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

Metha Prathaban,[2,3,4]★ David Yallup,[1,2] James Alvey[1,2,5] and Will Handley[1,2,5]
[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*
[4] *Pembroke College, University of Cambridge, Cambridge, CB2 1RF, UK*
[5] *Gonville and Caius College, University of Cambridge, Cambridge, CB2 1TA, UK*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

We present a GPU-accelerated implementation of 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 wall-time speedups of up to two orders of magnitude 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, thereby creating a vital reference against which future, more advanced parallel sampling algorithms can be rigorously assessed. Our results not only demonstrate significant speedups but also serve to decouple the effects of hardware performance from algorithmic innovation.

Key words: keyword1 – keyword2 – keyword3

1 INTRODUCTION

The era of gravitational wave (GW) astronomy, initiated by the landmark detections of 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 are 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). The community-standard software tools, the inference library `bilby` (Ashton et al. 2019) paired with a custom implementation of the nested sampler `dynesty` (Speagle 2020), have 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 time-consuming and creating a significant 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 GPU-accelerated Markov Chain Monte Carlo (MCMC) samplers like `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 are achievable for GW parameter estimation. While these MCMC-based approaches excel at rapidly generating samples from the posterior, they do not directly compute the Bayesian evidence, which remains key for robust model selection.

In this paper, we introduce a GPU-accelerated nested sampling 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. We leverage the `blackjax-ns` sampler, a recent tool designed for vectorized execution on GPUs (Yallup et al. 2025). This sampler, part of the `blackjax` library, is a novel reformulation of the core nested sampling algorithm for massive parallelization (Cabezas et al. 2024).

Instead of its default slice sampler, we developed 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.

★ E-mail: myp23@cam.ac.uk (MP)

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); 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 source parameters, θ, under a specific hypothesis, H, is given by Bayes' theorem (Bayes 1763):

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

Here, the posterior, $p(\theta|d,H)$, 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 $p(\theta|d,H)$ 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.

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 (NS) 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).

The algorithm starts by drawing a population of N 'live points' from the prior distribution, $\pi(\theta)$. It then proceeds iteratively. In each iteration, the live point with the lowest likelihood value, $\mathcal{L}_{\rm min}$, 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., $\mathcal{L}_{\rm new} > \mathcal{L}_{\rm min}$. 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 'inner sampling' task is a critical determinant of the sampler's overall efficiency and robustness.

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.

This architecture makes GPUs exceptionally effective for problems that can be expressed in a Single Instruction, Multiple Data (SIMD) paradigm. 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.

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. recently developed a vectorized formulation of the NS algorithm, specifically designed for highly parallel execution within the blackjax framework (Yallup et al. 2025; Cabezas et al. 2024).

One of the core innovations of this approach is the introduction 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. 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 constraint, $\mathcal{L} > \mathcal{L}_{\rm min}$, where $\mathcal{L}_{\rm min}$ 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. While the original work proposed a specific inner sampling kernel for this task, the vectorized framework itself is general. It provides a structure

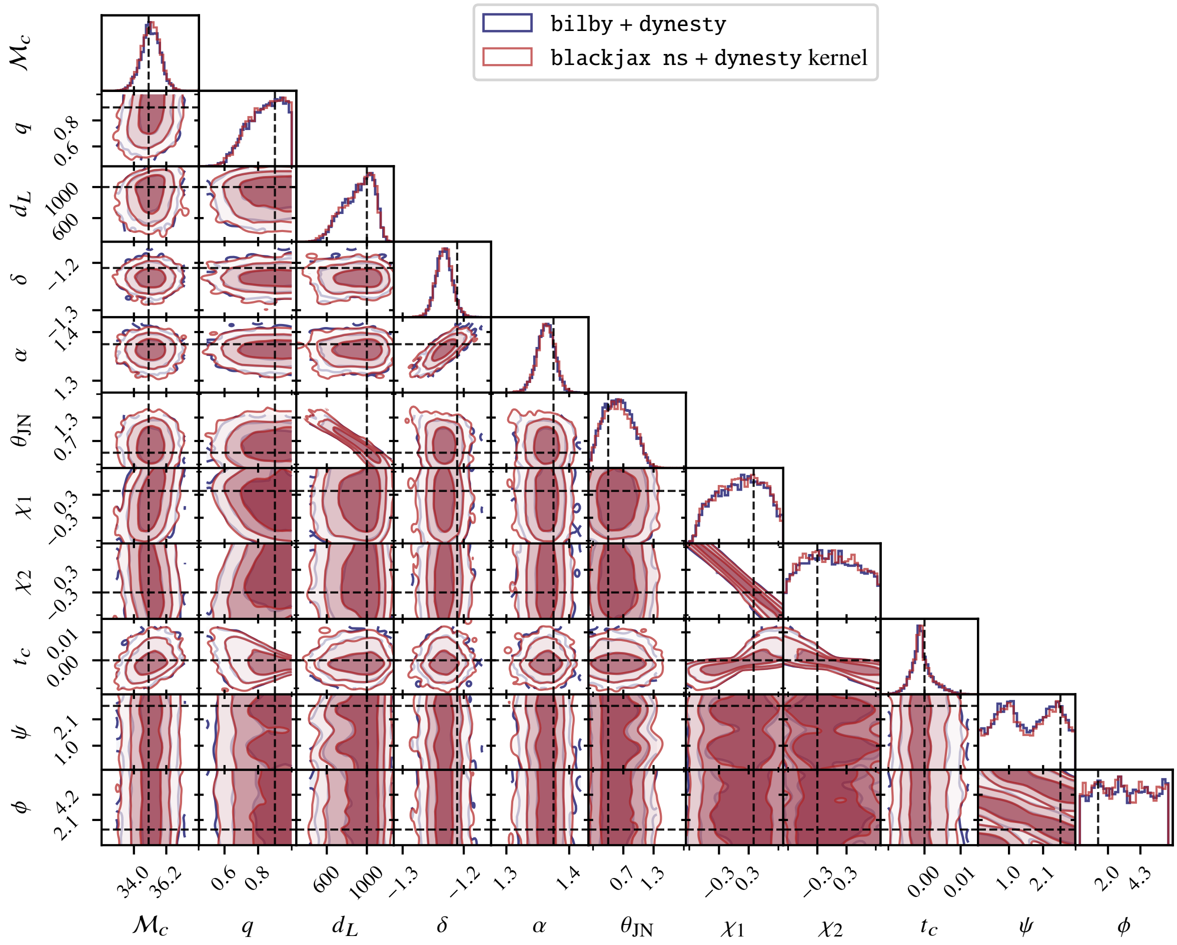

Figure 1. Recovered posteriors for the 4s signal. The posteriors are in agreement with each other, demonstrating that `blackjax-ns` implementation with our custom kernel is functionally equivalent to the `bilby` + `dynesty` implementation.

within which any suitable inner sampling method can be deployed in parallel.

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 1.0 or drawn from a gamma distribution. 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 walk length is adaptive on a per-iteration basis; the algorithm adjusts the number of MCMC steps dynamically to target a pre-defined num-

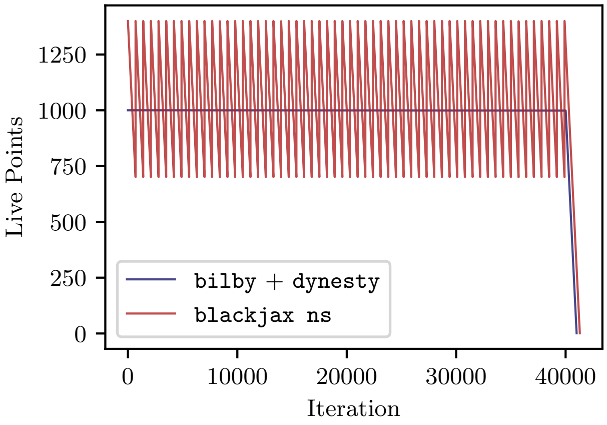

Figure 2. Comparison of the number of live points in the sequential CPU and batched GPU implementations. Although the nominal number of live points used in `blackjax-ns` is higher than in `bilby`, the saw-tooth pattern means that the effective number of live points is the same.

ber 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.

While this batch-adaptive approach is essential for efficient GPU vectorization, it introduces some important differences. In the sequential CPU algorithm, an individual MCMC walk that proves to be an outlier with a low acceptance rate will result in a longer walk only for the iteration after it. In our parallel algorithm, if a subset of points in a batch has a low acceptance rate, the average rate will be reduced, causing the walk length for the entire next batch to increase. The converse is also true: if a subset of points in a batch has a particularly high acceptance rate, the average rate will be increased, causing the walk length for the entire next batch to decrease. This can often lead to a very different total number of likelihood evaluations compared to the sequential counterpart. We discuss this further in Sections 3.2 and 4 but depending on the evidence, the likelihood evaluations can be higher or lower, and may need further tuning. Despite this architectural modification, our kernel is designed to be functionally analogous to the trusted `bilby` sampler, operating within the same unit hypercube space and utilizing the same DE proposal strategy to explore the parameter space. Even in cases where the GPU-based implementation performs more likelihood evaluations, it is significantly faster than its CPU-based counterpart.

3.2 Sampler configuration and settings

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 run 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.

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 and more global tuning means that the standard settings for `bilby` parameters such as `naccept` may no longer be optimal. For instance, on high-SNR signals which require many nested sampling iterations, the walk length can adapt to become overly long. Conversely, on low-SNR signals that converge quickly, the sampler has fewer opportunities to adapt, potentially resulting in a walk length that is shorter than optimal. We discuss this behaviour and provide more guidance on parameter settings in Sec. 4.

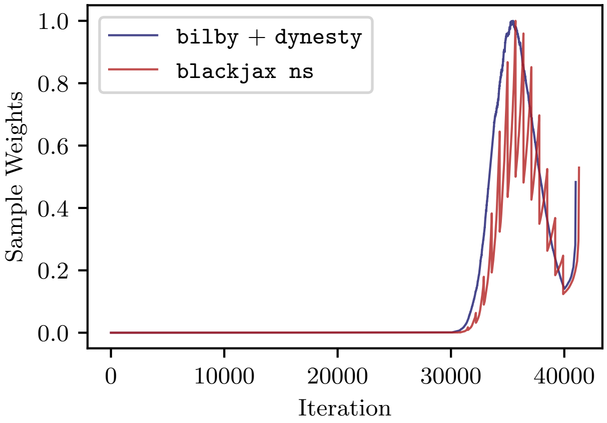

Figure 3. Comparison of the sample weights for each dead point for the sequential CPU and batched GPU implementations. The weights are calculated using the prior volumes enclosed between successive dead points. The shapes are similar, and both implementations enter the bulk of the posterior distribution at similar iterations, indicating 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.

Another subtle but critical consequence of this architectural shift 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_{live} to $n_{\text{live}} - k$ over one cycle of k iterations before being replenished (Fig. 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.

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_{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)

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

where n_i is the number of live points at iteration i. In the CPU-based implementation, $n_i \approx n_{\text{CPU}}$ is constant. After k iterations, 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` iterations. The total log volume shrinkage over one such cycle is the sum over the decreasing number of live points:

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

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

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

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

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

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. 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). This allows the waveform to be calculated in parallel across the frequency domain, 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 `ripple` library too.

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.

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.

[h!]

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 injection and are specified in Section 4.

Parameter	**Description**	**Prior Distribution**	**Range**
$\mathcal{M}_c$	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
$\theta_{\rm JN}$	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

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.

4 RESULTS AND DISCUSSION

We now validate the performance of our framework through a series of analyses. In all of the below 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 filtered 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 uniform 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

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

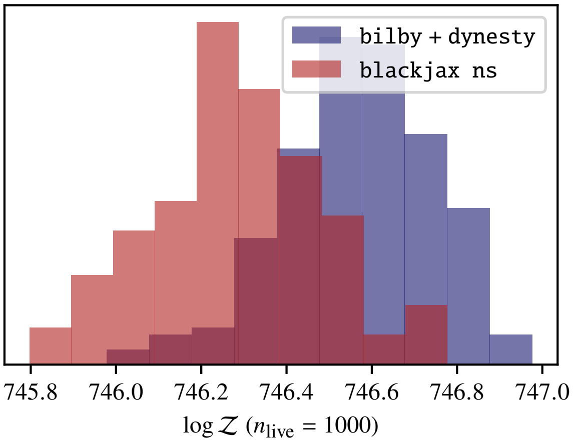

Figure 4. Comparison of the log evidence for the 4s signal. The results are in excellent agreement, demonstrating the robustness of the blackjax-ns implementation in recovering the same posteriors and evidence as the bilby implementation. This unifies parameter estimation and evidence evaluation into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

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 0.8 hours. This corresponds to a wall-time speedup factor of ×60. In this case, the batch-adaptive nature of the GPU sampler led to a slightly higher number of likelihood evaluations (70.6 million) compared to the sequential CPU sampler (62.9 million).

Beyond wall-time, we also consider the relative financial cost. Based on commercial on-demand rates from Google Cloud, the rental cost for the 16-core CPU instance and the 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 ×3.7 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, resulting in a lower runtime. Interestingly, however, when comparing the relative commercial hourly rates of the two GPUs, the L4 GPU analysis was actually cheaper. We summarise these results in Table 3.

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

Implementation + Hardware	Wall-time (h)	Speedup	Cost Saving
bilby (16 Icelake CPU cores)	47.8	-	-
blackjax-ns (NVIDIA L4)	0.8	60×	3.7×
blackjax-ns (NVIDIA A100)	0.6	80×	5×

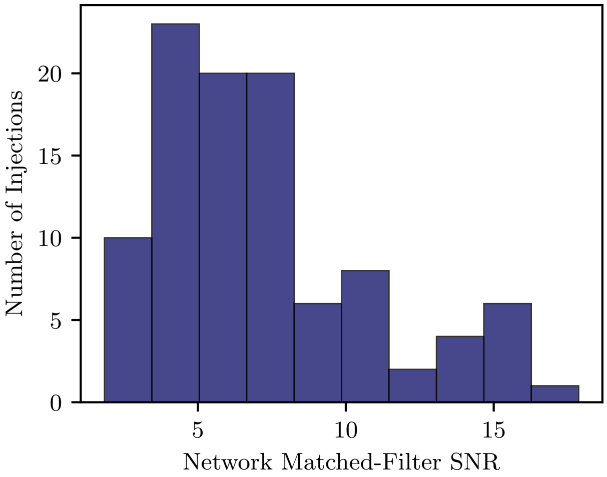

Figure 5. Distribution of the network signal-to-noise ratios (SNR) for the injected signals.

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 components 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 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.

In the first instance, both samplers were configured with naccept = 60 and maxmcmc = 5000, and the blackjax-ns sampler used a batch size of num_delete = 700. The posterior distributions were in good agreement with each other and the percentile-percentile (PP) plot was unbiased. However, due to the low SNR and the algorithmic differences arising from parallelisation, we found that the mean accepted steps per iteration was lower in blackjax-ns compared to bilby, leading to an unfair comparison in the runtimes. We therefore

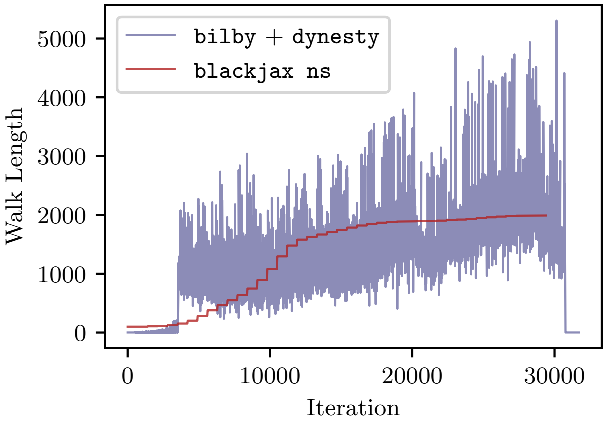

Figure 6. Comparison of the chain lengths for the sequential CPU and batched GPU implementations. The `blackjax-ns` implementation can only perform batch tuning, a fundamental algorithmic difference between the CPU-based and GPU-based implementations of the 'acceptance-walk' kernel. This leads to a staircase pattern in the chain lengths, which can be seen in the plot. However, setting `naccept = 120` in `blackjax-ns` and `naccept = 60` in `bilby` results in similar average chain lengths for the injection study.

applied more aggressive tuning to the `blackjax-ns` chain lengths to compensate for the limited number of opportunities to perform the tuning, and re-ran the analysis. In practice, this was achieved by setting `naccept = 120` for the `blackjax-ns` runs. This resulted in a similar number of likelihood evaluations and accepted steps per iteration to `bilby` (see Figure 6), enabling a fairer and more direct comparison of the two implementations. For low SNR signals, we therefore recommend setting `naccept` to a higher value to ensure similar run properties to `bilby`.

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. 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. TODO: add link/ref

The average chain length for the `blackjax-ns` sampler was 1376, consistent with the average of 1337 for the `bilby+dynesty` sampler. A more detailed breakdown of all run statistics is provided in Appendix B. In terms of wall-time, the `blackjax-ns` sampler completed the analyses in an average of 0.4 hours per injection, with individual run times ranging from 9 to 56 minutes. In contrast, the CPU-based runs required an average of 26.4 total CPU-hours, with some runs taking as long as 100 hours. Figure 8 shows the distribution of the resulting wall-time speedup factors, which have a mean value of ×81.2. 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 ×5.1 for the GPU-based analysis compared to its CPU-based counterpart.

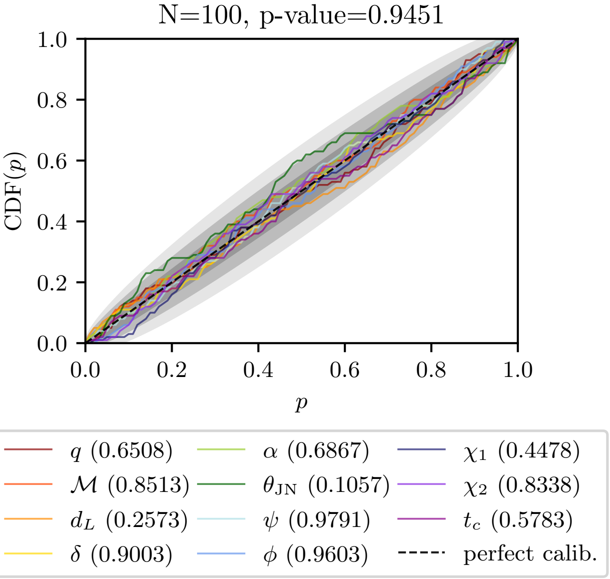

Figure 7. Percentile-percentile (PP) coverage plot for the 100-injection study, obtained with the `blackjax-ns` sampler. The cumulative fraction of events where the true injected parameter falls below a given credible level is plotted against that credible level. The proximity of all parameter curves 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.

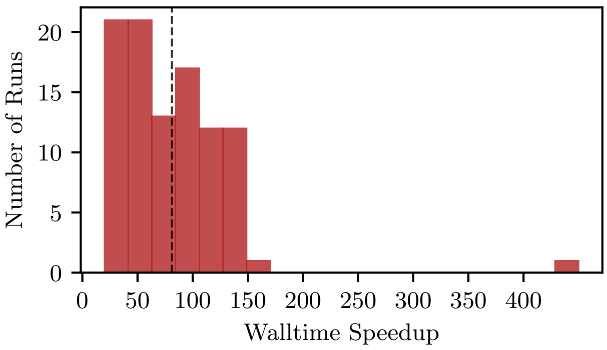

Figure 8. The wall-time speedups for all 100 events in the injection study.

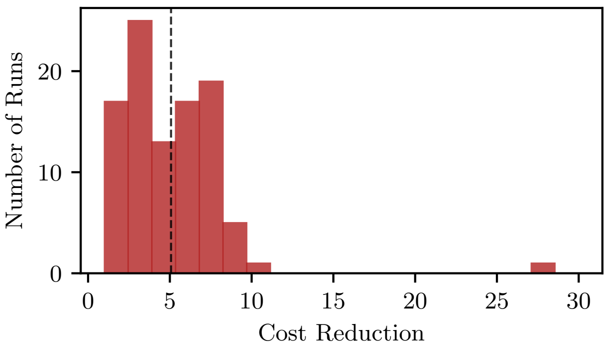

Figure 9. The cost reductions for all 100 events in the injection study.

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 $n_{\rm live}$ applied in our main parallel analyses. 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 10.44 hours. This represents a speedup of ×2.9. Finally, the fully parallel run, using its correctly scaled live point count of 1400 and a batch size of `num_delete` = 700, completed in 0.45 hours. This corresponds to a further speedup of ×23.2 over the sequential-GPU run.

This result demonstrates that while a GPU-native likelihood provides a significant performance benefit, the dominant contribution to the overall speedup (a total of ×67.6 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. (2025).

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 $n_{\rm live} = 1400$ for `blackjax-ns` and $n_{\rm live} = 1000$ for `bilby`, and an `naccept` target of 120 and 60, respectively. In this case, we revert to the default termination condition described in Section 4.1.1.

The recovered posterior distributions and log-evidence values are shown in Figure 10 and Figure 11. 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 6.37 hours, corresponding to a wall-time speedup factor of ×26.2. This was achieved despite performing 140 million likelihood evaluations, more than double that of the CPU run. This increase is attributable to longer average MCMC chain lengths in the GPU sampler, a consequence of the batch-adaptive tuning strategy. In hindsight, we could have reduced the `naccept` parameter from 120 to 60 to match the `bilby` run better for this signal. The associated cost-reduction factor for this analysis was ×1.64.

Table 4. Injection parameters for the 8s signal.

Parameter	Value
$\mathcal{M}$	10.0 $\mathrm{M}_\odot$
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

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 only have taken roughly double the time of the 4-second signal analysis, or $0.8 \times 2 = 1.6$ hours. However, we observe that the runtime increased significantly more than what would be predicted by the increase in 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 widely used and trusted algorithm within the gravitational-wave community's standard `bilby` and `dynesty` framework. This particular kernel was chosen as its structure can be adapted to use a uniform MCMC walk length across all parallel processes in a batch, a critical feature for avoiding thread divergence and ensuring efficient execution on GPU hardware. Our implementation leverages the vectorized nested sampling architec-

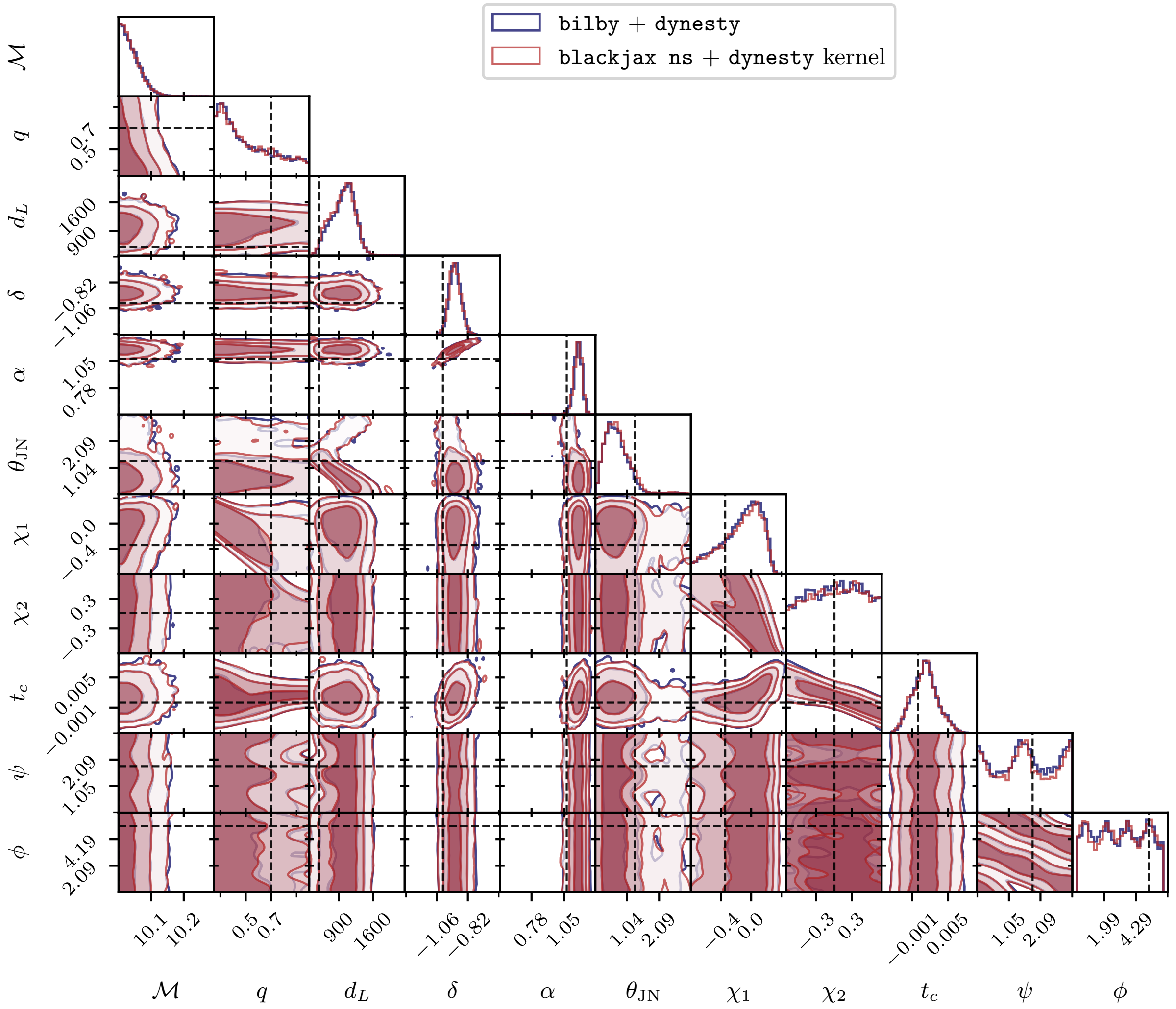

Figure 10. Recovered posterior distributions for the 8s simulated signal, comparing our GPU-based `blackjax-ns` sampler with the CPU-based `bilby` sampler. The injected values are marked by black lines. The strong statistical agreement confirms the validity of our implementation for longer-duration signals. Despite requiring more likelihood evaluations for this analysis, the GPU implementation still provided a wall-time speedup of 26.2× and a cost reduction of 1.64×.

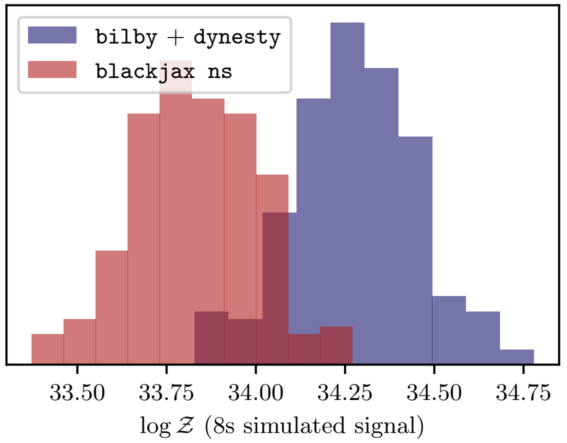

Figure 11. Comparison of the recovered log-evidence (Z) for the 8s signal. The results from both the `bilby` and `blackjax-ns` frameworks are consistent within their estimated numerical uncertainties.

ture of `blackjax-ns` to provide a tool for rapid, unified Bayesian parameter estimation and model selection.

Through systematic studies of simulated binary black hole signals, we have demonstrated that our implementation is functionally analogous to the trusted CPU-based framework, producing statistically consistent posterior distributions and evidence estimates. The architectural shift to the GPU yields substantial performance gains, with typical wall-time speedups of ×60-100 and cost reductions of ×4-5 for the problems studied here.

However, the contributions of this work extend beyond the immediate performance gains demonstrated in our analyses. By developing and validating a GPU-native implementation of the trusted `bilby` ‘acceptance-walk’ kernel, we address two fundamental points regarding the future of gravitational-wave inference.

First, this work represents a necessary step in future-proofing the community’s core analysis tools. The trajectory of high-performance computing is increasingly skewed towards GPU-centric architectures, a trend significantly accelerated by the proliferation of artificial intelligence and machine learning. The migration of its foundational algorithms to these parallel architectures is therefore a necessary evolution to ensure they remain computationally viable with increasing data volumes. Our work provides a direct and robust pathway for this transition, ensuring that a trusted, well-understood algorithm remains viable on next-generation hardware.

Second, 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. New kernels can now be assessed on their own algorithmic merit, beyond the inherent speedup of the GPU. This work enables a true, like-for-like comparison between different GPU-based samplers, which will be useful for guiding the development of the next generation of 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 fixed MCMC walk length of the 'acceptance-walk' kernel is advantageous for parallelisation, our results highlight a key challenge: its adaptive tuning strategy, designed for serial execution, translates sub-optimally to the batched GPU paradigm. This necessitates manual adjustment of the sampler settings for different problems, as the kernel is not natively optimised for this 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. (2025), which are better suited to GPUs and may offer more robust and efficient performance without problem-specific tuning.

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`. On the software side, future work will involve integrating additional features from `bilby` 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 nested sampling algorithm, such as those in Williams et al. (2021) and Prathaban et al. (2025), 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 the operational requirements of these models. 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

The authors would like to thank Ming Yang and Will Templeton for their invaluable help in the development and testing of early iterations of the code applying `blackjax-ns` to gravitational-wave inference. MP was supported by the Harding Distinguished Postgraduate Scholars Programme (HDPSP). This work was partly 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.

DATA AVAILABILITY

TODO: Add data availability statement.

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
Bayes T., 1763, Philosophical Transactions of the Royal Society of London, 53, 370
Branchesi M., et al., 2023, Journal of Cosmology and Astroparticle Physics, 2023, 068
Cabezas A., Corenflos A., Lao J., Louf R., 2024, BlackJAX: Composable Bayesian inference in JAX (`arXiv:2402.10797`)
Cornish N. J., 2013, Fast Fisher Matrices and Lazy Likelihoods (`arXiv:1007.4820`), `https://arxiv.org/abs/1007.4820`
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
Higson E., Handley W., Hobson M., Lasenby A., 2018, Bayesian Analysis, 13, 873
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`
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
Prathaban M., Handley W., 2024, Monthly Notices of the Royal Astronomical Society, 533, 1839
Prathaban M., Bevins H., Handley W., 2025, Monthly Notices of the Royal Astronomical Society, p. staf962
Romero-Shaw I. M., et al., 2020, Monthly Notices of the Royal Astronomical Society, 499, 3295
Skilling J., 2006, Bayesian Analysis, 1, 833

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

Yallup D., Kroupa N., Handley W., 2025, 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 results 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.

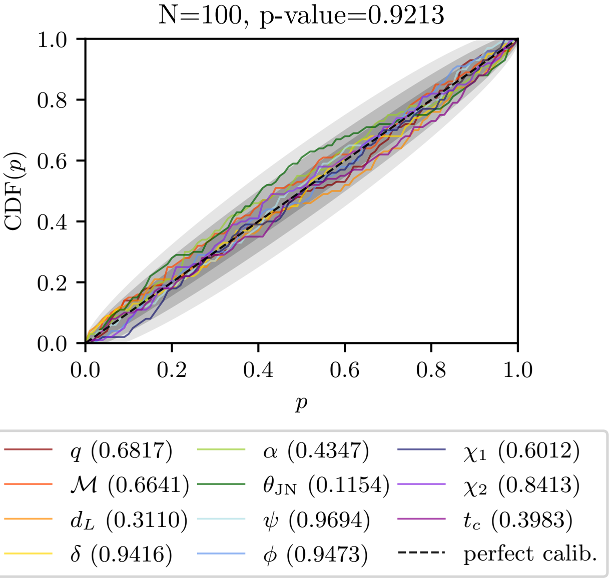

Figure A1. PP coverage plot for the 100-injection study, obtained with the CPU-based `bilby+dynesty` sampler. This plot is provided for direct comparison with the results from our `blackjax-ns` implementation shown in Figure 7. The results confirm that the CPU-based and GPU-based implementations are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

APPENDIX 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. While the performance metrics such as acceptance rate of the walks are broadly consistent between the two frameworks, we observe instances where the `blackjax-ns` sampler produces significantly longer MCMC chains paired with lower acceptance rates.

This behaviour is a direct consequence of the batch-adaptive tuning strategy, which is fundamental to the GPU implementation. As the nested sampling run progresses, the live points can transition into regions of the parameter space with a very different local likelihood geometry, such that there is a sudden shift in the difficulty of making moves that are accepted. The sequential CPU-based sampler can adapt to this change immediately by adjusting the walk length for the very next point. In contrast, the batch-based GPU sampler only tunes its walk length at the end of a full batch cycle.

If the sampler enters a challenging region shortly after a tuning step, it must continue with a now-suboptimal (too short) walk length for many iterations. This results in a sharp drop in the acceptance rate for that batch. When the tuning is eventually performed, the low average acceptance rate of the preceding batch triggers a significant, compensatory increase in the walk length for the subsequent batch. This mechanism explains the observed correlation between runs with very low acceptance rates and long MCMC chains.

This trade-off between adaptive responsiveness and parallel efficiency is an inherent characteristic of the batch-based design. Users of this framework should therefore always verify that the mean number of accepted steps per iteration remains sufficiently high for accurate results. For the injection study presented in this work, we find that in most of the instances where this effect occurred, the compensatory increase in chain length was sufficient to maintain an adequate number of accepted steps. The primary impact was therefore on computational efficiency (total likelihood evaluations and wall time) rather than the quality of the posteriors.

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

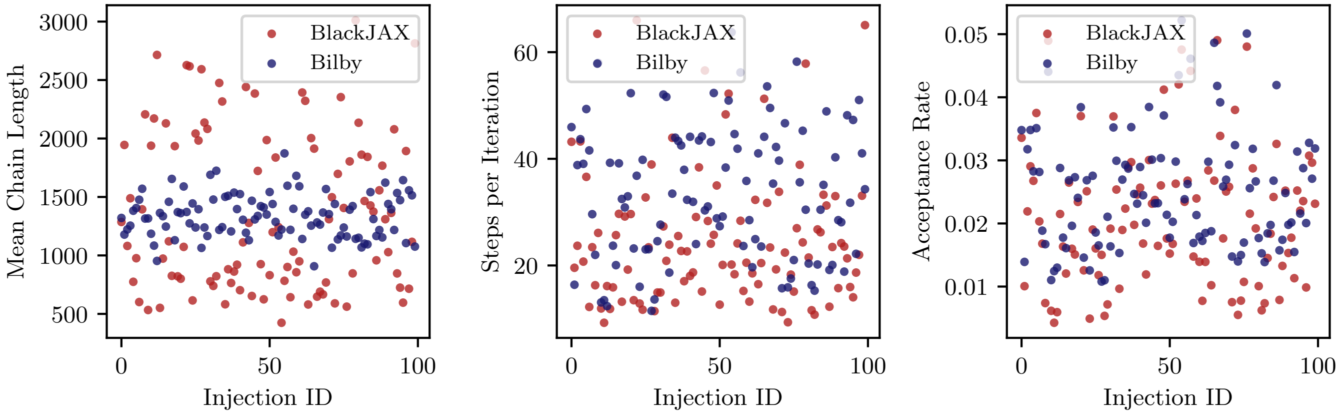

Figure B1. Comparison of internal run statistics for the 100-injection study. From left to right: the mean number of MCMC steps (walk length) per iteration, the mean number of accepted steps, and the mean acceptance rate, for both the `bilby` and `blackjax-ns` runs. The batch-adaptive nature of the GPU sampler can lead to outlier runs with longer chains and lower acceptance rates, as discussed in Appendix B.