Gravitational-wave inference at GPU speed: A `bilby`-like nested sampling kernel within `blackjax-ns`

Metha Prathaban,[2,3]★ David Yallup,[1,2] James Alvey,[1,2] Ming Yang,[1] Will Templeton,[1] and Will Handley[1,2]
[1]*Institute of Astronomy, University of Cambridge, Cambridge, CB3 0HA, UK*
[2]*Kavli Institute for Cosmology, University of Cambridge, Cambridge, CB3 0EZ, UK*
[3]*Department of Physics, University of Cambridge, Cambridge, CB3 0HE, UK*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

We present a GPU-accelerated implementation of the gravitational-wave Bayesian inference pipeline for parameter estimation and model comparison. Specifically, we implement the 'acceptance-walk' sampling method, a cornerstone algorithm for gravitational-wave inference within the `bilby` and `dynesty` framework. By integrating this trusted kernel with the vectorized `blackjax-ns` framework, we achieve typical speedups of 20-40x for aligned spin binary black hole analyses, while recovering posteriors and evidences that are statistically identical to the original CPU implementation. This faithful re-implementation of a community-standard algorithm establishes a foundational benchmark for gravitational-wave inference. It quantifies the performance gains attributable solely to the architectural shift to GPUs, creating a vital reference against which future parallel sampling algorithms can be rigorously assessed. This allows for a clear distinction between algorithmic innovation and the inherent speedup from hardware. Our work provides a validated community tool for performing GPU-accelerated nested sampling in gravitational-wave data analyses.

1 INTRODUCTION

The era of gravitational-wave (GW) astronomy, initiated by the landmark detections at the Laser Interferometer Gravitational-Wave Observatory (LIGO), Virgo and now KAGRA, has revolutionized our view of the cosmos (Abbott et al. 2016, 2017a, 2019, 2024, 2023b,a, 2021, 2017b). Extracting scientific insights from the data, from measuring the masses and spins of binary black holes to performing precision tests of general relativity, relies heavily on the framework of Bayesian inference (Thrane & Talbot 2019). This allows for the estimation of posteriors on source parameters (parameter estimation) and the comparison of competing physical models (model selection).

The process of Bayesian inference in GW astronomy is, however, computationally demanding. Realistic theoretical models for the GW waveform can be complex, and the stochastic sampling algorithms required to explore the high-dimensional parameter space can require millions of likelihood evaluations per analysis (Abbott et al. 2020b). There are several community-standard software tools for performing GW inference, such as `pycbc` (Biwer et al. 2019) and `lal` (Veitch et al. 2015). Here, we focus on the inference library `bilby` (Ashton et al. 2019) paired with a custom implementation of the nested sampler `dynesty` (Speagle 2020), which has proven to be a robust and highly effective framework, tuned for the specific needs of GW posteriors. However, this framework is predominantly executed on central processing units (CPUs), making individual analyses costly and creating a computational bottleneck. This challenge is set to become acute with the increased data volumes from future observing runs (The LIGO Scientific Collaboration et al. 2015; Acernese et al. 2014; Abbott et al. 2020a) and the advent of next-generation observatories, such as the Einstein Telescope (Branchesi et al. 2023), which promise unprecedented sensitivity and detection volumes (Hu & Veitch 2024).

In response to this challenge, the GW community has begun to leverage the immense parallel processing power of graphics processing units (GPUs). Pioneering work in this domain, such as the `jimgw` codebase (Wong et al. 2023a), has successfully implemented the GPU-accelerated Markov Chain Monte Carlo (MCMC) sampler `flowMC` (Wong et al. 2023b), paired with GPU implementations of waveform models provided by the `ripple` library (Edwards et al. 2024). This work has demonstrated that substantial, orders-of-magnitude speedups in runtime are achievable for GW parameter estimation. This success in accelerating parameter estimation motivates the development of complementary GPU-native algorithms to accelerate other key inference tasks, such as model selection.

In this paper, we introduce a GPU-accelerated nested sampling (NS) algorithm for gravitational-wave data analysis. Our method builds upon the trusted 'acceptance-walk' sampling method used in the community-standard `bilby` and `dynesty` framework (Ashton et al. 2019). We leverage the `blackjax` (Bradbury et al. 2018; Cabezas et al. 2024) implementation of nested sampling (Yallup et al. 2025a), a reformulation of the core nested sampling algorithm for massive GPU parallelization. In place of the default method for particle evolution recommended in Yallup et al. (2025a), we develop a custom kernel that implements the 'acceptance-walk' method. This ensures our sampler's logic is almost identical to the `bilby` and `dynesty` implementation, with differences primarily related to parallelization. This approach offers `bilby` users a direct path to GPU-accelerated inference for the most expensive problems in GW astronomy. They can achieve significant speedups while retaining the same robust and trusted algorithm at the core of their analyses.

A key motivation for this work is to establish a clear performance

baseline for GPU-based nested sampling in gravitational-wave astronomy. By adapting the community-standard `bilby` 'acceptance-walk' sampler for a GPU-native framework, we aim to isolate and quantify the speedup achieved from hardware parallelization alone. This provides a crucial reference point, enabling future work on novel sampling algorithms to be benchmarked in a way that disentangles algorithmic improvements from architectural performance gains.

In the following section, we summarise the core ideas of Bayesian inference, nested sampling and GPU architectures. We then describe the implementation of the 'acceptance-walk' sampling method in the `blackjax-ns` framework in Section 3, and validate it against the `bilby` and `dynesty` implementation in Section 4, discussing our results. Finally, we present our conclusions in Section 5.

2 BACKGROUND

2.1 Bayesian inference in GW astronomy

We provide a brief overview of the core concepts of Bayesian inference as applied to GW astronomy. For a more comprehensive treatment, we refer the reader to Skilling (2006); Thrane & Talbot (2019); Veitch et al. (2015); Ashton et al. (2019) and Abbott et al. (2020b).

The analysis of GW signals is fundamentally a problem of statistical inference, for which the Bayesian framework is the community standard. The relationship between data, d, and a set of parameters, θ, under a specific hypothesis, H, is given by Bayes' theorem (Bayes 1763):

$$\mathcal{P}(\theta|d,H) = \frac{\mathcal{L}(d|\theta,H)\pi(\theta|H)}{Z(d|H)}. \tag{1}$$

Here, the posterior, $\mathcal{P}(\theta|d,H)$, is the probability distribution of the source parameters conditioned on the observed data. It is determined by the likelihood, $\mathcal{L}(d|\theta,H)$, which is the probability of observing the data given a specific realisation of the model, and the prior, $\pi(\theta|H)$, which encodes initial beliefs about the parameter distributions.

The denominator is the Bayesian evidence,

$$Z(d|H) = \int \mathcal{L}(d|\theta,H)\pi(\theta|H)d\theta, \tag{2}$$

defined as the likelihood integrated over the entire volume of the prior parameter space.

There are two pillars of Bayesian inference of particular interest in GW astronomy. The first, parameter estimation, seeks to infer the posterior distribution $\mathcal{P}(\theta|d,H)$ of the source parameters of a signal or population of signals. The second, model selection, evaluates two competing models, H_1 and H_2, under a fully Bayesian framework by computing the ratio of their evidences, known as the Bayes factor, Z_1/Z_2. This enables principled classification of noise versus true signals, as well as the comparison of different waveform models (Abbott et al. 2020b).

In GW astronomy, the high dimensionality of the parameter space and the computational cost of the likelihood render the direct evaluation of Eq. 1 and Eq. 2 intractable (Abbott et al. 2020b). Analysis therefore relies on stochastic sampling algorithms to numerically approximate the posterior and evidence.

2.2 GPU-accelerated nested sampling

2.2.1 *The nested sampling algorithm*

Nested Sampling is a Monte Carlo algorithm designed to solve the Bayesian inference problem outlined in Sec. 2.1. A key strength of the NS algorithm is that it directly computes the Bayesian evidence, Z, while also producing posterior samples as a natural by-product of its execution (Skilling 2006; Higson et al. 2019).

The algorithm starts by drawing a population of N 'live points' from the prior distribution, $\pi(\theta|H)$. It then proceeds to iteratively evolve this population. In each iteration, the live point with the lowest likelihood value, $\mathcal{L}_{\min}$, is identified. This point is deleted from the live set and stored. It is then replaced with a new point, drawn from the prior, but subject to the hard constraint that its likelihood must exceed that of the deleted point, i.e., $\mathcal{L}_{\text{new}} > \mathcal{L}_{\min}$. This process systematically traverses nested shells of increasing likelihood, with the sequence of discarded points mapping the likelihood landscape.

The primary computational challenge within the NS algorithm is the efficient generation of a new point from the likelihood-constrained prior (Ashton et al. 2022). The specific method used for this so-called 'inner kernel' is a critical determinant of the sampler's overall efficiency and robustness (Buchner 2023).

2.2.2 *GPU architectures for scientific computing*

The distinct architectures of Central Processing Units (CPUs) and Graphics Processing Units (GPUs) offer different advantages for computational tasks. CPUs are comprised of a few powerful cores optimised for sequential task execution and low latency. In contrast, GPUs feature a massively parallel architecture, containing thousands of simpler cores designed for high-throughput computation (Owens et al. 2008).

This architecture makes GPUs exceptionally effective for problems that can be expressed in a Single Instruction, Multiple Data (SIMD) paradigm (NVIDIA Corporation 2025). In such problems, the same operation is performed simultaneously across a large number of data elements, leading to substantial performance gains over serial execution on a CPU. The primary trade-off is that algorithms must be explicitly reformulated to expose this parallelism, and not all computational problems are amenable to vectorization. There is algorithmic overlap with vectorisation and multi-threading on CPUs (Handley et al. 2015; Smith et al. 2020); nonetheless, the GPU execution model is uniquely suited to truly massive parallelism. We seek to exploit this capability in the present work.

2.2.3 *A vectorized formulation of nested sampling*

The iterative, one-at-a-time nature of the traditional NS algorithm is intrinsically serial, making it a poor fit for the parallel architecture of GPUs. To overcome this limitation, Yallup et al. (2025a) recently developed a vectorized formulation of the NS algorithm, specifically designed for highly parallel execution within the `blackjax` framework (Cabezas et al. 2024).

One of the core parts of this approach is the idea of batch processing. Instead of replacing a single live point in each iteration, the algorithm removes a batch of k points with the lowest likelihoods simultaneously (Henderson & Goggans 2014). The critical step of replacing these points is then parallelized. The algorithm launches k independent sampling processes on the GPU, with each process tasked with finding one new point that satisfies the likelihood con-

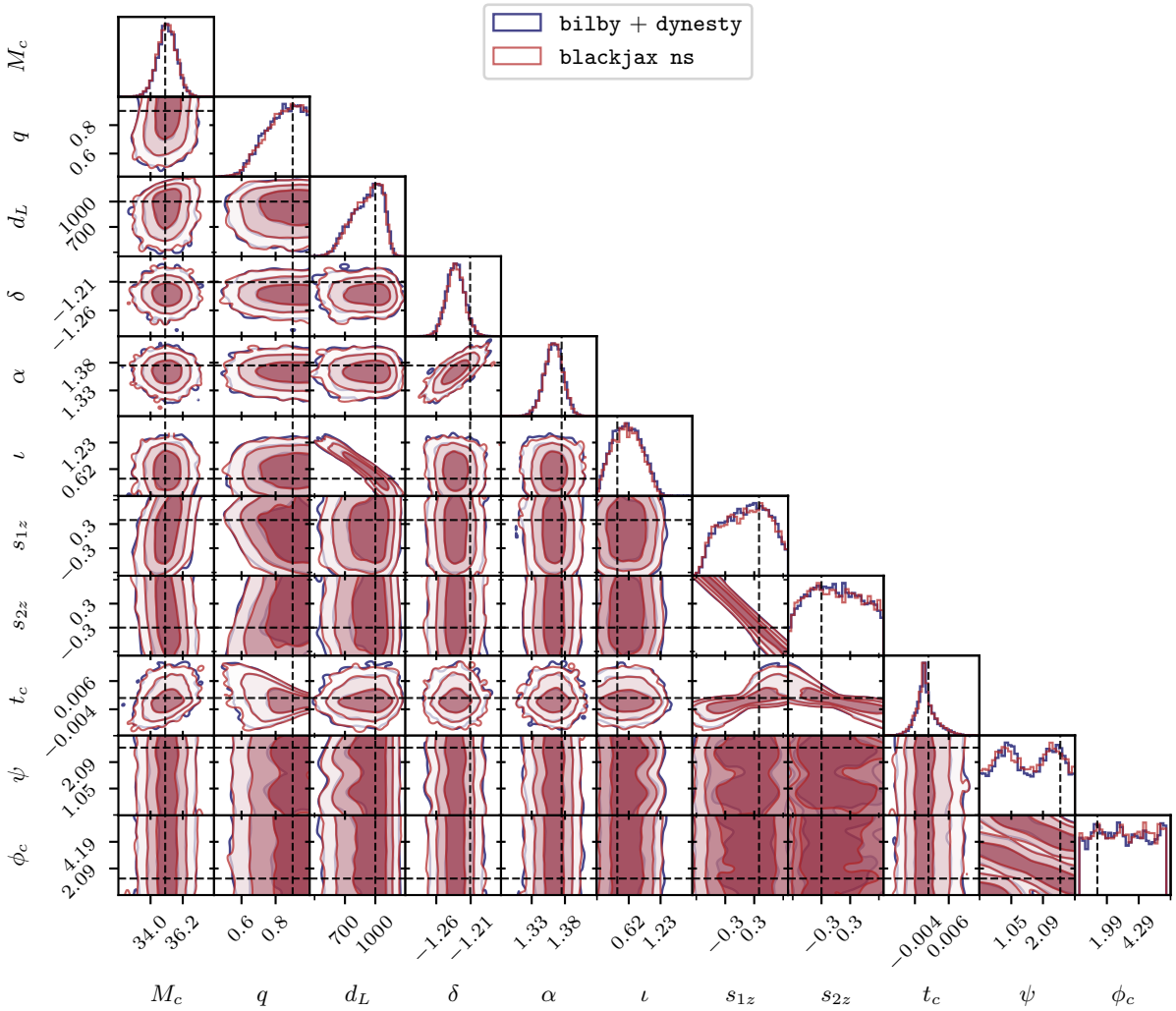

Figure 1. Recovered posteriors for the 4s signal. The posteriors are in excellent agreement with each other, demonstrating that the `blackjax-ns` implementation with our custom kernel is functionally equivalent to the `bilby + dynesty` implementation.

straint, $\mathcal{L} > \mathcal{L}_{\min}$, where $\mathcal{L}_{\min}$ is now the maximum likelihood of the discarded batch.

This reformulation transforms the computationally intensive task of sample generation from a serial challenge into a massively parallel one, thereby leveraging the architectural strengths of the GPU. Design and implementation of massively parallel Markov Chain Monte Carlo (MCMC) kernels is a non-trivial task, and an area of active research (Hoffman & Sountsov 2022; Hoffman et al. 2021). The original work of Yallup et al. (2025a) proposed a Hit-and-Run Slice Sampling (Neal 2003) kernel for this task, alongside providing a general vectorized framework which we leverage in this work. Here, we implement one particular inner kernel used within the `bilby` and `dynesty` framework. Establishing this baseline demonstrates the flexibility of the `blackjax` nested sampling framework, and gives scope for future work to explore other parallel inner sampling methods.

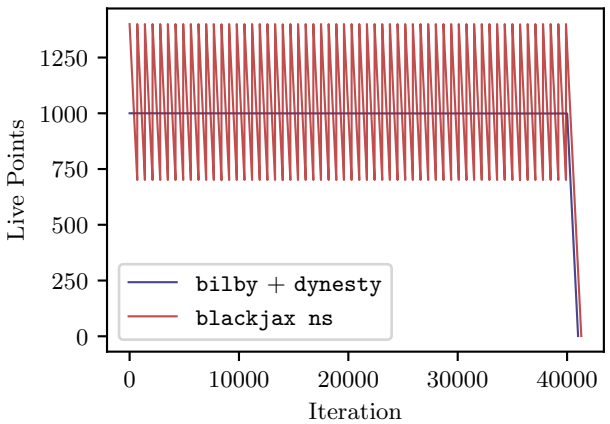

Figure 2. Comparison of the number of live points in the sequential CPU and batched GPU implementations. Although the nominal number of live points used in `blackjax-ns` is higher than in `bilby`, the saw-tooth pattern means that the effective number of live points is the same. Here, we use a batch size of $k = 0.5 \times n_{\text{live}}$.

3 METHODS

3.1 The inner sampling kernel

Several inner sampling methods are implemented within the `bilby` and `dynesty` framework (Ashton et al. 2019; Speagle 2020). In this work, we focus on a GPU-accelerated implementation of the 'acceptance-walk' method, which is a robust and widely used choice for GW analyses.

In the standard CPU-based `dynesty` implementation, the sampler generates a new live point by initiating a Markov Chain Monte Carlo (MCMC) walk from the position of the deleted live point. The proposal mechanism for the MCMC walk is based on Differential Evolution (DE) (Storn & Price 1997; ter Braak 2006), which uses the distribution of existing live points to inform jump proposals. A new candidate point is generated by adding a scaled vector difference of two other randomly chosen live points to the current point in the chain. Under the default `bilby` configuration, the scaling factor for this vector is chosen stochastically: with equal probability, it is either fixed at 1 or drawn from a scaled gamma distribution[1]. This proposed point is accepted if it satisfies the likelihood constraint, $\mathcal{L} > \mathcal{L}_{\rm min}$, where $\mathcal{L}_{\rm min}$ is the likelihood of the discarded point being replaced. The sampling process is performed in the unit hypercube space, with prior transformations applied to evaluate the likelihood of proposed points in the physical parameter space (Speagle 2020). The walk length is adaptive on a per-iteration basis; the algorithm adjusts the number of MCMC steps dynamically to target a pre-defined number of accepted steps (e.g., 60) for each new live point generated, up to a maximum limit.

This per-iteration adaptive strategy, however, is ill-suited for GPU architectures. The variable walk length required for each parallel sampler would lead to significant thread divergence, where different cores on the GPU finish their tasks at different times, undermining the efficiency of the SIMD execution model. To leverage GPU acceleration effectively, the computational workload must be as uniform as possible across all parallel processes.

Our implementation therefore preserves the core DE proposal mechanism but modifies the walk-length adaptation to be compatible with a vectorized framework. Within the `blackjax-ns` sampler, the number of MCMC steps is fixed for all parallel processes within a single batch of live point updates. The adaptive tuning is then performed at the batch level. After a batch of k new points has been generated, we compute the mean acceptance rate across all k walks. The walk length for the subsequent batch is then adjusted based on this average rate, using the same logic as `bilby` to target a desired number of accepted proposals. Adaptive tuning of walk lengths, or equivalent mechanisms (Dau & Chopin 2021), particularly in light of the GPU-batched approach, is a key avenue for future work to reduce overall runtime.

While this batch-adaptive approach is essential for efficient GPU vectorization, it introduces some important differences, discussed further in Section 3.2. Despite this architectural modification, our kernel is designed to be functionally analogous to the trusted `bilby` sampler. This design represents the most direct translation of the `bilby` logic to GPUs, making only the minimal changes required for efficient execution.

[1] Drawn from $\frac{2.38}{\sqrt{2n_{\rm dim}}} \times \Gamma(4, 0.25)$, where $n_{\rm dim}$ is the parameter space dimensionality.

3.2 Sampler configuration and settings

Several key hyperparameters govern the behaviour of the sampler and its inner kernel. The number of live points, $n_{\rm live}$, controls the resolution of the run, where a higher value results in more accurate posteriors and evidence estimates. The `n_accept` parameter sets the target number of accepted steps per MCMC walk, affecting the decorrelation of samples, and `maxmcmc` sets the maximum number of attempted steps per walk.

The primary architectural difference between our GPU-based implementation and the standard CPU-based 'acceptance-walk' kernel is the use of batched sampling. In our framework, a batch of k new points is generated and added to the live set simultaneously. This batch size is a user-configurable parameter, `num_delete`, and we find that a value of $k \approx 0.5 \times$ `n_live` provides a good balance of parallel efficiency and sampling accuracy for most problems (Yallup et al. 2025a).

This batched approach has direct consequences for the adaptive tuning of the MCMC walk length. The tuning is performed only once per `num_delete` iterations, rather than at every iteration, and every point in a given batch is tuned to have the same walk length. This design is important for preventing thread divergence and is the most natural way to implement this algorithm on a GPU. However, this less frequent tuning renders the standard `bilby` tuning parameter 'delay', a smoothing factor for the adaptation, sub-optimal. To achieve a comparable number of accepted steps per iteration, we therefore adjust this parameter for the parallel framework. The delay is modified to be a function of the batch size, recovering the standard `bilby` behaviour in the sequential limit, since it is functionally similar to averaging over batches of chains anyway.

Another subtle but critical consequence of the architectural shift to GPUs is its effect on the evolution of the live point set. In the sequential CPU case, the number of live points is approximately constant throughout the run. In the batched GPU case, the number of live points follows a 'saw-tooth' pattern, decreasing from $n_{\rm live}$ to $n_{\rm live} - k$ over one cycle of k deletions before being replenished (Figure 2). This pattern causes the effective number of live points to be lower than the nominal number, making it appear that the sampler is able to converge faster than its sequential counterpart when both are configured with the same number of live points. It is important to note that the saw-tooth pattern is common to all parallel nested sampling, including CPU-based implementations (Handley et al. 2015), and it is only the massively parallel nature of the GPU that makes it noticeable.

To conduct a fair and direct comparison between the two frameworks, this discrepancy must be accounted for. We therefore adjust the number of live points in the GPU sampler, $n_{\rm GPU}$, such that the expected prior volume compression matches that of the CPU sampler. Under these settings, both implementations will converge in approximately the same number of nested sampling iterations (see Figure 3). The following derivation outlines this adjustment.

3.2.1 *Setting n_{GPU}*

The expected log prior volume fraction, $\log X$, remaining after k iterations of a nested sampling run is given by (Skilling 2006; Higson et al. 2019; Hu et al. 2024)

$$E\left[\log X_k\right] = \sum_{i=1}^{k} \frac{1}{n_i}, \tag{3}$$

where n_i is the number of live points at iteration i. In the CPU-based implementation, $n_i = n_{\rm CPU}$ is roughly constant. After k iterations,

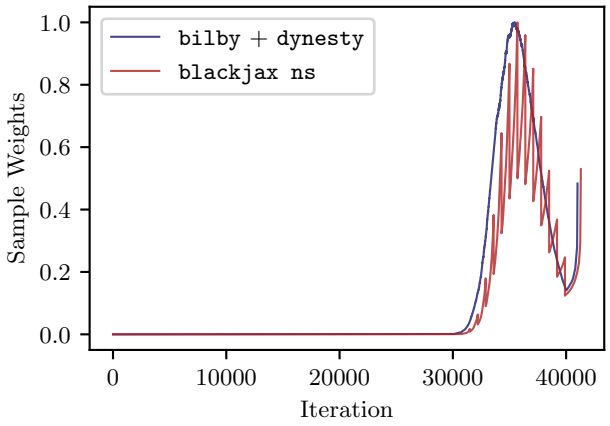

Figure 3. Comparison of the sample weights for each dead point for the sequential CPU and batched GPU implementations. The weights are calculated using the prior volumes enclosed between successive dead points and the likelihoods. The shapes are similar, and both implementations enter the bulk of the posterior distribution at similar iterations, indiciating that setting the number of live points in `blackjax-ns` to 1.4 times the number of live points in `bilby` does indeed result in a like-for-like comparison. The same saw-tooth pattern can be seen in the weights for the `blackjax-ns` implementation.

the expected log volume fraction is therefore

$$\log X_{\text{CPU}} = \frac{k}{n_{\text{CPU}}}. \tag{4}$$

In our GPU implementation, the number of live points decreases over one batch cycle of $k =$ `num_delete` deletions. The total log volume shrinkage over one such cycle is the sum over the decreasing number of live points:

$$\log X_{\text{GPU}} = \sum_{i=0}^{k-1} \frac{1}{n_{\text{GPU}} - i} \approx \ln\left(\frac{n_{\text{GPU}}}{n_{\text{GPU}} - k}\right). \tag{5}$$

To ensure a fair comparison, we equate the expected shrinkage of both methods over one cycle of k iterations:

$$\frac{k}{n_{\text{CPU}}} \approx \ln\left(\frac{n_{\text{GPU}}}{n_{\text{GPU}} - k}\right). \tag{6}$$

Using our recommended setting of $k = 0.5 \times n_{\text{GPU}}$, this simplifies to $0.5 \times n_{\text{GPU}}/n_{\text{CPU}} \approx \ln(2)$. We can therefore derive the required number of live points for the GPU sampler to be

$$n_{\text{GPU}} \approx 2\ln(2) \times n_{\text{CPU}} \approx 1.4 \times n_{\text{CPU}}. \tag{7}$$

In all comparative analyses presented in this paper, we configure the number of live points according to this relation to ensure an equal effective number of live points and like-for-like timing comparisons.

3.3 Likelihood

To assess the performance of our framework, we employ a standard frequency-domain likelihood (Veitch et al. 2015; Christensen & Meyer 2022). The total speedup in our analysis is achieved by accelerating the two primary computational components of the inference process: the sampler and the likelihood evaluation. The former is parallelized at the batch level as described in Sec. 3.1, while the latter is accelerated using a GPU-native waveform generator.

For this purpose, we generate gravitational waveforms using the `ripple` library, which provides GPU-based implementations of common models (Edwards et al. 2024; Wong et al. 2023a; Wouters et al. 2024). This allows the waveform to be calculated in parallel across the frequency domain and across multiple sets of parameters, enabling massive efficiency gains by ensuring that this calculation does not become a serial bottleneck. To isolate the speedup from this combined GPU-based framework, we deliberately avoid other established acceleration methods like heterodyning (Krishna et al. 2023; Zackay et al. 2018; Leslie et al. 2021; Cornish 2013), though these are available in the `jimgw` library too (Wong et al. 2023a).

For the analyses in this paper, we restrict our consideration to binary black hole systems with aligned spins, for which we use the `IMRPhenomD` waveform approximant (Khan et al. 2016). Further details on the specific likelihood configuration for each analysis, including noise curves and data segments, are provided in Section 4.

Table 1. Prior distributions for the parameters of the binary black hole system. The specific ranges for the masses and spins are dependent on the signal and are specified in Section 4.

Parameter	Description	Prior Distribution	Range
M_c	Chirp Mass	Uniform	-
q	Mass Ratio	Uniform	-
χ_1, χ_2	Aligned spin components	Uniform	-
d_L	Luminosity Distance	Power Law (2)	[100, 5000] Mpc
θ_{JN}	Inclination Angle	Sine	[0, π] rad
ψ	Polarization Angle	Uniform	[0, π] rad
ϕ_c	Coalescence Phase	Uniform	[0, 2π] rad
t_c	Coalescence Time	Uniform	[-0.1, 0.1] s
α	Right Ascension	Uniform	[0, 2π] rad
δ	Declination	Cosine	[-π/2, π/2] rad

3.4 Priors

For this initial study, we adopt a set of standard, separable priors on the source parameters, which are summarized in Table 1. The specific ranges for these priors are dependent on the duration of the signal, and are also given in Section 4.

As is the default within the `bilby` framework, we sample directly in chirp mass, $\mathcal{M}$, and mass ratio, q. We use priors that are uniform in these parameters directly, instead of uniform in the component masses. The aligned spin components, χ_1 and χ_2, are also taken to be uniform over their allowed range. The coalescence time, t_c, is assumed to be uniform within a narrow window around the signal trigger time.

For the luminosity distance, d_L, we adopt a power-law prior of the form $p(d_L) \propto d_L^2$. This prior corresponds to a distribution of sources that is uniform in a Euclidean universe. While this is a simplification that is less accurate at higher redshifts (Romero-Shaw et al. 2020), it is a standard choice for many analyses and serves as a robust baseline for this work.

These priors were chosen to facilitate a direct, like-for-like comparison against the CPU-based `bilby` and `dynesty` framework, and in all such comparisons identical priors were used. The implementation of more astrophysically motivated, complex prior distributions for mass, spin, and luminosity distance is left to future work and is independent of the algorithmic performance.

4 RESULTS AND DISCUSSION

We now validate the performance of our framework through a series of analyses. In all of the results, the `bilby` analyses were executed on

a 16-core CPU instance using Icelake nodes, while the `blackjax-ns` analyses were performed on a single NVIDIA L4 GPU, unless otherwise specified.

4.1 Simulated signals

4.1.1 4-second simulated signal

We begin by analysing a 4-second simulated signal from a binary black hole (BBH) merger. The injection parameters for this signal are detailed in Table 2. To ensure a direct, like-for-like comparison, the signal was injected into simulated detector noise using the `bilby` library, and then loaded into both sets of analyses. The analysis uses a three-detector network, with the design sensitivity for the fourth LIGO-Virgo-KAGRA observing run (O4). The frequency range of data analysed is from 20 Hz to 1024 Hz, and the network matched filter SNR is 39.6.

Both the CPU and GPU-based analyses were configured with 1000 effective live points. As detailed in Sec. 3.2.1, this corresponded to setting the number of live points in `blackjax-ns` to 1400, with `num_delete` = 700. The termination condition in both cases was set to dlogZ < 0.1 (the default in `bilby`) and we used the settings `naccept` = 60, `maxmcmc` = 5000 and `use_ratio = True`. In both cases, periodic boundary conditions were used for the right ascension, polarization angle, and coalescence phase parameters. The prior ranges for the chirp mass and mass ratio were set to $[25.0, 50.0]\ M_\odot$ and $[0.25, 1.0]$, respectively, with priors on the aligned spin components over the range $[-1, 1]$.

The recovered posterior distributions, shown in Figure 1, demonstrate excellent statistical agreement between the two frameworks. This result validates that our custom 'acceptance-walk' kernel within the vectorized `blackjax-ns` framework is functionally equivalent to the trusted sequential implementation in `bilby`. The computed log-evidence values are also in strong agreement, as shown in Figure 4, confirming that our implementation provides a robust unified framework for both parameter estimation and model selection.

The CPU-based `bilby` run completed in 2.99 hours on 16 cores, for a total of 47.8 CPU-hours. In contrast, the GPU-based `blackjax-ns` run completed in 1.25 hours. This corresponds to a sampling time speedup factor of 38×. The `bilby` and `blackjax-ns` runs performed 62.9 million and 62.5 million likelihood evaluations and had a mean number of accepted steps per iteration of 28.5 and 30.7, respectively.

Beyond sampling time, we also consider the relative financial cost. Based on commercial on-demand rates from Google Cloud[2], the rental cost for a 16-core CPU instance and an L4 GPU instance were approximately equivalent at the time of this work. This equivalence in hourly cost implies a direct cost-saving factor of approximately 2.4× for the GPU-based analysis.

In the interest of benchmarking, we also performed the analysis on an A100 NVIDIA GPU configured with the same run settings as the L4 GPU. The A100 is a more powerful GPU and resulted in a lower runtime. Interestingly, however, when comparing the relative commercial hourly rates of the two GPUs, the L4 GPU analysis was actually cheaper. Indeed the A100 analysis actually had a worse cost compared to the CPU implementation. We summarise these results in Table 3.

[2] Accessed on 2025-09-03 from https://cloud.google.com/compute/gpus-pricing.

Table 2. Injection parameters for the 4s signal.

Parameter	Value
$\mathcal{M}$	35.0 $M_\odot$
q	0.90
χ_1	0.40
χ_2	-0.30
d_L	1000 Mpc
$\theta_{\rm JN}$	0.40 rad
ψ	2.66 rad
ϕ	1.30 rad
t_c	0.0 s
α	1.38 rad
δ	-1.21 rad

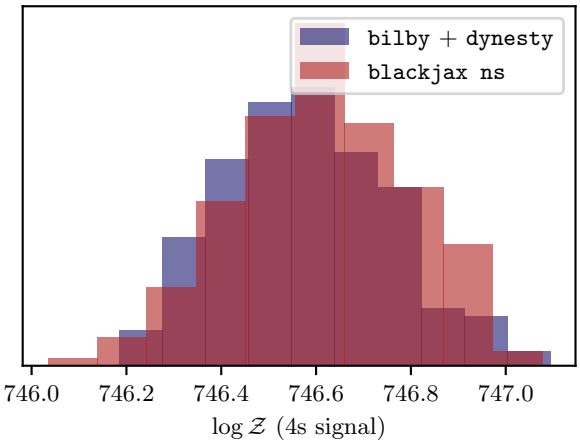

Figure 4. Comparison of the log evidence for the 4s signal. The results are in excellent agreement, demonstrating the robustness of the `blackjax-ns` implementation in recovering the same posteriors and evidence as the `bilby` implementation. This unifies parameter estimation and evidence evaluation into a single GPU-accelerated framework.

Table 3. Comparison of the sampling times and cost savings for the 4s signal.

Implementation + Hardware	Time (h)	Speedup	Cost Saving
`bilby` (16 Icelake CPU cores)	47.8	-	-
`blackjax-ns` (NVIDIA L4)	1.25	38×	2.4×
`blackjax-ns` (NVIDIA A100)	0.93	51×	0.6×

4.2 Injection Study

To systematically assess the performance and robustness of our framework across a diverse parameter space, we performed an injection study comparing our GPU-based `blackjax-ns` sampler against the CPU-based `bilby+dynesty` implementation. A population of 100 simulated BBH signals was generated using `bilby` and injected into noise representative of the O4 detector network sensitivity. The network signal-to-noise ratios (SNR) for this injection set span a range from 1.84 to 17.87 (Figure 5). As above, we use a three-detector network for the analysis.

For this study, the prior ranges were set to $[20.0, 50.0]\ M_\odot$ for the chirp mass and $[0.5, 1.0]$ for the mass ratio. The aligned spin com-

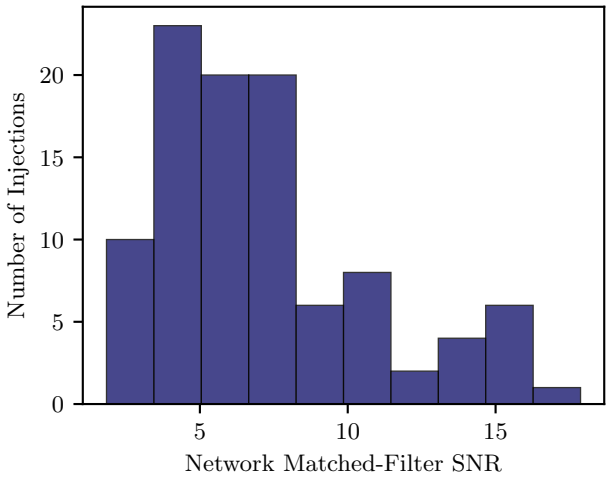

Figure 5. Distribution of the network signal-to-noise ratios (SNR) for the injected signals.

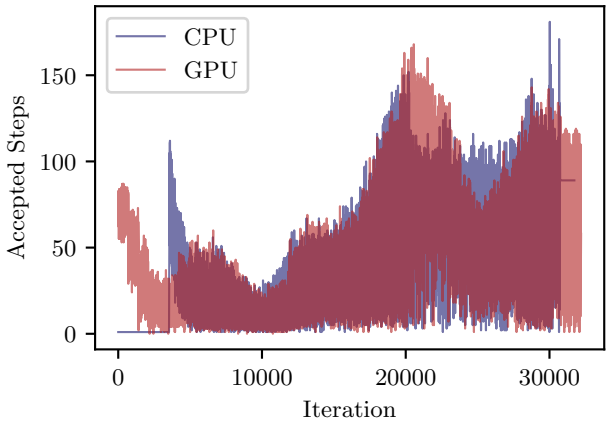

Figure 6. Comparison of the accepted number of steps per iteration for the sequential CPU and batched GPU analyses of the first signal from the injection study. We adapt our 'delay' parameter from the tuning formula such that we obtain similar accepted steps for the two implementations. The `blackjax-ns` implementation can only perform tuning every `num_delete` iterations, so we tune the chain length more aggressively.

ponents were bounded by uniform priors over the range $[-0.8, 0.8]$. All other prior distributions are as defined in Table 1.

All analyses in this study were performed using 1000 live points for `bilby` and 1400 live points for `blackjax-ns`. Given that quite a few signals in the set have a low SNR, and therefore a low Bayes factor when comparing the signal hypothesis to the noise-only hypothesis, a more robust termination condition was required to ensure accurate evidence computation as well as posterior estimation. We therefore set the termination criterion based on the fractional remaining evidence, such that the analysis stops when the estimated evidence in the live points is less than 0.1% of the accumulated evidence. Both samplers were configured with `naccept` = 60 and `maxmcmc` = 5000, and the `blackjax-ns` sampler used a batch size of `num_delete` = 700.

The resulting PP plot for our `blackjax-ns` sampler with the 'acceptance-walk' kernel is shown in Figure 7. This plot evaluates the self-consistency of the posteriors by checking the distribution of true injected parameter values within their recovered credible intervals.

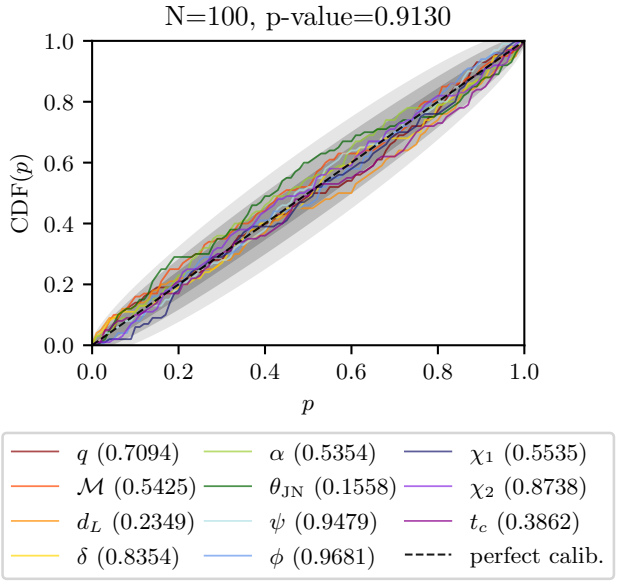

Figure 7. Percentile-percentile (PP) coverage plot for the 100-injection study, obtained with the `blackjax-ns` sampler. The cumulative fraction of events where the true injected parameter falls below a given credible level is plotted against that credible level. The proximity of all parameter curves to the diagonal indicates that the posterior credible intervals are statistically well-calibrated. A corresponding plot for the `bilby+dynesty` analysis, which should be identical, is provided in the Appendix for reference.

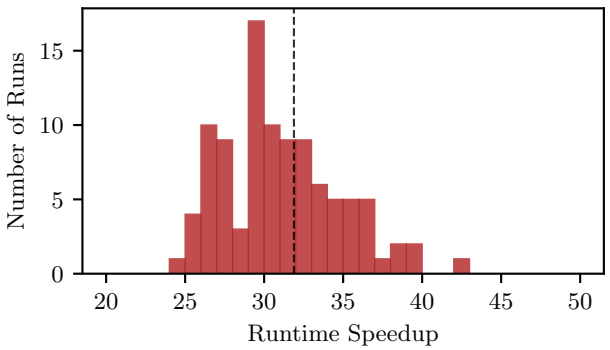

Figure 8. The runtime speedups for all 100 events in the injection study. For the CPU-based analyses, we take the wall-time and multiply by 16 to get the total runtime. For the GPU-based analyses, we use the wall-time directly.

The results demonstrate that the credible intervals are well-calibrated, with the cumulative distributions for all parameters lying within the expected statistical variance of the line of perfect calibration. This confirms the robustness of our implementation. The full set of posterior and evidence results for all 100 injections is made publicly available in the data repository accompanying this paper (Prathaban et al. 2025a). We show the consistency of the evidence estimates in Figure 10.

The average number of likelihood evaluations for the `blackjax-ns` sampler was 38.4 million, consistent with the average of 37.5 million for the `bilby+dynesty` sampler. A more detailed breakdown of all run statistics is provided in Appendix B. In terms of runtime, the `blackjax-ns` sampler completed the analy-

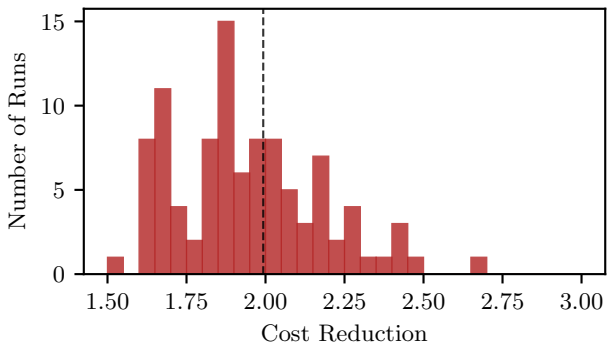

Figure 9. The cost reductions for all 100 events in the injection study. Although the runtime speedups are significant, the more honest and fair comparison is of the relative costs of the two sets of analyses, since GPUs are more expensive than CPUs. We use the commercial on-demand rates for a 16-core CPU instance and an L4 GPU instance at the time of this work, and compare the cost savings obtained from using our GPU-accelerated pipeline.

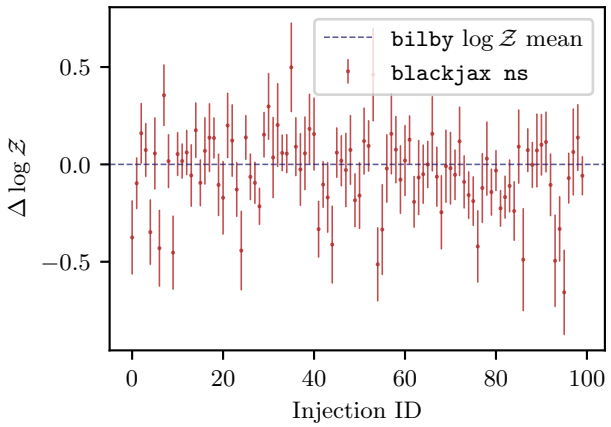

Figure 10. The log evidence difference between the `blackjax-ns` and `bilby+dynesty` samplers for each injection in the study. For each estimate, we take the mean log evidence estimated by `bilby` to be the reference value, and subtract the `bilby` estimates from the log evidences calculated by `blackjax-ns`. The error bars represent 1σ confidence intervals for these differences. There appears to be some underestimation of the error bars, potentially due to some sets of samples not being as well decorrelated as others, despite a reasonable average decorrelation. However, a calibration test was performed to quantify this. For the $\Delta \log \mathcal{Z}$ estimates, we calculate the percentiles at which $\Delta \log \mathcal{Z} = 0$ lies. If the error bars are correctly calibrated, the resulting distribution of percentiles should be uniform; the Kolmogorov-Smirnov test returns a p-value of 0.062, indicating that at the 5% significance level the error bars are well-calibrated.

ses in an average of 0.82 hours per injection, with individual run times ranging from 32 to 76 minutes. In contrast, the CPU-based runs required an average of 26.3 total CPU-hours, with some runs taking as long as 45 hours. Figure 8 shows the distribution of the resulting runtime speedup factors, which have a mean value of 32×. As in section 4.1.1, we also translate this performance gain into a cost reduction using commercial cloud computing rates. As shown in Figure 9, the speedup corresponds to an average cost-reduction factor of 2.0× for the GPU-based analysis compared to its CPU-based counterpart.

4.3 Disentangling Sources of GPU Acceleration

The overall performance gain of our framework arises from two distinct forms of parallelisation: the parallel evaluation of a single likelihood across its frequency bins (intra-likelihood), and the parallel sampling process itself, where multiple MCMC chains are run simultaneously (inter-sample). In this section, we attempt to disentangle these two contributions.

We configured our `blackjax-ns` sampler to run in a sequential mode. This was achieved by setting the batch size to one (`num_delete` = 1) and the number of live points to 1000, identical to the CPU analysis. With a batch size of one, the 'saw-tooth' pattern in the live point population described in Sec. 3.2 is eliminated, making the effective number of live points equal to the nominal value. This removes the need for the corrective scaling of n_{live} applied in our main parallel analyses, and we can also use the same 'delay' for tuning the walk lengths as for the `bilby` runs. In this configuration, the algorithm proceeds analogously to the `bilby` implementation, and the only source of acceleration relative to the CPU is the GPU's ability to parallelise the likelihood calculation over the frequency domain.

For this test, we used the first signal from our injection study (network SNR of 8.82). The baseline CPU-based `bilby` analysis required 30.4 total CPU-hours to complete. The sequential-GPU analysis, which benefits only from intra-likelihood parallelisation, completed in 9.1 hours. This represents a speedup of 3.3×. Finally, the fully parallel run, using its correctly scaled live point count of 1400 and a batch size of `num_delete` = 700, completed in 0.82 hours. This corresponds to a further speedup of 11.1× over the sequential-GPU run.

This result demonstrates that while a GPU-native likelihood provides a significant performance benefit, the more dominant contribution to the overall speedup (a total of 37.1× in this case) originates from the massively parallel, batched sampling architecture. This result underscores the importance of the algorithmic reformulation of nested sampling pioneered in Yallup et al. (2025a).

4.4 8-second simulated signal

To investigate the scalability of our framework with a more computationally demanding likelihood, we now analyse a simulated 8-second BBH signal. The injection parameters are detailed in Table 4. For this analysis, the data are evaluated between 20 Hz and 2048 Hz, quadrupling the number of frequency bins compared to the 4-second signal analysis. As above, the signal was injected into noise from a three-detector network with O4 sensitivity, resulting in a network SNR of 11.25.

The prior ranges for the chirp mass and mass ratio were set to $[10.0, 25.0]\ M_\odot$ and $[0.25, 1.0]$, respectively. The aligned spin components were bounded by uniform priors over the range $[-0.8, 0.8]$, with all other prior distributions as defined in Table 1. The sampler configurations were identical to those used in the injection study (Sec. 4.2), with $n_{\text{live}} = 1400$ for `blackjax-ns` and $n_{\text{live}} = 1000$ for `bilby`, and an `naccept` target of 60. Again, we use the termination condition described in Section 4.2.

The recovered posterior distributions and log-evidence values are shown in Figure 11 and Figure 12. We again find excellent agreement between the two frameworks, further validating the accuracy of our implementation, despite the differences arising from parallelisation. The CPU-based `bilby` analysis required 167.2 total CPU-hours to converge, performing 58 million likelihood evaluations. Our GPU-based `blackjax-ns` sampler completed the same analysis in 4.55

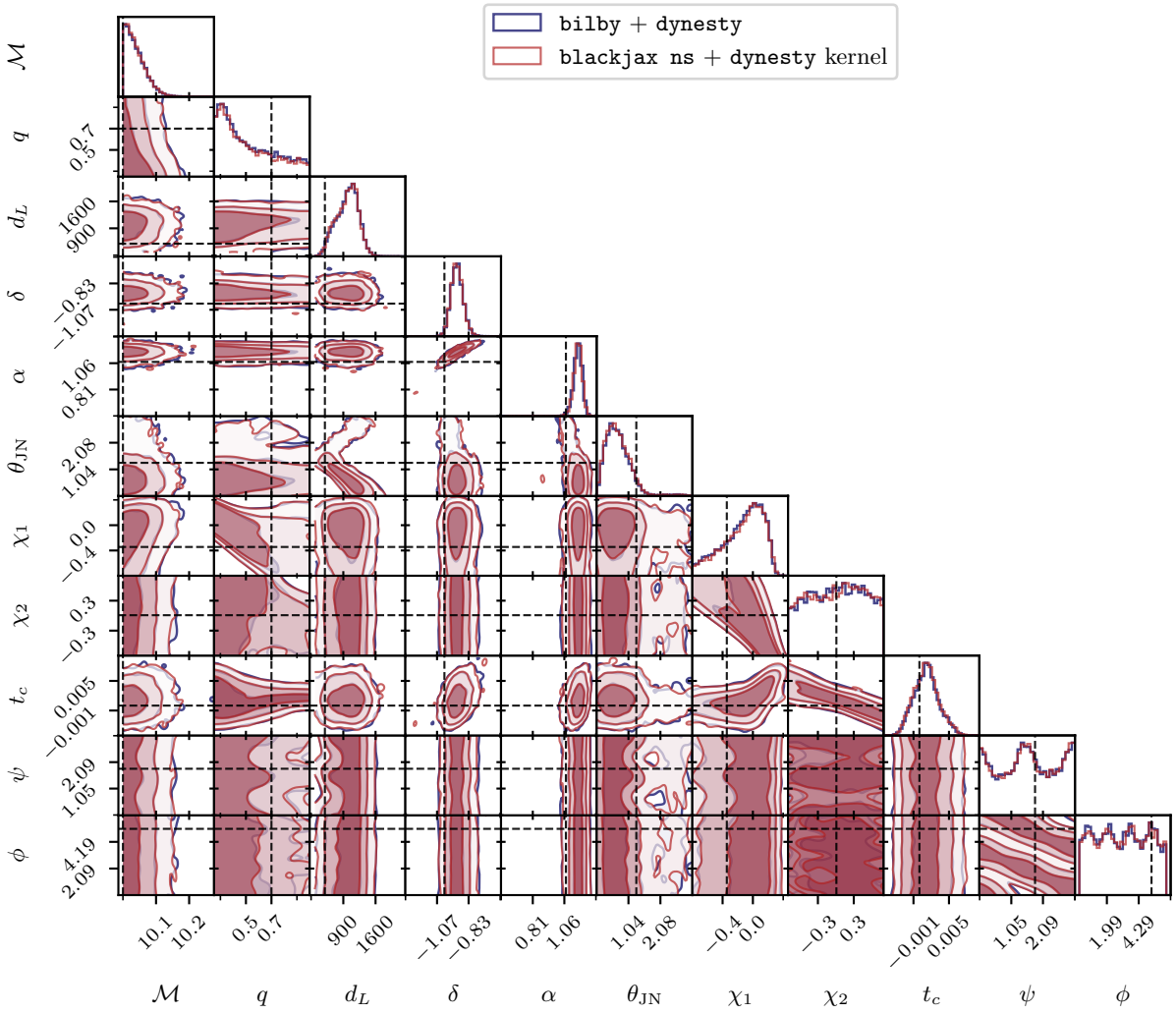

Figure 11. Recovered posterior distributions for the 8s simulated signal, comparing our GPU-based `blackjax-ns` sampler with the CPU-based `bilby` sampler. The injected values are marked by black lines. The strong statistical agreement confirms the validity of our implementation for longer-duration signals. The GPU implementation provided a wall-time speedup of 46× and a cost reduction of 2.9×.

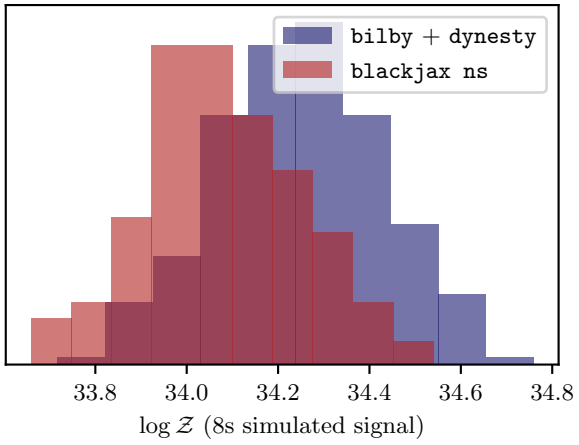

Figure 12. Comparison of the recovered log-evidence (Z) for the 8s signal. The results from both the `bilby` and `blackjax-ns` frameworks are consistent within their estimated numerical uncertainties.

Table 4. Injection parameters for the 8s signal.

Parameter	Value
$\mathcal{M}$	10.0 $M_\odot$
q	0.70
χ_1	-0.34
χ_2	0.01
d_L	500 Mpc
$\theta_{\rm JN}$	1.30 rad
ψ	1.83 rad
ϕ	5.21 rad
t_c	0.0 s
α	1.07 rad
δ	-1.01 rad

hours, with 61 million likelihood evaluations, corresponding to a wall-time speedup factor of 37×. The associated cost-reduction factor for this analysis was 2.3×.

The scaling of the runtime for this longer-duration signal provides

a key insight into the practical limits of GPU parallelisation. In an ideal model where the GPU has sufficient parallel cores for every frequency bin, the likelihood evaluation time would be independent of signal duration. In such a scenario, the total runtime would be dictated primarily by the number of likelihood evaluations needed for convergence. In this case, the analysis should have taken roughly the same time as the 4-second signal analysis, or $1.25 \times 61/63 = 1.21$ hours. However, we observe that the runtime increased significantly more than what would be predicted by the relative number of likelihood evaluations alone.

This discrepancy arises because the computational load of the larger frequency array exceeds the parallel processing capacity of the L4 GPU. As a result, the GPU's internal scheduler must batch the calculation across the frequency domain, re-introducing a partial serial dependency that makes the likelihood evaluation time scale with the number of bins. This result serves as an important practical clarification to a common argument for GPU acceleration. While the idealised model of parallelisation suggests that evaluation time should be independent of signal duration, our work demonstrates that this does not hold indefinitely. Once the problem size saturates the GPU's finite resources, such as its compute or memory bandwidth, the runtime once again begins to scale with the number of frequency bins, a behaviour analogous to the scaling seen in the CPU-based case.

5 CONCLUSIONS

In this work, we have presented the development and validation of a GPU-accelerated implementation of the 'acceptance-walk' nested sampling kernel, a trusted algorithm from the community-standard `bilby` and `dynesty` framework. By integrating this kernel into the vectorized `blackjax-ns` framework, we have created a tool for rapid, unified Bayesian parameter estimation and model selection. Through systematic studies of simulated binary black hole signals, we demonstrated that our implementation is functionally analogous to its CPU-based counterpart, producing statistically consistent posterior distributions and evidence estimates with typical wall-time speedups of 20-40× and cost reductions of 1.5-2.5×.

It is important to contextualize the cost savings reported in this work. As the trajectory of high-performance computing is increasingly driven by large-scale investment in GPU hardware for artificial intelligence, we anticipate that the cost-per-computation on these devices will continue to fall relative to CPUs. Therefore, the cost-saving advantage of our approach is expected to become more pronounced in the future, even if the relative runtimes remain unchanged. Furthermore, the JAX (Bradbury et al. 2018) ecosystem is very well suited to distribution over multiple devices and architectures. It has been closely developed alongside the Tensor Processing Unit (TPU) architecture (Jouppi et al. 2023), offering a clear avenue to integrate this work with such frontier low-cost hardware.

A key conclusion of this work is that the performance gains reported here are primarily a consequence of the architectural shift to the GPU. This implies that similar relative speedups are achievable for more computationally demanding waveforms, such as those including spin precession and higher order modes (e.g., `IMRPhenomXPHM` (Pratten et al. 2021)), provided they are also implemented in a GPU-native framework. Our analysis of a longer-duration signal confirmed, however, that this scalability is ultimately limited by the finite resources of the hardware, a practical constraint that can be addressed with more powerful GPUs or likelihood-acceleration techniques like heterodyning (Krishna et al. 2023; Zackay et al. 2018; Leslie et al. 2021; Cornish 2013). Importantly, by significantly reducing the computational cost of nested sampling, our framework makes tackling previously prohibitive next-generation analyses (Hu & Veitch 2024; Baker et al. 2025), such as those involving long-duration signals or high SNR signals, a more viable prospect.

Beyond future-proofing the community's core analysis tools, this work establishes an important performance baseline. As the community develops novel, GPU-accelerated sampling algorithms, it is essential to disentangle performance gains originating from the hardware parallelization itself from those arising from genuine algorithmic innovation. By developing a functionally equivalent, GPU-native version of one of the community-standard algorithms, we have isolated and quantified the speedup attributable solely to the architectural shift, which allows for batched sampling and parallelized likelihood evaluations on a GPU. This provides a robust reference against which future, more advanced GPU-based samplers can be rigorously benchmarked, in order to guide the development of next generation Bayesian inference tools.

There are several future avenues of research and development. Having established this performance baseline, we can now rigorously evaluate novel inner sampling kernels designed specifically for the GPU architecture. While the speedups reported in this work are significant, the 'acceptance-walk' sampling method was never designed to be optimised for the GPU architecture. This motivates the exploration of alternative inner sampling kernels, such as those based on the Hit-and-Run Slice Sampling (HRSS) algorithm introduced in Yallup et al. (2025a), which may offer more efficient performance on this architecture.

GPU-accelerated sampling algorithms such as the one presented in this work can only run when paired with a GPU-native likelihood. Our work therefore provides additional motivation for the continued development and porting of more complex and physically comprehensive waveform models to GPU-based libraries like `ripple` (Edwards et al. 2024). On the software side, future work will involve integrating additional features from `bilby` (Ashton et al. 2019) into our framework, such as the astrophysically motivated prior distributions for masses, spins, and luminosity distance, to enable more realistic analyses that are of interest to the GW community.

Finally, the batched architecture of our sampler is highly compatible with machine learning techniques used for accelerating Bayesian inference. Methods that use normalizing flows to speed up the core nested sampling algorithm, such as those in Williams et al. (2021) and Prathaban et al. (2025c), are most computationally efficient when they can evaluate a large number of samples in parallel. The serial nature of a conventional CPU-based sampler represents a significant bottleneck for these models. In contrast, our parallel framework is a natural fit for these operational requirements. This compatibility enables the development of highly efficient, hybrid inference pipelines that combine GPU-accelerated sampling with GPU-native machine learning models, further lowering the computational cost of future gravitational-wave analyses.

ACKNOWLEDGEMENTS

This work was supported by the research environment and infrastructure of the Handley Lab at the University of Cambridge. MP is supported by the Harding Distinguished Postgraduate Scholars Programme (HDPSP). JA is supported by a fellowship from the Kavli Foundation. This work was performed using the Cambridge Service for Data Driven Discovery (CSD3), part of which is operated by the University of Cambridge Research Computing on behalf of the

STFC DiRAC HPC Facility (www.dirac.ac.uk). The DiRAC component of CSD3 was funded by BEIS capital funding via STFC capital grants ST/P002307/1 and ST/R002452/1 and STFC operations grant ST/R00689X/1. DiRAC is part of the National e-Infrastructure. This material is based upon work supported by the Google Cloud research credits program, with the award GCP397499138.

DATA AVAILABILITY

All the data used in this analysis, including the relevant nested sampling chains, can be obtained from Prathaban et al. (2025a). We include a file with all the code to reproduce the plots in this paper. The `blackjax-ns` code is available from Yallup et al. (2025b), and the custom kernel implementated in this paper can be found at Prathaban et al. (2025b). The latter link also contains instructions and example files on how to use the kernel to run GW analyses.

SOFTWARE AND TOOLS

The authors acknowledge the use of AI tools in preparing this manuscript and the accompanying software. Generative models, specifically Gemini 2.5 Pro and Claude 4.0 Sonnet, were used to assist with drafting and refining the text. Additionally, the AI coding agent Claude Code was utilized during software development. All AI-generated outputs, including text and code, were carefully reviewed, edited, and validated by the authors to ensure their accuracy.

REFERENCES

Abbott B. P., et al., 2016, Phys. Rev. Lett., 116, 061102
Abbott B. P., et al., 2017a, Phys. Rev. Lett., 119, 161101
Abbott B. P., et al., 2017b, Nature, 551, 85
Abbott B. P., et al., 2019, Phys. Rev. X, 9, 031040
Abbott B. P., et al., 2020a, Living Rev. Rel., 23, 3
Abbott B. P., et al., 2020b, Classical and Quantum Gravity, 37, 055002
Abbott R., et al., 2021, Phys. Rev. D, 103, 122002
Abbott R., et al., 2023a, Phys. Rev. X, 13, 011048
Abbott R., et al., 2023b, Phys. Rev. X, 13, 041039
Abbott R., et al., 2024, Phys. Rev. D, 109, 022001
Acernese F., et al., 2014, Classical and Quantum Gravity, 32, 024001
Ashton G., et al., 2019, Astrophys. J. Suppl., 241, 27
Ashton G., Bernstein N., Buchner J., et al. 2022, Nature Reviews Methods Primers, 2, 39
Baker A. M., Lasky P. D., Thrane E., Golomb J., 2025, Significant challenges for astrophysical inference with next-generation gravitational-wave observatories (arXiv:2503.04073)
Bayes T., 1763, Philosophical Transactions of the Royal Society of London, 53, 370
Biwer C. M., Capano C. D., De S., Cabero M., Brown D. A., Nitz A. H., Raymond V., 2019, Publications of the Astronomical Society of the Pacific, 131, 024503
Bradbury J., et al., 2018, JAX: composable transformations of Python+NumPy programs, http://github.com/google/jax
Branchesi M., et al., 2023, Journal of Cosmology and Astroparticle Physics, 2023, 068
Buchner J., 2023, Statistics Surveys, 17, 169
Cabezas A., Corenflos A., Lao J., Louf R., 2024, BlackJAX: Composable Bayesian inference in JAX (arXiv:2402.10797)
Christensen N., Meyer R., 2022, Rev. Mod. Phys., 94, 025001
Cornish N. J., 2013, Fast Fisher Matrices and Lazy Likelihoods (arXiv:1007.4820), https://arxiv.org/abs/1007.4820
Dau H.-D., Chopin N., 2021, Waste-free Sequential Monte Carlo (arXiv:2011.02328), https://arxiv.org/abs/2011.02328
Edwards T. D. P., Wong K. W. K., Lam K. K. H., Coogan A., Foreman-Mackey D., Isi M., Zimmerman A., 2024, Phys. Rev. D, 110, 064028
Handley W. J., Hobson M. P., Lasenby A. N., 2015, Mon. Not. Roy. Astron. Soc., 453, 4385
Henderson R. W., Goggans P. M., 2014, in AIP Conference Proceedings. pp 100–105, doi:10.1063/1.4903717
Higson E., Handley W., Hobson M., Lasenby A., 2018, Bayesian Analysis, 13, 873
Higson E., Handley W., Hobson M., Lasenby A., 2019, Statistics and Computing, 29, 891
Hoffman M. D., Sountsov P., 2022, in Camps-Valls G., Ruiz F. J. R., Valera I., eds, Proceedings of Machine Learning Research Vol. 151, Proceedings of The 25th International Conference on Artificial Intelligence and Statistics. PMLR, pp 7799–7813, https://proceedings.mlr.press/v151/hoffman22a.html
Hoffman M., Radul A., Sountsov P., 2021, in Banerjee A., Fukumizu K., eds, Proceedings of Machine Learning Research Vol. 130, Proceedings of The 24th International Conference on Artificial Intelligence and Statistics. PMLR, pp 3907–3915, https://proceedings.mlr.press/v130/hoffman21a.html
Hu Q., Veitch J., 2024, Costs of Bayesian Parameter Estimation in Third-Generation Gravitational Wave Detectors: a Review of Acceleration Methods (arXiv:2412.02651), https://arxiv.org/abs/2412.02651
Hu Z., Baryshnikov A., Handley W., 2024, Mon. Not. Roy. Astron. Soc., 532, 4035
Jouppi N., et al., 2023, in Proceedings of the 50th annual international symposium on computer architecture. pp 1–14
Khan S., Husa S., Hannam M., Ohme F., Pürrer M., Jiménez Forteza X., Bohé A., 2016, Phys. Rev. D, 93, 044007
Krishna K., Vijaykumar A., Ganguly A., Talbot C., Biscoveanu S., George R. N., Williams N., Zimmerman A., 2023, Accelerated parameter estimation in Bilby with relative binning (arXiv:2312.06009), https://arxiv.org/abs/2312.06009
Leslie N., Dai L., Pratten G., 2021, Physical Review D, 104
NVIDIA Corporation 2025, CUDA C++ Programming Guide, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
Neal R. M., 2003, The Annals of Statistics, 31, 705
Owens J. D., Houston M., Luebke D., Green S., Stone J. E., Phillips J. C., 2008, Proceedings of the IEEE, 96, 879
Prathaban M., Handley W., 2024, Monthly Notices of the Royal Astronomical Society, 533, 1839
Prathaban M., et al., 2025b, blackjax_ns_gw: blackjax-ns for Gravitational Wave Inference on GPUs, https://github.com/mrosep/blackjax_ns_gw
Prathaban M., Yallup D., Alvey J., Yang M., Templeton W., Handley W., 2025a, Gravitational-wave inference at GPU speed: A bilby-like nested sampling kernel within blackjax-ns, doi:10.5281/zenodo.17012011, https://doi.org/10.5281/zenodo.17012011
Prathaban M., Bevins H., Handley W., 2025c, Monthly Notices of the Royal Astronomical Society, p. staf962
Pratten G., et al., 2021, Phys. Rev. D, 103, 104056
Romero-Shaw I. M., et al., 2020, Monthly Notices of the Royal Astronomical Society, 499, 3295
Skilling J., 2006, Bayesian Analysis, 1, 833
Smith R. J. E., Ashton G., Vajpeyi A., Talbot C., 2020, Mon. Not. Roy. Astron. Soc., 498, 4492
Speagle J. S., 2020, Monthly Notices of the Royal Astronomical Society, 493, 3132–3158
Storn R., Price K., 1997, J. Global Optim., 11, 341
The LIGO Scientific Collaboration et al., 2015, Classical and Quantum Gravity, 32, 074001
Thrane E., Talbot C., 2019, Publications of the Astronomical Society of Australia, 36
Veitch J., et al., 2015, Physical Review D, 91
Williams M. J., Veitch J., Messenger C., 2021, Phys. Rev. D, 103, 103006
Wong K. W. K., Isi M., Edwards T. D. P., 2023a, Fast gravitational

wave parameter estimation without compromises (arXiv:2302.05333), https://arxiv.org/abs/2302.05333
Wong K. W. k., Gabrié M., Foreman-Mackey D., 2023b, J. Open Source Softw., 8, 5021
Wouters T., Pang P. T. H., Dietrich T., Van Den Broeck C., 2024, Phys. Rev. D, 110, 083033
Yallup D., Handley W., Cabezas J., et al., 2025b, blackjax: A Library of Bayesian Inference Algorithms in JAX (Nested Sampling Branch), https://github.com/handley-lab/blackjax/tree/nested_sampling
Yallup D., Kroupa N., Handley W., 2025a, in Frontiers in Probabilistic Inference: Learning meets Sampling. https://openreview.net/forum?id=ekbkMSuPo4
Zackay B., Dai L., Venumadhav T., 2018, Relative Binning and Fast Likelihood Evaluation for Gravitational Wave Parameter Estimation (arXiv:1806.08792), https://arxiv.org/abs/1806.08792
ter Braak C. J. F., 2006, Statistics and Computing, 16, 239

APPENDIX A: PP COVERAGE PLOT FOR `BILBY` AND `DYNESTY`

For completeness, we present the PP coverage plot for the CPU-based `bilby+dynesty` analysis in Figure A1. This serves as a reference for the corresponding plot for our `blackjax-ns` implementation shown in the main text.

As expected for two functionally equivalent samplers, the resulting PP plot and p-values demonstrate results consistent with those from our GPU-based framework. We note, however, that while the two samplers are algorithmically analogous, apart from differences in the parallelisation strategy, they will not produce identical PP plots due to their stochastic nature and the inherent uncertainties in posterior reconstruction from nested sampling.

The primary sources of this variance include the statistical uncertainty in the weights assigned to the dead points, which are themselves estimates (Skilling 2006; Higson et al. 2018), and an additional uncertainty inherent to parameter estimation with nested sampling that is not captured by standard error propagation methods (Prathaban & Handley 2024). Consequently, the PP curves and their associated p-values are themselves subject to statistical fluctuations. The results from the two frameworks are therefore only expected to be consistent within these intrinsic uncertainties. We do not explicitly account for these uncertainties here, instead leaving this analysis for future work.

APPENDIX B: FULL RUN STATISTICS FOR INJECTION STUDY

We present a detailed comparison of the internal sampling statistics for the 100-injection study in Figure B1. We tuned the 'delay' parameter in the walk length tuning formula to target a similar level of sample decorrelation to `bilby`. The results show that this was successful, and both samplers ended up performing similar numbers of likelihood evaluations, with similar acceptance rates. Figure B1 shows the full statistics for each injection. Although we had to employ batch tuning to the chains, in contrast to the per-iteration strategy employed in `bilby`, the results show that this still resulted in similar behaviours for the chains and thus our results represent a like-for-like comparison.

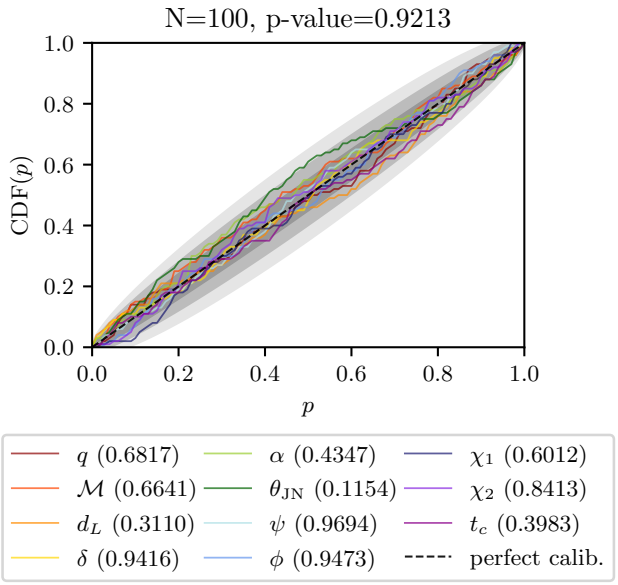

Figure A1. PP coverage plot for the 100-injection study, obtained with the CPU-based `bilby+dynesty` sampler. This plot is provided for direct comparison with the results from our `blackjax-ns` implementation shown in Figure 7. The results confirm that the CPU-based and GPU-based implementations are functionally similar.

APPENDIX C: LIKELIHOOD LEVEL PARALLELISATION PERFORMANCE

The performance gains from a GPU sampling framework arises from the ability to exploit massive parallelism. In particular, we seek to exploit parallelism across multiple short Markov chains, as this is the central construction of NS. The potential bottlenecks for parallelism arise from the NS algorithms ability to effectively synchronise the chains, and (if this condition is met) the ability to parallelise the likelihood evaluation itself across these chains. Developing algorithms that can effectively achieve the former goal is part of the ongoing programme of research this work establishes on a GPU GW problem. Regardless of effective parallelism across chains, the algorithm in this work is bottlenecked by the ability to parallelise the likelihood evaluation itself. We demonstrate the scaling of the likelihood evaluation time with the number of samples in Figure C1. The emerging pattern of polynomial runtime scaling with the number of samples, particularly for the *Full* bin likelihood we investigate primarily in this work, indicates that the parallel throughput of the GPU is being saturated. This bottleneck can be mitigated by either; using a compressed likelihood (*Heterodyning* being an example of this shown in the figure), expanding processing over multiple GPUs, or, if possible, rewriting the likelihood to be more parallelism friendly.

This paper has been typeset from a TEX/LATEX file prepared by the author.

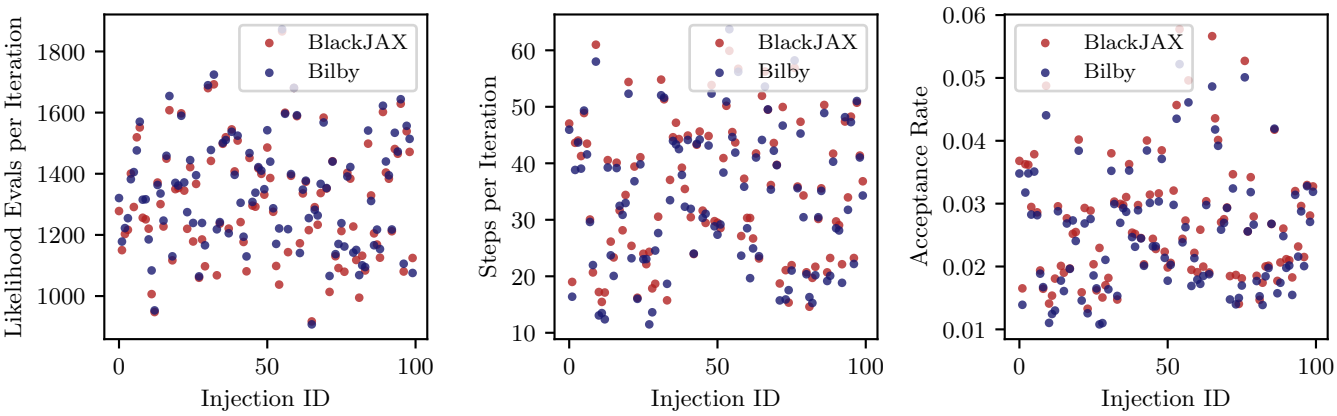

Figure B1. Comparison of internal run statistics for the 100-injection study. From left to right: the mean number of likelihood evaluations per iteration, the mean number of accepted steps, and the mean acceptance rate, for both the `bilby` and `blackjax-ns` runs. The results show that despite the differences in tuning strategy, motivated by architectural differences between the CPU and GPU, the two samplers exhibit similar properties.

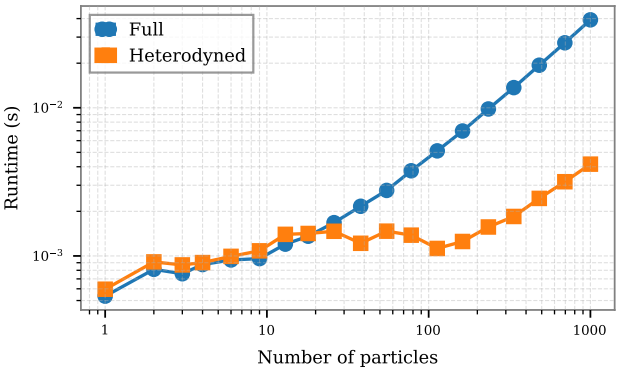

Figure C1. Scaling of likelihood wallclock time on an NVIDIA L4 GPU. A typical representative `jimgw` likelihood for H1 and L1 detectors on a 4 second window using the `IMRPhenomD` waveform model from `ripple` is used. Calls of likelihood vectorized over N samples are made, and the wall time averaged over 50 calls is reported. Comparison is made between the *Full* likelihood using all frequency bins, and a *Heterodyned* likelihood compressing to 300 frequency bins.