	ith object's dimensionless spin projection on to the orbital plane: $\chi_i^\perp = \lvert a_i \sin(\theta_i)\rvert$	$\chi_i^\perp$	$\varnothing$
	chi_eff*†	effective inspiral spin	χ_{eff}	$\varnothing$
	chi_p*†	effective precession spin	χ_p	$\varnothing$
	spin_i_k*†	kth component of ith spin in euclidean coords.	$S_{i,k}$	$\varnothing$
Relatives to position	ra	right ascension (from 0 to 2π)	α	rad
	dec	declination (from $-\pi/2$ to $\pi/2$)	δ	rad
	zenith	zenith angle with respect to the detector	κ	rad
	azimuth	azimuthal angle with respect to the detector	ϵ	rad
	luminosity_distance	luminosity distance	d_L	Mpc
	comoving_distance	comoving distance	d_C	Mpc
	redshift	cosmological redshift (requieres assuiming a particular cosmology)	z	$\varnothing$
Relatives a la orientation	iota*	inclination angle of orbital plane	ι	rad
	cos_iota*	cosine of ι	$\cos(\iota)$	$\varnothing$
	theta_jn	inclination angle of plane perpendicular to $\vec{J}$	θ_{JN}	rad
	cos_theta_jn	cosine of θ_{JN}	$\cos(\theta_{JN})$	$\varnothing$
	psi	polarization angle	Ψ	rad
	phase	phase in t_c	ϕ_C	rad
	geocent_time	GPS coalescence time	t_C	s
	IFO_time	GPS coalescence time on certain IFO (L1, H1, etc)	t_{IFO}	s
	time_jitter	shift to apply in time marginalization	δt	s
	eccentricity*†	orbital eccentricity	e	$\varnothing$
matter	lambda_i	dimensionless tidal deformability of ith object	Λ_i	$\varnothing$
	lambda_tilde	combined deformability	$\tilde{\Lambda}$	$\varnothing$
	delta_lambda_tilde	relative difference of $\tilde{\Lambda}$	$\delta\tilde{\Lambda}$	$\varnothing$

Table 9: Exhaustive table of possible parameters of a compact binary coalescence. Not all of them are compatible with each other since they describe the same physical observables, as is the case, for example, with q and η. Parameters used in full estimation of this work shown in lilac. Superscripts: * implies definition with respect to reference frecuency, † implies inability of sampling, but capability of *a posteriori* generation. Table adapted from apendix 'E' of (I. M. Romero-Shaw et al. 2020).

B Tangent discussions

B.1 Network Antenna Pattern

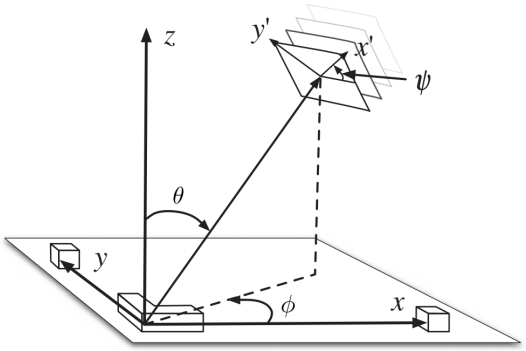
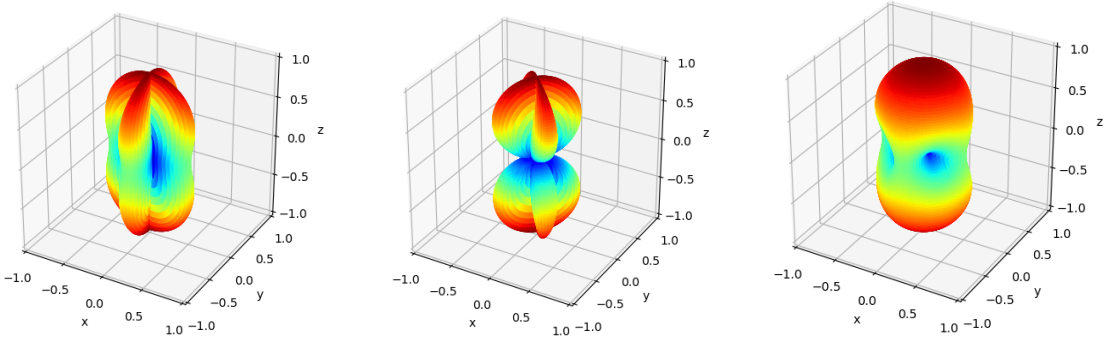


Figure 31: Frame of reference of an incident wave with respect to the detector. Diagram taken from Abadie et al. 2010.

Gravitational wave detectors can be compared to microphones due to their all sky detection and current frequency sensibility. Much like microphones, interferometers also have blind-spots. For example, in ideal conditions (+ polarization, with null azimuthal, polar and polarization angles) the detector is at its most sensitive. However, if we are to rotate the polar angle by $\pi/4$ then both arms are stretched equally and no interference is produced. To codify this behaviour we define the antenna patterns, $F_{+,\times}(\theta, \phi, \psi)$ (equations 2.1 and 2.2). These modulate detector response as a function of angular position and polarization angle (Schutz 2011). They can be deduced from diagram 31 and the produced patterns are shown in figure 32.
These functions consider the detector as the origin of coordinates. Images on figure 32 show their many blind-spots, but once a collection of interferometers around the globe is operational these start to disappear. The combination of these antenna patterns is the ***N**etwork **A**ntenna **P**ower*, or NAP for short. What is interesting is that, for a sufficiently complex network, a measurement of NAP could allow for reconstruction of sky coordinates without many degeneracies (Álvares et al. 2021).

$$F_+(\theta, \phi, \psi) = \frac{1}{2}(1 + \cos(\theta)^2)\cos(2\phi)\cos(2\psi) - \cos(\theta)\sin(2\phi)\sin(2\psi) \tag{2.1}$$

$$F_\times(\theta, \phi, \psi) = \frac{1}{2}(1 + \cos(\theta)^2)\cos(2\phi)\sin(2\psi) + \cos(\theta)\sin(2\phi)\cos(2\psi) \tag{2.2}$$



(a) How a "+" polarized would be detected for $\psi = 0$.

(b) How a "×" polarized would be detected for $\psi = 0$.

(c) Average response to both polarizations: $\bar{F} = \sqrt{F_+^2 + F_\times^2}$.

Figure 32: Antenna patterns, with $|F_i|$ being represented by both radius and color. Ideal detector conditions correspond with $\vec{k}$ perpendicular to the plane formed by the detector's arms and there exists 4 blind-spots at ±45° and ±135° around that plane's axis. This is a visualisation of the aforementioned loss of sensitivity when both arms were being stretched equally, producing no interference. It is also relevant to note that average response does not depend on polarization angle. This last pattern is widely used and commonly referred to as "peanut-pattern".

B.2 Further training

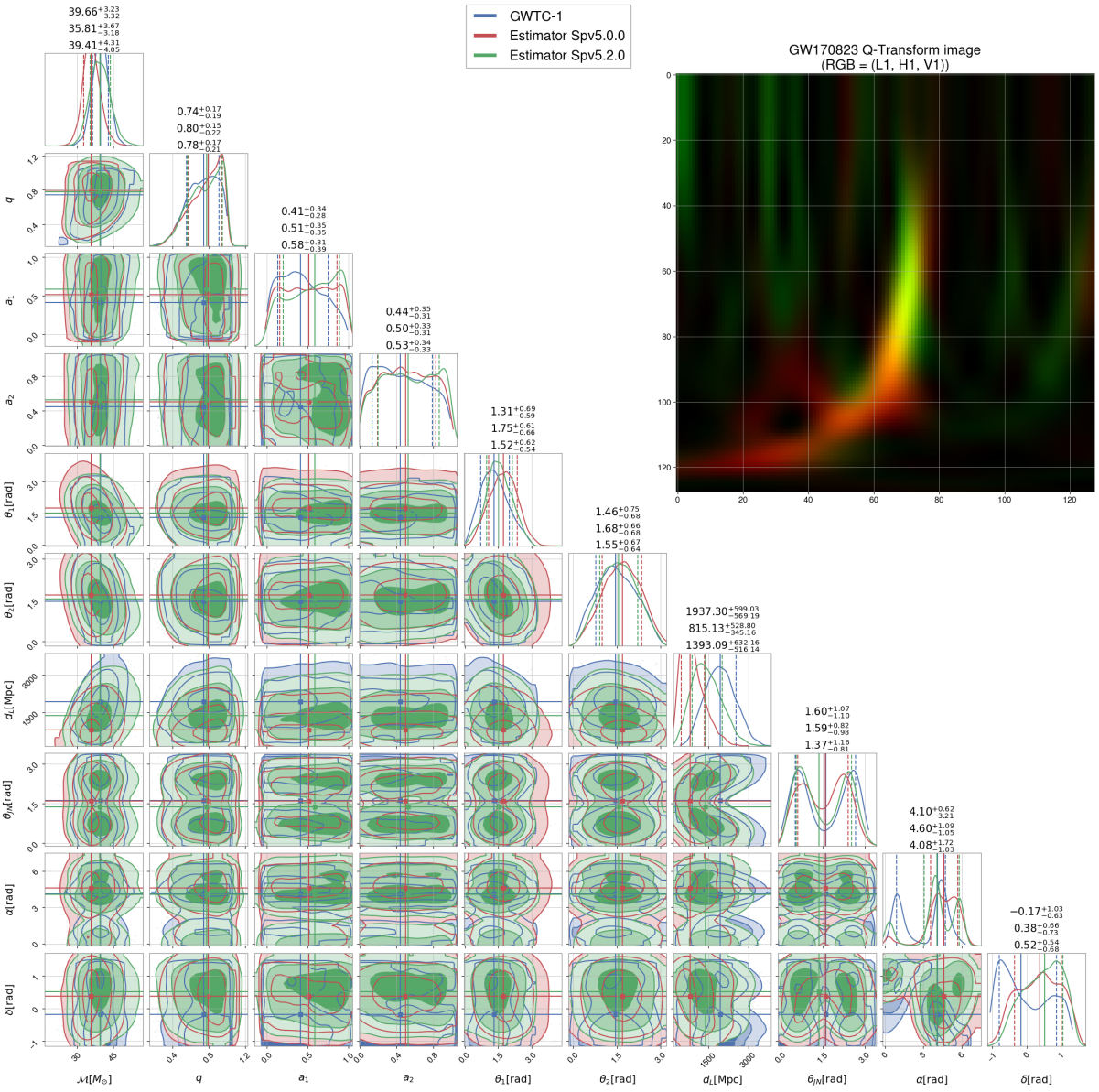


Figure 33: Comparison between my current flagship model (Spv5.0.0, 14 parameters), a similar model (Spv5.2.0) and GWTC-1 (B. P. Abbott et al. 2019[c]) MCMC samples on event GW170823. 10000 samples generated with the image shown used unprocessed as reference context for the first model, and normalized to $(\mu, \sigma) = (0, 1)$ for the second.

As pictured in figure 33, chirp mass and luminosity distance were the biggest improvements on this newer model, with marginal improvements on tilt angles. However, this was at the cost of spin magnitudes and declination worsening.
The second model was trained with 60k images (a 20% increase over the 50k of the previous), though as figure 34 seems to imply, we may be having a case of diminishing returns. This reinforces the idea that a better generation scheme may be preferable as opposed to ever larger datasets.

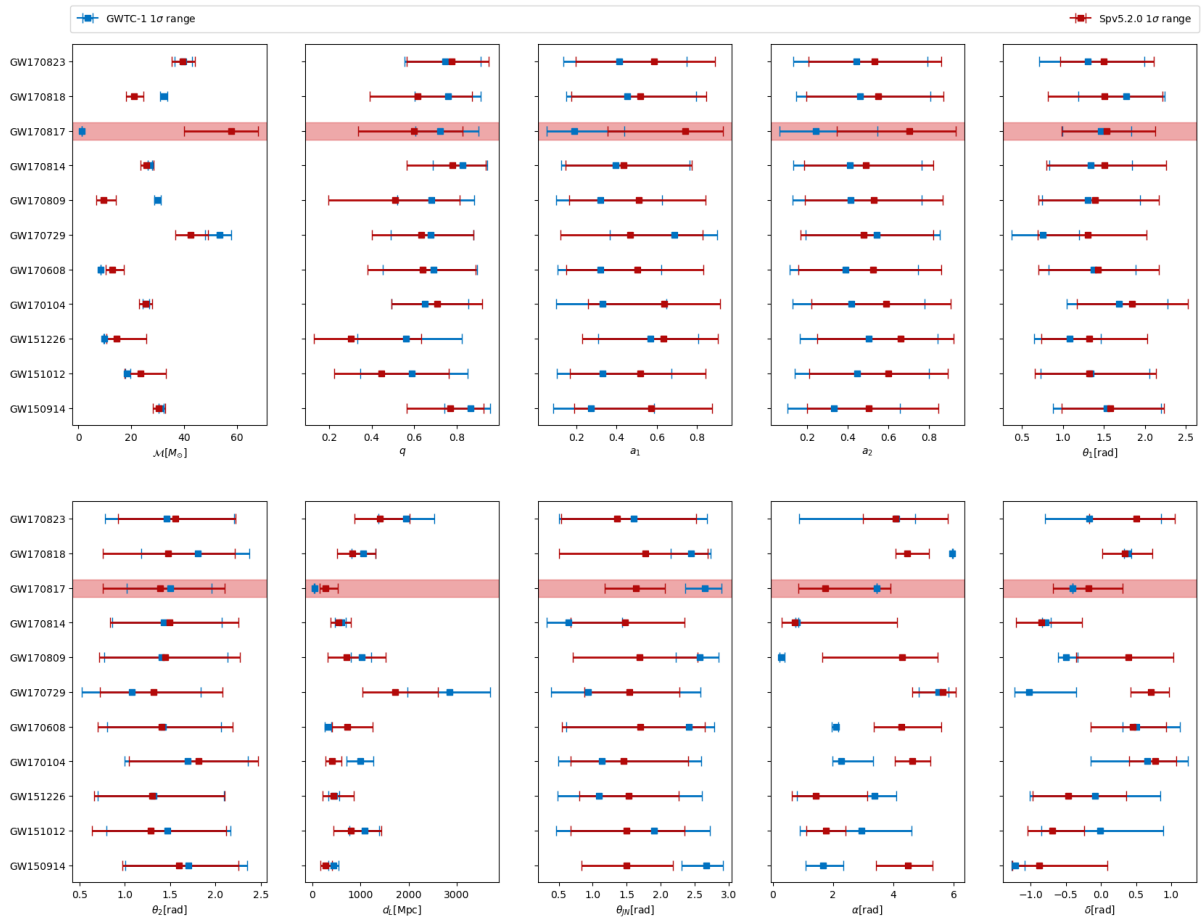


Figure 34: Comparison of 1 standard deviation ranges for all 11 events of GWTC-1 (B. P. Abbott et al. 2019[c]) for the 10 available parameters from publicly released samples.

Continuing our analysis of figure 33, figure 34 shows that improvement is marginal at best, with GW170823 being a best-case scenario. For that reason the main analysis presented in this work is that of the original model, which trained faster and needed no pre-processing of the context images.

Given that best results were, for the most part, being achieved on the mid-mass events ($\mathcal{M} \sim 30-50$) of GWTC-1, and those of the second observation campaign (O2) in particular, it was worth developing a dataset centered around these events. For this, a pool of three noise patches[12] of 500 seconds each was sampled uniformly, taking the PSD and using it for their respective interferometer, L1 and H1. 240 thousand images were used, for a training time of 6 hours. This was achieved by reducing resolution from 128 by 128 pixels to just 48 by 72..

[12] close to mergers GW170729 (t_{GPS}:1185366342), GW170818 (t_{GPS}: 1187058342) and GW170823 (t_{GPS}:1187490342)

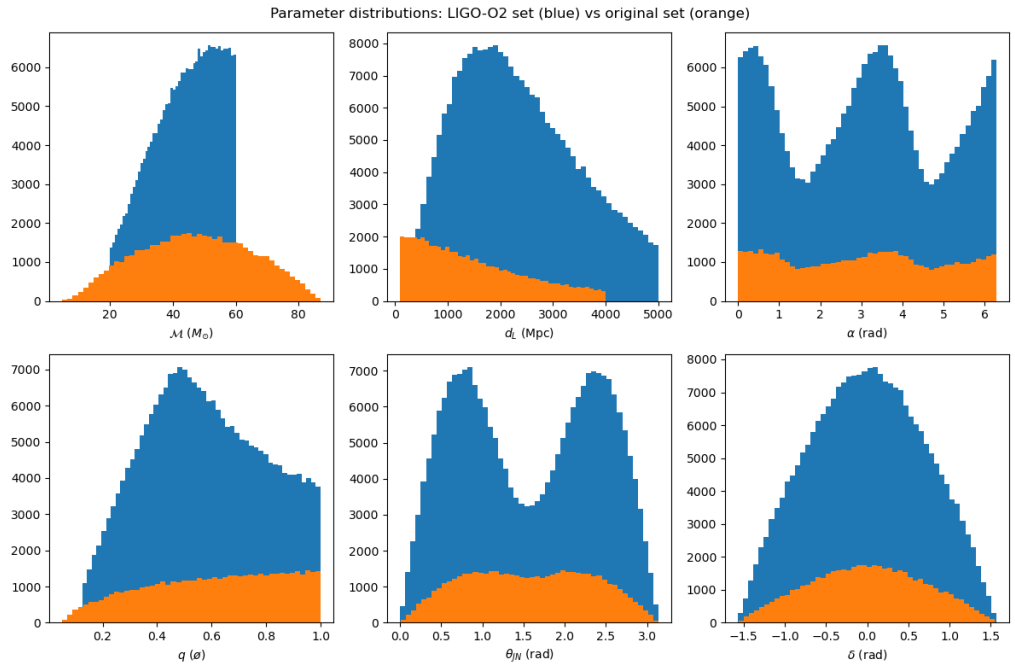


Figure 35: Comparison between selection of parameter distributions for the new O2-based dataset and the original. Counts represent number of injections, and ilustrate the increase in data for a comparable amount of memory.

Figure 35 puts into perspective the substantial growth in dataset size (as in number of images, not memory allocation) from the main results of this works to these next ones. Apart from the size difference, the new dataset is based on bilby's priors, explaining the bigger distance range and increased mass ratio lower cutoff. To maintain some consistency the prior is sampled, and if SNR falls below the minimum (still 5), it is resampled.

In spite of the missing parameters of figures 36 and 37 it is indeed a 14-parameter estimation. Considering that ϕ_{JL}, ϕ_{12}, ϕ and ψ tend to resemble reference data in subsequent catalogues, it is sensible to assume that they are acceptably matched, though a comparison with further catalogues would not make sense given the lack of Virgo data and O2 specific noise.

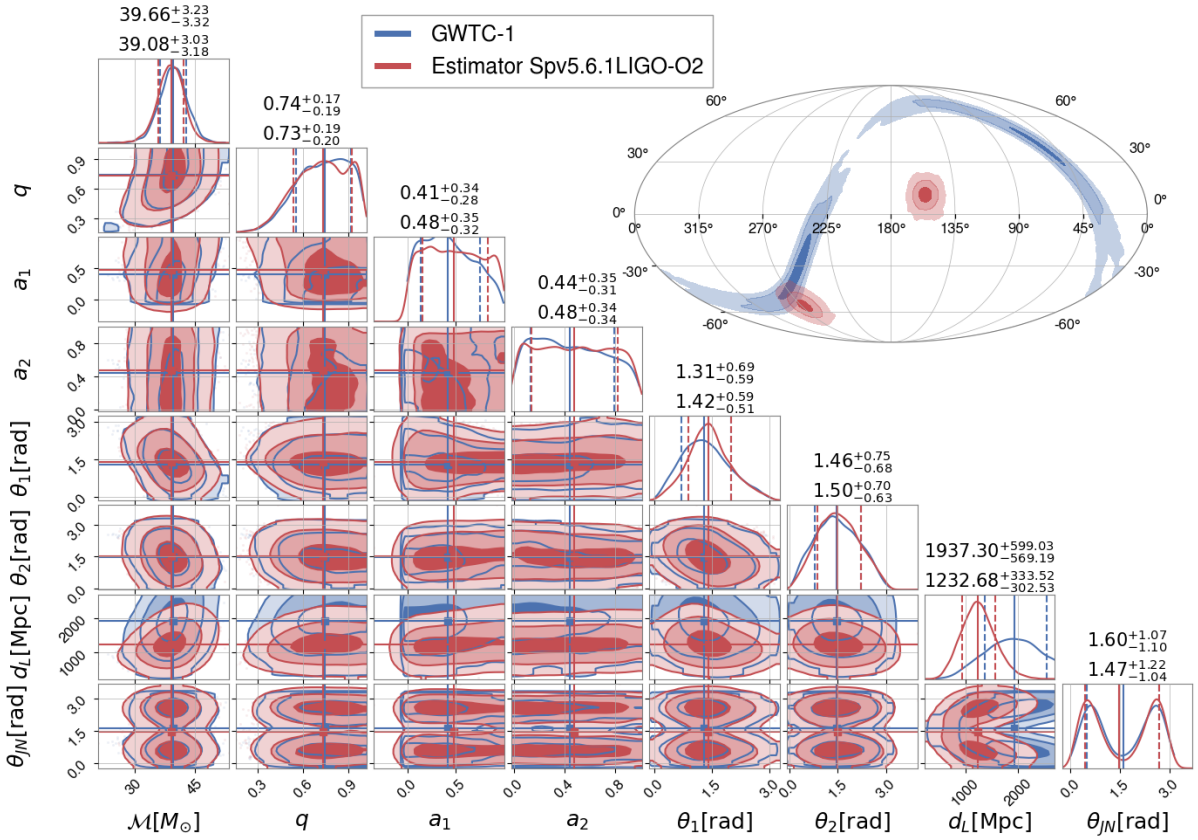


Figure 36: Comparison between samples of the GWTC-1 release of GW170823 and model Spv5.6.1LIGO-O2, taylored specifically to O2 events such as this one.

Figure 36 demonstrates that the model can learn degeneracies in position stemming from the training data consisting of just two detectors, L1 and H1 in this case. Nonetheless, they are still far away from reproducing the expected patterns and their accuracy, while convincing, still needs to improve.

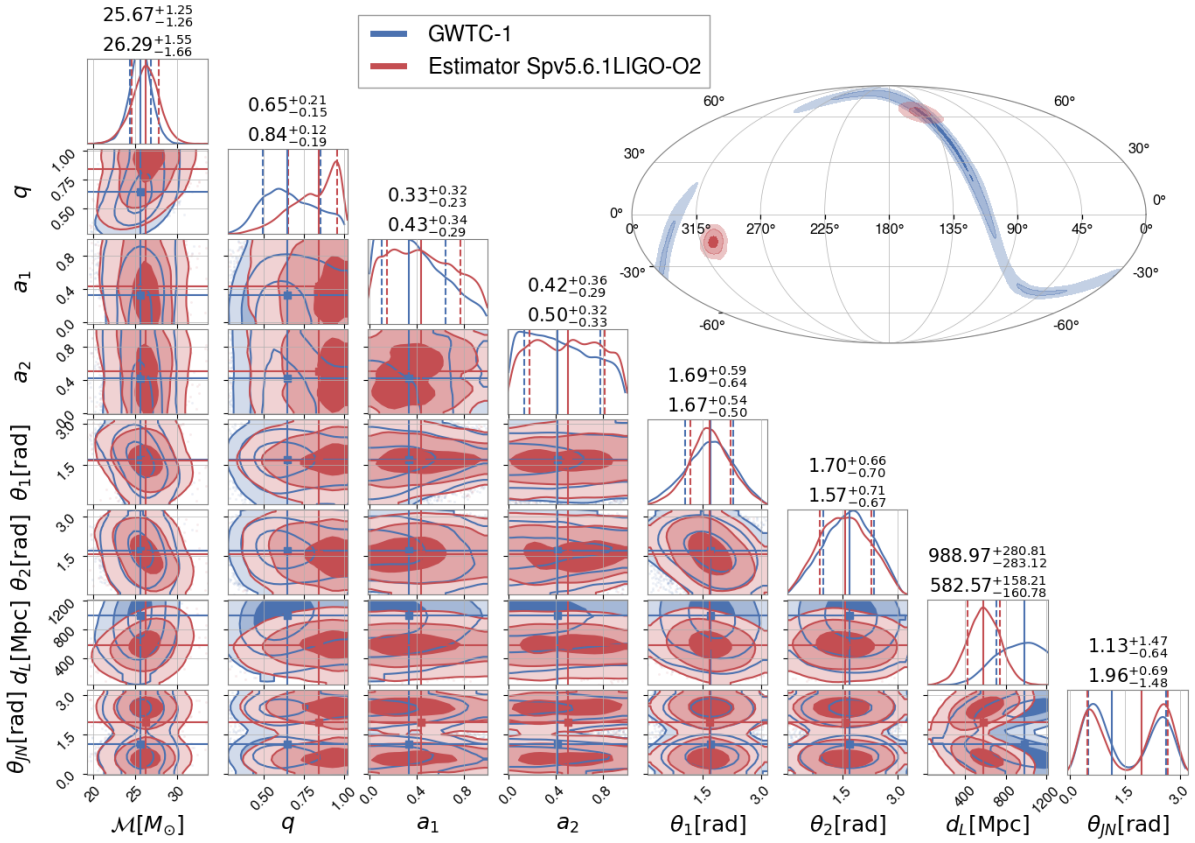


Figure 37: Comparison between samples of the GWTC-1 release of GW170104 and model Spv5.6.1LIGO-O2, taylored specifically to O2 events such as this one.

In figure 37 we see an even closer agreement with the density peak, though this model returns a mostly Gaussian patch, which may indicate that it needs a deeper flow (this particular model had 16 steps, double any previous 14-parameter model), requiring more powerful computational facilities.

C Software discussed in this work

B | D | G | L | N | P

B

Bilby Acronym for ***B**ayesian **I**nference **LiB**rar**Y***. Python library specialised in MCMC estimation (Ashton et al. 2019). Although designed for CBCs, it supports external models and samplers such as dynesty (Speagle 2020). A parallel implementation has since been developed (Smith et al. 2020). Docs . 9, 10, 13, 16, 37, 50, 54

D

Dingo Acronym for ***D**eep **IN**ference for **G**avitational wave **O**bservations*. Python library specialised in neural posterior estimation (Dax et al. 2021[b]; Dax et al. 2021[a]). Although designed for CBCs, its models could be trained on a variety of tasks. This work is meant to provide an image based implementation of NPE estimators as opposed to Dingo's timeseries approach. Docs . 9, 12, 13

G

gwpy Acronym for ***G**ravitational **W**ave **Py**thon*. Python library with the minimal functionality neccesary for strudying gravitational waves without excesive knowledge of signal analysis (Macleod et al. 2021). It also gives acces to the GWOSC servers. Docs . 61

L

LALSuite (indirectly) Acronym for ***LIGO A**lgorithm **L**ibrary **Suite***. Collection of GW-oriented utilities developed by the LIGO Collaboration over the last three decades (LIGO Scientific Collaboration et al. 2018). Through their Python and Octave interfaces (Wette 2020), it forms the backbone of all programs used in this work. Available only for GNU Linux y macOS operating systems. Docs . 14, 27

N

nflows Python library that implements a collection of the most common normalizing flow architectures (Durkan et al. 2020). My `Estimator` and `CBCEstimator` classes wrap its `Flow` class and the transforms discussed are constructed from theirs. This work uses an up-to date fork of the library. Repo . 12

numpy Main implementation of the multidimensional arrays on the python language. Its `ndarray` object set the standard for other similar data structures. Docs . 24, 57

P

pandas Data handling and analysis library for the python language. Focused on manipulating tables and series, its `DataFrame` and `Series` classes form the backbone of many o this work's utilities such as the `TrainSet` object discussed in 3.2. Docs . 24

PESummary Acronym for ***P**osterior **E**stimation **S**ummary*. Python software package for processing and visualizing data from any parameter estimation code (Hoy and Raymond 2021). My code relies on it to handle model samples by subclassing its `SamplesDict` object on its core module (or rather a copy of it, see this paragraph for more information). Docs . 1, 14, 21, 27, 39

PyCBC Acronym for ***Py**thon **C**ompact **B**inary **C**oalescence*. Python library centered around CBC-derived physics. It implements most approximant models (Nitz et al. 2023), as well as inference functionalities (Biwer et al. 2019) and its own search pipeline (Usman et al. 2016) Docs. 7, 10, 16

PyTorch Python port of the Torch machine learning library. It is one of the main development platforms for machine learning algorithms. Its main feature is the `tensor` class, not too dissimilar from numpy's `ndarray`, but with the possibility of storing gradient information, vital for backpropagation algorithms. Docs. 16, 22

D Glossary

A | B | C | G | K | L | M | N | O | P | Q | S | T | V | W

A

ASD Shorthand for ***A**mplitude **S**pectral **D**ensity*. Measures the amplitude of the signal as a function of frequency. Calculated by chaining overlapping Fourier transforms. Measured in h/Hz. In GW circles it is more common to use the PSD, related to h^2, but it can be just as useful. 60

B

Backpropagation Referring to the -efficient- calculation of the gradient of a loss function (in our case the Log-probability of the distribution) with respect to the weights of the machine-learning model. It is calculated by applying the chain rule to the subsequent layers of the model. The gradient is then used to compute a weight correction by an optimizer such as Stochastic Gradient Descent. 12

BBH Shorthand for ***B**inary **B**lack **H**oles*, it corresponds to a CBC composed by two black holes. Among CBCs detected, it is by far the most common with over 95% of detections over the first three catalogs. 1, 5, 7, 9, 29, 30, 37, 42, 45, 47, 58

BHNS Shorthand for ***B**lack **H**ole + **N**eutron **S**tar*. As the name would suggest, it reffers to the merger of a black hole and a neutron star. 5, 45, 47, 58

BNS Shorthand for ***B**inary **N**eutron **S**tar*, it describes a CBC composed by two black neutron stars. These events are key for the emerging field of multi-messenger astronomy. Detector sensitivity is usually measured not by the minimum h it can resolve but by the maximum distance at which it can detect these events according to equation 1.10. In particular, it enunciates the distance in Mpc at which the fusion of two neutron stars of 1.4 $M_\odot$ each could be detected with SNR = 8 (B. P. Abbott et al. 2020[c]). As for GW170817, Virgo observed the event despite operating at a shorter range, $\sim$ 30 Mpc, but with an SNR under the trigger's minimum. 5, 8, 9, 42, 43, 47, 58

C

CBC Shorthand for ***C**ompact **B**inary **C**oalescence*. They are the most (and almost only) studied GW source, involving the latter stages of a binary system of bodies denoted as compact (Black holes or neutron stars). It is classified into either BBH, BNS or BHNS depending on the exact composition of the binary. 5, 7, 8, 9, 10, 15, 28, 45, 47, 57, 58, 59, 61

CCSN Shorthand for ***C**ore **C**olapse **S**uper **N**ova*, or type II supernova. Ends the lifecycle of a star of $M \gtrsim 8M_\odot$ leaving behind either a neutron star or a black hole depending on the exact mass, creating heavy elements in the process. If their collapse is asymmetric (should be in most cases) they are expected to produce gravitational waves. 61

G

GraceDB Shorthand for ***Gra**vitational **C**andidate **E**vent **D**ata**b**ase*. Database maintained by LIGO for new GW candidate alerts. Web Page. 59

G-strain Unit-less observable defined as the quotient between change in length of an object (typically the detector's arms when a wave passes) and the object's rest-length. Usually denoted as h. Since it is indicative of the wave's amplitude, its intensity is given by h^2. 4, 5

GWOSC Shorthand for ***G**ravitational **W**ave **O**pen **S**cience **C**enter*. Access protal for released observatory data as well as annual workshops. An ideal starting point for delving into this field. Web Page. If, as is the case with this work, an active campaign is taking place one can stay up to date with event candidates through GraceDB. 47, 57, 59

GWTC Shorthand for ***G**ravitational **W**ave **T**ranscient **C**atalogue*. as the name suggests, it catalogues CBCs. Currently catalogues GWTC-1, 2.1 and 3 record ~90 events whose characteristics and (a limited selection of) parameters are available through GWOSC. Web Page. 13, 23, 28, 29, 35, 36, 37, 39, 40, 41, 42, 43, 45, 46, 47, 48, 52, 53

K

KAGRA Acronym for ***KA**mioaka **GRA**vity*. Gravitational wave detector located in the Kamioaka mine, Japan, as part of the Kamioaka observatory. Despite beginning operations on February of 2020 the COVID-19 pandemic forced its halting, deeply impacting sensibility goals. On May the 25th of 2023 it joined the **O4** campaign. Web page. 13, 59

L

LIGO Shorthand for ***L**aser **I**nterforemeter **G**ravitational-wave **O**bservatory*. GW observatory composed of two detectors in Livingston and Hanford, USA, as well as a future interferometer in Maharashtra, India. There technically was an 'initial LIGO' (iLIGO) in operation between 2002 and 2008 and the actual detectors form the 'advanced LIGO' or aLIGO, operative from 2015. Since iLIGO didn't ever detect waves I will be referring to aLIGO as LIGO throughout this work. Web page. 3, 7, 13, 15, 27, 47, 48, 57, 58, 59, 61

Log-probability As the name suggests, the logarithm of a probability. I will also refer to it as "logprob" throughout this work. The reason to use it instead of the probability itself is that addition is much simpler and less error prone than multiplication for computers (less risk of underflow). Also, the probability of multiple independent events is its product, which the logarithm maps to a sum, making it almost trivial to differentiate in comparison. 12, 58

LULinear (transform) Type of transform where the matrix of weights that takes input to output (hence linear) is decomposed as a product of a lower-triangular (L) and an upper-triangular (U) matrix. 12

LVK Shorthand for ***LIGO** **V**irgo **KAGRA** Collaboration*. Internationtal collaboration of the LIGO, Virgo and KAGRA observatories, which make up the network of cutting-edge detectors. 47

M

Markov chain Set of linked states whose probability of transition depend only on the current state. In other words, it is a memory-less chain. In this divulgation video Markov chains are explained exemplified with the `PageRank` algorithm, which powers most search engines. As a matter of fact, this precise glossary with the references that link its entries could be modelled as one of these. 60

Mass gap event CBC in which at least one of the components is within the inferior mass gap: the region of parameter space between 2 and 5 $M_\odot$ too massive to be a stable neutron star but to light to be a black hole formed by known mechanisms. 5, 43

Matched Filtering We denote as *matched filtering* the process correlating a template signal with another signal in order to detect the presence of the template (or a similar enough signal) in the second signal. It is the main signal detection technique on the field and combined with template banks they can give a rough estimation of the parameters of an event. 15, 61

MCMC Shorthand for ***M**arkov **C**hain **M**onte **C**arlo*. Algorithm family that enables sampling of an unknown probability density -of which we know some characteristics-. These are based on multiple random walkers over a Markov chain designed so that its stationary distribution is the desired distribution. These algorithms improve upon traditional Monte Carlo since samples are not fully independent, but instead depend on the sample that came immediately before. 1, 10, 12, 27, 39, 40, 41, 47, 48, 52, 57

N

NPE Shorthand for ***N**eural **P**osterior **E**stimation*. Although a precise definition is hard to come by, given the bottom-up nature of research and the recency of this topic, NPE tends to refer to the use of neural algorithms (neural networks and derived technologies, such as normalizing flows) to perform Bayesian statistics and in particular, posterior calculations (see equation 2.1).
Deep Tempest constructs likelihood-free models. That is, posterior distributions are estimated without the use of Bayes' theorem, as the models are taught to relate context data to parameter posteriors through supervised learning. 1, 10, 12, 14, 34, 57

O

OOP Shorthand for ***O**bject-**O**riented **P**rograming*. Programing paradigm in which objects -classes- contain not just data, but also their own functions -methods-. Defined as opposed to functional programing in which functions and data are kept separate.
Python's implementation relies on so-called "dunder" methods, as in `__method__`. Special methods that define the interactions between the object and the outside world. Objects in python have items and attributes. What differentiates them is the way they are accessed. Items are accessed as "`object['item_key']`", which calls the `__getitem__` method under the hood, while attributes have the syntax "`object.attribute`", handled by `__getattr__`. If and object -O- has a callable attribute -a- (that is, the attribute is itself an object with a defined `__call__` method) then we refer to -a- as a method of -O- and we can use it as:
"`results = O.a(arguments)`". 24

P

PSD Shorthand for ***P**ower **S**pectral **D**ensity*. Measures the power of the signal as a function of frequency (note that in signal analysis we refer use *power* to denote the amplitude's square, not to be confused with the physical definition of power). In gravitational wave astrophysics it is commonly expressed in h^2/Hz even if aware that h is unitless so as to differentiate it from the ASD, which is measured in h/Hz. If one is to take a sufficiently long signal it would give us a noise profile of a detector. It is calculated with the following limit: $\lim_{T\to\infty} |\mathcal{F}[x](f)|^2/T$, with $\mathcal{F}$ as the Fourier transform of the signal of period T. 53, 58, 61

Q

Q transform Relative to the Fourier transform which adds a "quality factor" Q that controls the size of frequency bins, making it ideal to cover vast ranges. It resembles a spectrogram in the sense that it graphs the frequencies that form a signal over time. Before the development on gravitational astrophysics it was a common tool for sound engineers and musicians, which further cements the analogy between gravitational waves and sound. 7, 15, 42, 45

S

SNR Shorthand for ***S**ignal to **N**oise **R**atio*, signal analysis terminology. As the name implies it expreses the level of signal over noise background as a timeseries. It is achived through a convolution of the signal you expect to find and the data. If there is a match a peak will appear on the resulting SNR series. 15, 16, 28, 39, 48, 54, 58, 61

supervised learning Machine learning paradigm in which data is labelled (as opposed to unsupervised learning and a whole spectrum within). In the case at hand, data is labelled by the values of the injection's parameters. 60

T

Template bank We denote as template banks the collection of template signals used in a matched filtering pipeline. Given the number of templates is finite the parameter space has to sampled intelligently, covering as much volume with as low number of signals as possible while avoiding the introduction of population distribution artifacts A multitude of templates may trigger the detection, giving an early estimation of event parameters. For example, if there was a circular gap of center $(3, 3)M_\odot$ and a radius of 2 $M_\odot$ on the bank catalogues and their subsequent study would confirm the existence of the lower mass gap because if an event on that range occurred we would miss it. They are generally constructed in such a way that any given signal is close enough to a template to recover 97% of its SNR (Dhurkunde et al. 2022; Mukherjee et al. 2021) . 59

V

Virgo Gravitational wave observatory on the outskirts of Pisa, Italy. It is smaller than the two LIGO interferometers, and also less sensitive, so it is expected for the blue channel of the event images to be more noisy and the chirp signature less obvious. Despite this, the flows should be able to pick up valuable information from its strain data. Web page . 13, 15, 27, 40, 47, 54, 58, 59

W

Waveform The shape of the signal one expects from the detector. Generaly reffered to a timeseries but can also be given as a frequency series. A CBC produces a chrip-type waveform, but other sources (even transitoty ones such as an asymmetric CCSN) can have drastically different waveforms. 7, 14, 15, 16

Weak (field) limit When considering systems of low gravitational interaction one can separate the metric into a background metric, oftentimes Minkowski spacetime, and a perturbation as $g_{\mu\nu} \approx \eta_{\mu\nu} + h_{\mu\nu}$. What makes this formalism so powefull is the fact that the field equations become lineal in $h_{\mu\nu}$. 3

Whitening We denote as *whitening* the process of normilizing a series -whether time or frequency- such that it maintains the same power throughout all its frequencies. gwpy's `TimeSeries` object implements a whitening method. Nonetheless one can manually obtain one by dividing the series -in frecuency space- over the square root of its PSD. 15

References

LVK Collaboration (Aasi, J et al.)
— (Mar. 2015). "Advanced LIGO". In: *Classical and Quantum Gravity* 32.7, p. 074001. DOI.
LVK Collaboration (Abadie, J. et al.)
— (2010). "Calibration of the LIGO gravitational wave detectors in the fifth science run". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 624.1, pp. 223–240. ISSN: 0168-9002. DOI. URL.
LVK Collaboration (Abbott, B. P. et al.)
— (Feb. 2016[a]). "Observation of Gravitational Waves from a Binary Black Hole Merger". In: *Phys. Rev. Lett.* 116 (6), p. 061102. DOI. URL.
— (June 2016[b]). "Properties of the Binary Black Hole Merger GW150914". In: *Physical Review Letters* 116.24. DOI.
— (Oct. 2016[c]). "The basic physics of the binary black hole merger GW150914". In: *Annalen der Physik* 529.1-2, p. 1600209. DOI.
— (Oct. 2017). "GW170817: Observation of Gravitational Waves from a Binary Neutron Star Inspiral". In: *Phys. Rev. Lett.* 119 (16), p. 161101. DOI. URL.
— (May 2018). "Search for Tensor, Vector, and Scalar Polarizations in the Stochastic Gravitational-Wave Background". In: *Physical Review Letters* 120.20. DOI.
— (Jan. 2019[a]). "Properties of the Binary Neutron Star Merger GW170817". In: *Physical Review X* 9.1. DOI.
— (July 2019[b]). "Tests of General Relativity with GW170817". In: *Phys. Rev. Lett.* 123 (1), p. 011102. DOI. URL.
— (Sept. 2019[c]). "GWTC-1: A Gravitational-Wave Transient Catalog of Compact Binary Mergers Observed by LIGO and Virgo during the First and Second Observing Runs". In: *Phys. Rev. X* 9 (3), p. 031040. DOI. URL.
— (Feb. 2020[a]). "A guide to LIGO–Virgo detector noise and extraction of transient gravitational-wave signals". In: *Classical and Quantum Gravity* 37.5, p. 055002. DOI.
— (Mar. 2020[b]). "GW190425: Observation of a Compact Binary Coalescence with Total Mass~3.4 $M_\odot$". In: *The Astrophysical Journal Letters* 892.1, p. L3. DOI.
— (July 2020[c]). "Prospects for observing and localizing gravitational-wave transients with Advanced LIGO, Advanced Virgo and KAGRA". In: *Living Reviews in Relativity* 23.1. DOI.
LVK Collaboration (Abbott, R. et al.)
— (June 2020[a]). "GW190814: Gravitational Waves from the Coalescence of a 23 Solar Mass Black Hole with a 2.6 Solar Mass Compact Object". In: *The Astrophysical Journal Letters* 896.2, p. L44. DOI. URL.
— (July 2020[b]). "GW190521: A Binary Black Hole Merger with a Total Mass of 150$M_\odot$". In: *Phys. Rev. Lett.* 125 (10), p. 101102. DOI. URL.
— (June 2021[a]). "Observation of Gravitational Waves from Two Neutron Star–Black Hole Coalescences". In: *The Astrophysical Journal Letters* 915.1, p. L5. DOI.
— (Dec. 2021[b]). "Tests of General Relativity with GWTC-3". In: arXiv [gr-qc].
— (Dec. 2023). "GWTC-3: Compact Binary Coalescences Observed by LIGO and Virgo during the Second Part of the Third Observing Run". In: *Phys. Rev. X* 13 (4), p. 041039. DOI. URL.
— (Jan. 2024). "GWTC-2.1: Deep extended catalog of compact binary coalescences observed by LIGO and Virgo during the first half of the third observing run". In: Phys. Rev. D 109.2, 022001, p. 022001. DOI. arXiv [gr-qc].
Álvares, João D et al. (July 2021). "Exploring gravitational-wave detection and parameter inference using deep learning methods". In: *Classical and Quantum Gravity* 38.15, p. 155010. DOI.
Andersson, Nils (Nov. 2019). *Gravitational-Wave Astronomy: Exploring the Dark Side of the Universe*. Oxford University Press. ISBN: 9780198568032. DOI. URL.
Ashton, Gregory et al. (Apr. 2019). "BILBY: A User-friendly Bayesian Inference Library for Gravitational-wave Astronomy". In: *Astrophysical Journal, Supplement* 241.2, 27, p. 27. DOI. arXiv [astro-ph.IM].
Berry, Christopher P. L. et al. (May 2015). "Parameter Estimation for Binary Neutron-star Coalescences with Realistic Noise during the Advanced LIGO Era". In: ApJ 804.2, 114, p. 114. DOI. arXiv [astro-ph.HE].

Biwer, C. M. et al. (Jan. 2019). "PyCBC Inference: A Python-based Parameter Estimation Toolkit for Compact Binary Coalescence Signals". In: *Publications of the Astronomical Society of the Pacific* 131.996, p. 024503. DOI. URL.

Blanchet, Luc (Feb. 2014). "Gravitational Radiation from Post-Newtonian Sources and Inspiralling Compact Binaries". In: *Living Reviews in Relativity* 17.1. DOI.

Bohé, Alejandro et al. (May 2016). *PhenomPv2 – technical notes for the LAL implementation*. URL.

Butter, Anja et al. (2019). "The Machine Learning landscape of top taggers". In: *SciPost Phys.* 7. Ed. by Gregor Kasieczka and Tilman Plehn, p. 014. DOI. arXiv [hep-ph].

Carroll, Sean M. (2014). *Spacetime and Geometry: An Introduction to General Relativity*. Pearson. Chap. 7.

Christensen, Nelson and Renate Meyer (Apr. 2022). "Parameter estimation with gravitational waves". In: *Rev. Mod. Phys.* 94 (2), p. 025001. DOI. URL.

Coccia, E. et al., eds. (2020). *Proceedings, 200th Course of Enrico Fermi School of Physics: Gravitational Waves and Cosmology (GW-COSM): Varenna (Lake Como) (Lecco), Italy, July 3-12, 2017*.

Dax, Maximilian et al. (Nov. 2021[a]). "Group equivariant neural posterior estimation". In: arXiv [cs.LG].

— (Dec. 2021[b]). "Real-Time Gravitational Wave Science with Neural Posterior Estimation". In: *Phys. Rev. Lett.* 127 (24), p. 241103. DOI. URL.

De Santi, Federico et al. (May 2024). "Deep learning to detect gravitational waves from binary close encounters: Fast parameter estimation using normalizing flows". In: Phys. Rev. D 109.10, 102004, p. 102004. DOI. arXiv [gr-qc].

Dhurkunde, Rahul et al. (May 2022). "Hierarchical approach to matched filtering using a reduced basis". In: *Phys. Rev. D* 105 (10), p. 103001. DOI. URL.

Durkan, Conor et al. (June 2019). "Neural Spline Flows". In: *arXiv e-prints*, arXiv:1906.04032, arXiv:1906.04032. DOI. arXiv [stat.ML].

— (Nov. 2020). *nflows: normalizing flows in PyTorch*. Version v0.14. DOI. URL.

Ezquiaga, Jose María and Miguel Zumalacárregui (Dec. 2018). "Dark Energy in Light of Multi-Messenger Gravitational-Wave Astronomy". In: *Frontiers in Astronomy and Space Sciences* 5. DOI.

Field, Scott E. et al. (July 2014). "Fast Prediction and Evaluation of Gravitational Waveforms Using Surrogate Models". In: *Physical Review X* 4.3, 031006, p. 031006. DOI. arXiv [gr-qc].

Filipović, Miroslav D and Nicholas F H Tothill, eds. (2021). *Multimessenger Astronomy in Practice*. 2514-3433. IOP Publishing. Chap. 9. ISBN: 978-0-7503-2344-4. DOI. URL.

Gal, Yarin and Zoubin Ghahramani (June 2015). "Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference". In: *arXiv e-prints*, arXiv:1506.02158, arXiv:1506.02158. DOI. arXiv [stat.ML].

— (June 2016). "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of The 33rd International Conference on Machine Learning*. Vol. 48. PMLR, pp. 1050–1059. URL.

Gao, Le et al. (2023). "The Application of ResNet-34 Model Integrating Transfer Learning in the Recognition and Classification of Overseas Chinese Frescoes". In: *Electronics* 12.17. ISSN: 2079-9292. DOI. URL.

García-Quirós, Cecilio et al. (July 2020). "Multimode frequency-domain model for the gravitational wave signal from nonprecessing black-hole binaries". In: Phys. Rev. D 102.6, 064002, p. 064002. DOI. arXiv [gr-qc].

Germain, Mathieu et al. (July 2015). "MADE: Masked Autoencoder for Distribution Estimation". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 881–889. URL.

Green, Stephen R and Jonathan Gair (June 2021). "Complete parameter inference for GW150914 using deep learning". In: *Machine Learning: Science and Technology* 2.3, 03LT01. DOI. URL.

Green, Stephen R., Christine Simpson, et al. (Nov. 2020). "Gravitational-wave parameter estimation with autoregressive neural network flows". In: *Physical Review D* 102.10. ISSN: 2470-0029. DOI. URL.

Guidry, Mike (2019). *Modern General Relativity: Black Holes, Gravitational Waves, and Cosmology*. Cambridge University Press. Chap. 22 to 24. DOI.

Hall, Evan D. (2022). "Cosmic Explorer: A Next-Generation Ground-Based Gravitational-Wave Observatory". In: *Galaxies* 10.4. ISSN: 2075-4434. DOI. URL.

Hannam, Mark et al. (Oct. 2014). "Simple Model of Complete Precessing Black-Hole-Binary Gravitational Waveforms". In: *Physical Review Letters* 113.15. DOI.

Hastings, W. K. (Apr. 1970). "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1, pp. 97–109. ISSN: 0006-3444. DOI. eprint. URL.

He, Kaiming et al. (2015). *Deep Residual Learning for Image Recognition*. arXiv [cs.CV].

Heaviside, O. (1893). "A Gravitational and Electromagnetic Analogy". In: *The Electrician*. Reproduced by Oleg D. Jefimenko, updated mathematical notation.

Hoy, Charlie and Vivien Raymond (2021). "PESummary: The code agnostic Parameter Estimation Summary page builder". In: *SoftwareX* 15, p. 100765. ISSN: 2352-7110. DOI. URL.

Huerta, Eliu et al. (Jan. 2018). "Detection and characterization of eccentric compact binary coalescence at the interface of numerical relativity, analytical relativity and machine learning". In: *APS April Meeting Abstracts*. Vol. 2018. APS Meeting Abstracts, S13.006, S13.006.

Khan, Sebastian, Katerina Chatziioannou, et al. (July 2019). "Phenomenological model for the gravitational-wave signal from precessing binary black holes with two-spin effects". In: *Physical Review D* 100.2. DOI.

Khan, Sebastian, Sascha Husa, et al. (Feb. 2016). "Frequency-domain gravitational waves from nonprecessing black-hole binaries. II. A phenomenological model for the advanced detector era". In: *Physical Review D* 93.4. DOI.

Kolmus, Alex et al. (2024). *Tuning neural posterior estimation for gravitational wave inference*. arXiv [astro-ph.IM].

Langendorff, Jurriaan et al. (Apr. 2023). "Normalizing Flows as an Avenue to Studying Overlapping Gravitational Wave Signals". In: *Phys. Rev. Lett.* 130 (17), p. 171402. DOI. URL.

Lee, Jaewook et al. (2015). "Metamodel for Efficient Estimation of Capacity-Fade Uncertainty in Li-Ion Batteries for Electric Vehicles". In: *Energies* 8.6, pp. 5538–5554. ISSN: 1996-1073. DOI. URL.

Leyde, Konstantin et al. (Mar. 2024). "Gravitational wave populations and cosmology with neural posterior estimation". In: *Physical Review D* 109.6. ISSN: 2470-0029. DOI. URL.

LVK Collaboration (LIGO Scientific Collaboration et al.)

— (2018). *LVK Algorithm Library - LALSuite*. free software (GPL). DOI.

Macleod, D. M. et al. (2021). "GWpy: A Python package for gravitational-wave astrophysics". In: *SoftwareX* 13, p. 100657. ISSN: 2352-7110. DOI. URL.

Metropolis, Nicholas et al. (June 1953). "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6, pp. 1087–1092. ISSN: 0021-9606. DOI. eprint. URL.

Mukherjee, Debnandini et al. (Apr. 2021). "Template bank for spinning compact binary mergers in the second observation run of Advanced LIGO and the first observation run of Advanced Virgo". In: *Phys. Rev. D* 103 (8), p. 084047. DOI. URL.

Müller, Thomas et al. (2019). *Neural Importance Sampling*. arXiv [cs.LG].

Nitz, A. H. et al. (Jan. 2023). *gwastro/pycbc: v2.0.6 release of PyCBC*. Version v2.0.6. DOI. URL.

Papamakarios, George, Eric Nalisnick, et al. (2021). "Normalizing Flows for Probabilistic Modeling and Inference". In: *Journal of Machine Learning Research* 22.57, pp. 1–64. URL.

Papamakarios, George, Theo Pavlakou, et al. (2017). "Masked Autoregressive Flow for Density Estimation". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL.

Peters, P. C. (Nov. 1964). "Gravitational Radiation and the Motion of Two Point Masses". In: *Phys. Rev.* 136 (4B), B1224–B1232. DOI. URL.

Planck Collaboration (Planck Collaboration et al.)

— (Sept. 2016). "Planck 2015 results. XIII. Cosmological parameters". In: A&A 594, A13, A13. DOI. arXiv [astro-ph.CO].

— (Sept. 2020). "Planck 2018 results - VI. Cosmological parameters". In: A&A 641, A6. DOI. URL.

Poincaré, H. (1905). "Sur la dynamique de l'électron". In: URL.

Pompili, Lorenzo et al. (Mar. 2023). *Laying the foundation of the effective-one-body waveform models SEOBNRv5: improved accuracy and efficiency for spinning non-precessing binary black holes*. arXiv [gr-qc].

Pratten, Geraint et al. (May 2021). "Computationally efficient models for the dominant and subdominant harmonic modes of precessing binary black holes". In: Phys. Rev. D 103.10, 104056, p. 104056. DOI. arXiv [gr-qc].

Ramos-Buades, Antoni et al. (June 2021). "IMRPhenomXE: First phenomenological waveform model for eccentric binary black holes with aligned spins". In: *11th Iberian gravitational wave meeting*. URL.

Rezende, Edmar et al. (Dec. 2017). "Malicious Software Classification Using Transfer Learning of ResNet-50 Deep Neural Network". In: DOI.

Romero-Shaw, I M et al. (Sept. 2020). "Bayesian inference for compact binary coalescences with bilby: validation and application to the first LIGO–Virgo gravitational-wave transient catalogue". In: *Monthly Notices of the Royal Astronomical Society* 499.3, pp. 3295–3319. ISSN: 0035-8711. DOI. eprint. URL.

Romero-Shaw, Isobel et al. (Nov. 2021). "Signs of Eccentricity in Two Gravitational-wave Signals May Indicate a Subpopulation of Dynamically Assembled Binary Black Holes". In: *The Astrophysical Journal Letters* 921.2, p. L31. DOI. URL.

Ruhe, David et al. (Nov. 2022). "Normalizing Flows for Hierarchical Bayesian Analysis: A Gravitational Wave Population Study". In: *arXiv e-prints*, arXiv:2211.09008, arXiv:2211.09008. DOI. arXiv [astro-ph.IM].

Russakovsky, Olga et al. (2015). *ImageNet Large Scale Visual Recognition Challenge*. arXiv [cs.CV].

Saulson, Peter R. (2013). "Gravitational wave detection: Principles and practice". In: *Comptes Rendus Physique* 14.4. Gravitational waves / Ondes gravitationnelles, pp. 288–305. ISSN: 1631-0705. DOI. URL.

Schutz, B. F. (May 2011). "Networks of gravitational wave detectors and three figures of merit". In: *Classical and Quantum Gravity* 28.12, p. 125023. DOI.

Singer, Leo P. and Larry R. Price (Jan. 2016). "Rapid Bayesian position reconstruction for gravitational-wave transients". In: Phys. Rev. D 93.2, 024013, p. 024013. DOI. arXiv [gr-qc].

Smith, Rory J E et al. (Aug. 2020). "Massively parallel Bayesian inference for transient gravitational-wave astronomy". In: *Monthly Notices of the Royal Astronomical Society* 498.3, pp. 4492–4502. ISSN: 0035-8711. DOI. eprint. URL.

Soni, S. et al. (Oct. 2021). "Discovering features in gravitational-wave data through detector characterization, citizen science and machine learning". In: *Classical and Quantum Gravity* 38.19, 195016, p. 195016. DOI. arXiv [gr-qc].

Speagle, Joshua S. (Apr. 2020). "DYNESTY: a dynamic nested sampling package for estimating Bayesian posteriors and evidences". In: *Monthly Notices of the RAS* 493.3, pp. 3132–3158. DOI. arXiv [astro-ph.IM].

Taracchini, Andrea et al. (Mar. 2014). "Effective-one-body model for black-hole binaries with generic mass ratios and spins". In: *Physical Review D* 89.6. ISSN: 1550-2368. DOI. URL.

Tiglio, Manuel and Aarón Villanueva (Dec. 2022). "Reduced order and surrogate models for gravitational waves". In: *Living Reviews in Relativity* 25.1, 2, p. 2. DOI. arXiv [gr-qc].

Usman, Samantha A et al. (Oct. 2016). "The PyCBC search for gravitational waves from compact binary coalescence". In: *Classical and Quantum Gravity* 33.21, p. 215004. DOI. URL.

Vajente, Gabriele et al. (2019). "Chapter Three - Precision interferometry for gravitational wave detection: Current status and future trends". In: Advances In Atomic, Molecular, and Optical Physics 68. Ed. by Louis F. Dimauro et al., pp. 75–148. ISSN: 1049-250X. DOI. URL.

Veitch, J. et al. (Feb. 2015). "Parameter estimation for compact binaries with ground-based gravitational-wave observations using the LALInference software library". In: *Phys. Rev. D* 91 (4), p. 042003. DOI. URL.

Wagner, Katelyn J. and Richard O'Shaughnessy (2024). *Parameter Estimation for Low-Mass Eccentric Black Hole Binaries*. arXiv [gr-qc].

Weinberg, Steven (1972). *Gravitation and Cosmology*. Wiley. Chap. 10.

Wette, Karl (2020). "SWIGLAL: Python and Octave interfaces to the LALSuite gravitational-wave data analysis libraries". In: *SoftwareX* 12, p. 100634. DOI.

Will, Clifford M (July 2018). "Solar system versus gravitational-wave bounds on the graviton mass". In: *Classical and Quantum Gravity* 35.17, 17LT01. DOI.