











SGNL: Scalable Low-Latency Gravitational Wave Detection Pipeline for Compact Binary Mergers

Yun-Jing Huang ^{1,2,*} Chad Hanna ^{1,2,3,4} Leo Tsukada ^{5,6} Amanda Baylor ⁷ Patrick Godwin ^{8,1,2} Prathamesh Joshi ^{1,2,9} James Kennington ^{1,2} Cody Messick ⁷ Surabhi Sachdev ⁹ Ron Tapia^{1,4} and Zach Yarbrough ¹⁰

¹*Department of Physics, The Pennsylvania State University, University Park, PA 16802, USA*

²*Institute for Gravitation and the Cosmos, The Pennsylvania State University, University Park, PA 16802, USA*

³*Department of Astronomy and Astrophysics, The Pennsylvania State University, University Park, PA 16802, USA*

⁴*Institute for Computational and Data Sciences, The Pennsylvania State University, University Park, PA 16802, USA*

⁵*Department of Physics and Astronomy, University of Nevada,*

Las Vegas, 4505 South Maryland Parkway, Las Vegas, NV 89154, USA

⁶*Nevada Center for Astrophysics, University of Nevada, Las Vegas, Las Vegas, NV 89154, USA*

⁷*Leonard E. Parker Center for Gravitation, Cosmology, and Astrophysics,*

University of Wisconsin-Milwaukee, Milwaukee, WI 53201, USA

⁸*LIGO Laboratory, California Institute of Technology, MS 100-36, Pasadena, California 91125, USA*

⁹*School of Physics, Georgia Institute of Technology, Atlanta, GA 30332, USA*

¹⁰*Department of Physics and Astronomy, Louisiana State University, Baton Rouge, LA 70803, USA*

We present SGNL, a scalable, low-latency gravitational-wave search pipeline. It reimplements the core matched-filtering principles of the GstLAL pipeline within a modernized framework. The Streaming Graph Navigator library, a lightweight Python streaming framework, replaces GstLAL’s GStreamer infrastructure, simplifying pipeline construction and enabling flexible, modular graph design. The filtering core is reimplemented in PyTorch, allowing SGNL to leverage GPU acceleration for improved computational scalability. We describe the pipeline architecture and introduce a novel implementation of the Low-Latency Online Inspiral Detection algorithm in which components are pre-synchronized to reduce latency. Results from a 40-day Mock Data Challenge show that SGNL’s event recovery and sensitivity are consistent with GstLAL’s within statistical and systematic uncertainties. Notably, SGNL achieves a median latency of 5.4 seconds, a 42% reduction compared to GstLAL’s 9.3 seconds.

I. INTRODUCTION

The fourth Gravitational-Wave Transient Catalog (GWTC-4.0) [1] now includes more than 200 candidate events observed by the LIGO–Virgo–KAGRA (LVK) Collaboration [2–4], combining discoveries from previous observing runs [5–7] and the first part of the fourth run (O4), and illustrating the continued increase in detection rates. During the ongoing O4 run, over 200 significant alerts have already been issued in low latency [8]. As detector sensitivities improve and the global detector network expands [4, 9, 10], the rate of real-time detections is expected to grow further, highlighting the need for pipelines that are fast, scalable, and sustainable over the long term.

The GstLAL [11–16] low-latency search pipeline has been central to LVK observing runs, including performing the first low-latency detection of GW170817 [17], and it continues to identify low-latency events throughout O4 [8]. GstLAL is a stream-based matched-filtering pipeline [11, 18] in which data are processed via GStreamer [19], a C-based multimedia framework. Its core matched-filtering engine employs the Low-Latency Online Inspiral Detection (LLOID) algorithm [18], which splits templates into downsampled time slices and performs singular value

decomposition (SVD) on groups of templates. The full pipeline, including data handling and LLOID filtering, is constructed by connecting GStreamer elements into a processing graph. While GStreamer is powerful and robust, its C-based architecture makes the pipeline difficult to extend and maintain, and limits portability to modern computing hardware such as graphics processing units (GPUs).

The SGNL [20] pipeline implements these ideas within a modern, modular, and sustainable framework. It replaces GstLAL’s GStreamer infrastructure with the Streaming Graph Navigator (SGN) library [21], a lightweight Python framework for constructing real-time dataflow graphs. SGN organizes computation into modular elements, enabling flexible pipeline design, precise synchronization, and efficient utilization of CPU and GPU resources. At its core, matched filtering is implemented in PyTorch, allowing GPU acceleration and batch operations across multiple detectors and template banks, which has been shown to scale effectively on GPUs [22]. This combination of a Python-based streaming architecture and GPU-accelerated computation provides a maintainable and extensible platform for low-latency gravitational-wave detection.

In this paper, we present the design and validation of the SGNL pipeline. Section II details the architecture of the SGNL online inspiral analysis, including data ingestion, whitening, the reimplementations of the LLOID

* yun-jing.huang@ligo.org

filtering algorithm, and event identification. We introduce a novel pre-synchronization scheme that prevents additional latency during multirate reconstruction. Section III evaluates SGNL’s performance using a 40-day Mock Data Challenge (MDC), comparing its sensitivity, event recovery, and latency against GstLAL. Section IV concludes with a summary and outlook for future developments.

II. PIPELINE DESCRIPTION

In this section, we describe the SGNL online inspiral pipeline and its key components. In Section IIA, we discuss the Python streaming framework (SGN) on which SGNL is built. Section IIB covers data ingestion and conditioning, including the handling of external data, power spectral density (PSD) estimation and whitening, and the removal of invalid segments and glitches. In Section IIC, we describe the LLOID algorithm and its implementation in SGNL, including a novel pre-synchronization technique to avoid additional latency. Section IID explains event identification, covering signal-to-noise ratio (SNR) peak detection, coincidence formation, significance estimation, background collection, and output of event candidates. Fig. 1 illustrates the overall workflow of the SGNL online inspiral pipeline. The offline pipeline closely mirrors the online pipeline through filtering and trigger generation, but differs in upstream data access, LR/FAR computation, and the low-latency alerting infrastructure, which are specific to the online pipeline.

A. Streaming Graph Navigator

The SGNL pipeline is built on the lightweight Python streaming framework SGN. In contrast, the GstLAL pipeline uses GStreamer, a C-based multimedia framework. SGN adapts GStreamer’s core ideas into a simpler and more flexible form for Python. It organizes computation into modular elements connected through source and sink pads. Pipelines form directed acyclic graphs where data flows continuously between stages. This design makes complex workflows easy to define and efficient to run in real time.

The SGN-TS [23] extension builds on this framework. It adds explicit notions of time and time series tracking. It also provides tools for buffering data and performing signal processing tasks such as resampling, correlation, synchronization, and stream combination. These features support a wide range of streaming applications. They are especially useful for gravitational-wave analysis, where precise alignment and coincidence of detector signals are crucial.

The SGN-LIGO [24] extension builds further on this system. It integrates LVK software such as LALSuite [25] by wrapping standard tools as SGN elements. This al-

lows gravitational-wave analysis routines to run directly inside the streaming pipeline. It removes the need for separate preprocessing steps or manual data conversions. The result is a simpler and more unified system where all components operate together in real time.

At the top level, SGNL assembles these components into a complete gravitational-wave search pipeline. Each stage, including data acquisition, conditioning, whitening, matched filtering, and event identification, is implemented as a connected network of SGN elements. Data flows continuously from input to output. This modular design balances flexibility and performance. It also makes the pipeline easy to extend by adding or modifying elements without disrupting other components.

B. Data ingestion and conditioning

1. Data ingestion

Gravitational-wave data from the LVK detectors are distributed in low latency to shared memory on the LIGO Data Grid in one-second chunks as Gravitational-Wave Frame (GWF) files, typically at a sample rate of 16384 Hz. These data contain both gravitational-wave strain and state vector channels, the latter providing data quality information from auxiliary detector channels. The SGNL low-latency pipeline begins with a source element that reads these GWF files and streams both channels to downstream elements in one-second buffers. Unlike GstLAL, which uses a separate source element for each detector, SGNL employs a single unified source element for all detectors. This design allows the source to re-establish synchronization whenever data delivery to shared memory is misaligned across detectors and prevents additional latency when detector streams fall out of sync.

Because low-latency data can experience network or delivery delays, the source element waits up to 60 seconds for missing data before inserting gap buffers to represent missing segments. During this period, it emits heartbeat buffers to maintain activity. After 60 seconds, it begins emitting one-second gap buffers at a one-second cadence to maintain a steady 60-second lag. When new data arrive, the source resumes normal output if the data are continuous. Otherwise, it continues emitting one-second gap buffers as quickly as the pipeline can process them until the discontinuity is fully bridged.

The state vector channel is also used to mask out strain data. Segments flagged as invalid are gated and replaced with gaps, ensuring that only valid data are passed downstream for analysis.

2. PSD estimation and whitening

The SGNL pipeline reimplements the GstLAL PSD estimation and whitening method [11, 26] in Python within

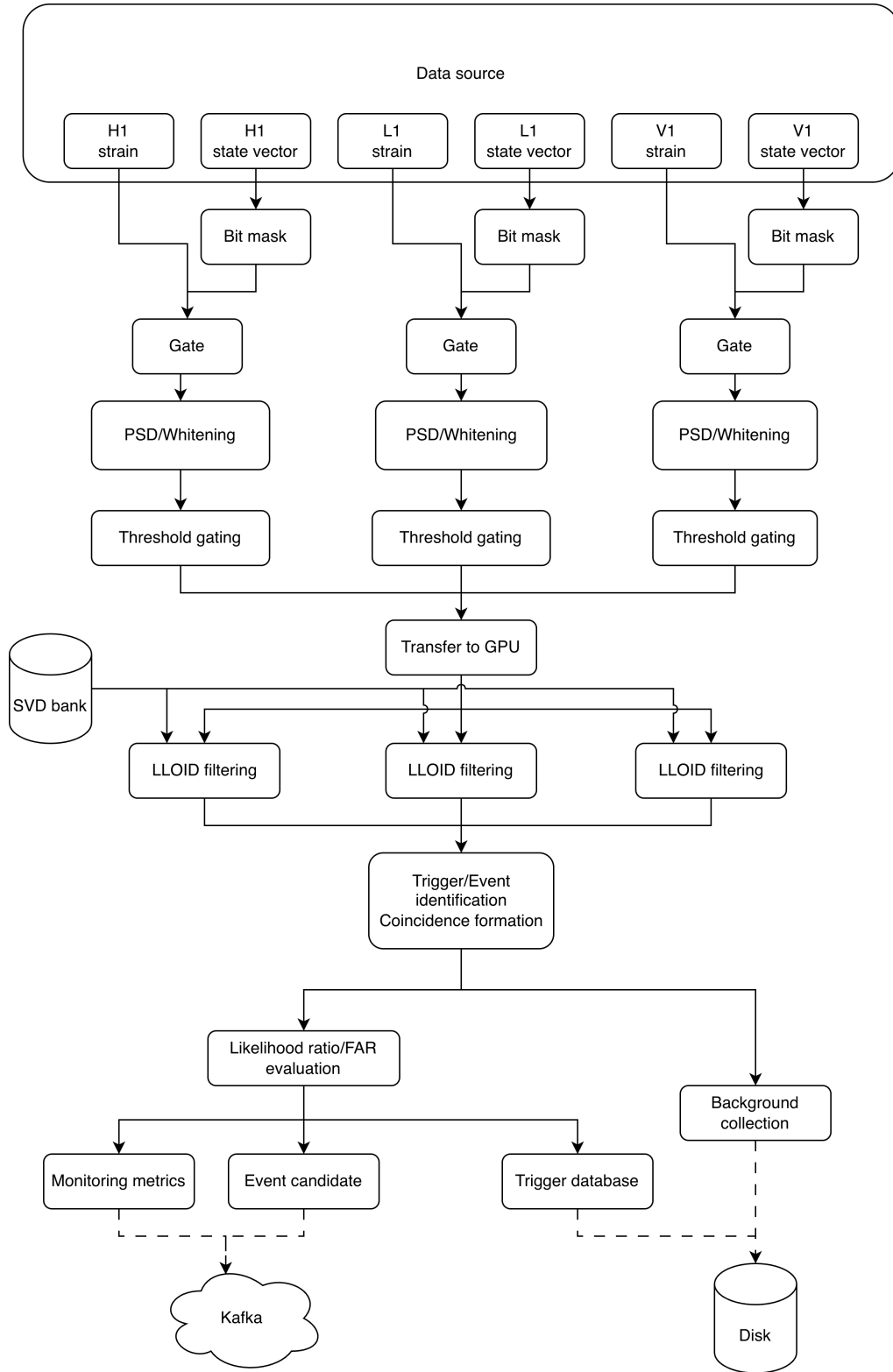


FIG. 1: Workflow of the SGNL online inspiral pipeline. Data from multiple detectors are read in a single source element, including both strain and state vector channels. Invalid segments flagged by the state vector are gated from the strain data. The PSD is estimated, and data are whitened. Noise transients above a threshold are gated out. Conditioned data are synchronized across detectors and transferred to the GPU if in GPU mode. Each detector's data are matched-filtered with a pregenerated SVD bank to produce SNR time series. SNR peaks are identified as triggers and combined into coincident events across detectors. Likelihood ratios and FAR are computed. Non-coincident triggers form background data and are saved to disk. The best events are sent to Kafka for aggregation and GraceDB alerts. Event candidates are saved to an SQLite database on disk.

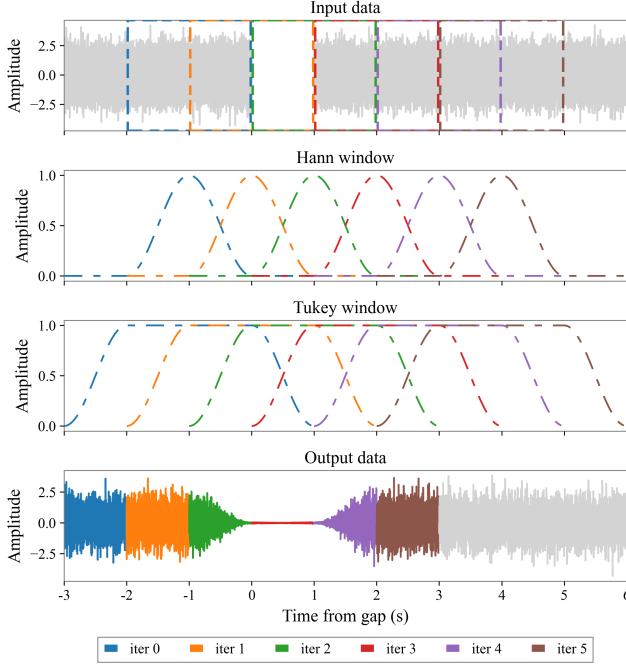


FIG. 2: Overlapping and windowing scheme used in the PSD estimation and whitening process, including the behavior around a gap in the input data. In this example, N is set to four seconds of samples, and Z is set to one second. The top panel shows a simulated white noise input stream with a one-second gap between 0 and 1 second. Colored rectangles indicate six consecutive processing iterations (blue, orange, green, red, purple, and brown). Each two-second input block is multiplied by a Hann window of the same color (second panel), zero-padded to form a four-second FFT block, and transformed to estimate the PSD. The second panel illustrates the overlapping Hann windows, which sum to unity, while the third panel shows the overlapping Tukey windows applied after whitening and inverse FFT. The final panel shows the whitened output of each iteration, each lagging the input by two seconds, corresponding to the two-second latency introduced by the four-second FFT length configuration.

an SGN element. PSD estimation and whitening are performed on fast Fourier transform (FFT) blocks of length N , constructed from overlapping input segments. Each input segment of length $N - 2Z$ is multiplied by a Hann window and zero-padded with Z points on each side to form a length- N block for the FFT computation. This reduces spectral leakage and ensures that overlapping windows sum to unity. The windowed blocks are Fourier transformed to produce instantaneous PSD estimates. For each frequency bin, the median of the most recent n_{med} PSD estimates is used to update a running geometric mean. Assuming stationary Gaussian noise, the PSD bins are χ^2 -distributed, so the geometric mean can be inferred from the median by dividing by a constant factor. To whiten the data, this geometric mean is converted to an arithmetic mean by multiplying by $\exp(\gamma)$, where γ is Euler's constant, giving the PSD estimate [11].

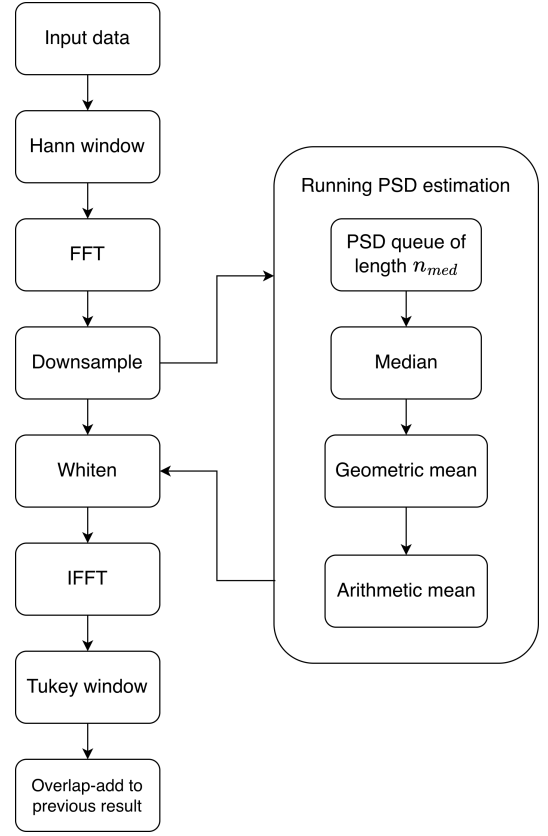


FIG. 3: Workflow of the whitening and PSD estimation in SGNL. Input data are first multiplied by a zero-padded Hann window, and an FFT is performed. Unlike GstLAL [26], downsampling is applied after the FFT, where high-frequency components are removed in the frequency domain. The resulting data are used both to generate the whitened output and to update the running PSD estimate. After whitening with the latest PSD, an inverse FFT is applied, followed by a Tukey window. The processed block is then combined with results from previous iterations using overlap-add, and the portion that no longer overlaps is output as the final whitened data.

For data whitening, each length- N FFT block is divided by the square root of the PSD to generate the whitened data. An inverse FFT is then applied to obtain the whitened data in the time domain, which is tapered with a Tukey window. Successive blocks are combined using the overlap-add method, with each new Tukey-windowed block added to the previous one with an overlap of $N/2 + Z$. The first $N/2 - Z$ samples of each processed block constitute the final output whitened data, which therefore lag the input by $N/2$ points. The input segment is then advanced by $N/2 - Z$ points for the next iteration.

The FFT block and overlap scheme introduce a latency of $N/2$ sample points in PSD estimation and whitening. For low-latency operation, N is usually four seconds of samples, Z is one second, and $n_{med} = 7$. This gives an effective latency of two seconds. Fig. 2 illustrates how the input, Hann window, Tukey window, and output

blocks overlap and advance.

Strain data, typically sampled at 16,384 Hz, must be downsampled to match the maximum sample rate of the template bank, usually 2,048 Hz. In SGNL, downsampling is performed in the frequency domain after the FFT by removing high-frequency components. In contrast, GstLAL performs downsampling before the strain data enter the PSD estimation and whitening stage, using a finite impulse response (FIR) filter (see Sec. IIC 2a) in the time domain. This FIR-based approach adds extra latency, whereas frequency-domain downsampling during the FFT avoids it. Fig. 3 illustrates the full PSD estimation and whitening workflow.

Fig. 2 also shows the effect of data gaps. When gaps occur, any FFT block with input data containing gaps is skipped, and no PSD is estimated for that segment (orange and green iterations in Fig. 2). However, whitened data continue to be produced until the overlap from previous iterations is fully drained.

3. Threshold gating

LVK detector strain data contain non-Gaussian, short-duration noise transients, or “glitches,” which can mimic gravitational-wave signals from high-mass compact binaries. The SGNL pipeline employs the same gating method as GstLAL to address these artifacts [11, 12]. Once the data are whitened, giving them unit variance, any brief excursion exceeding a threshold defined in standard deviations (σ) is set to a gap buffer, with a 0.5s padding applied on either side.

4. Transferring data to GPU

In SGNL’s GPU mode, whitened and gated data from each detector are synchronized and transferred to the GPU. If half-precision computation is enabled, the data are converted accordingly at this stage. When running in CPU mode, this step is a no-op.

C. The LLOID filtering algorithm

The SGNL analysis builds upon the time-domain matched filtering method used in GstLAL, in which compact binary coalescence signals are identified by correlating the detector strain time series with a large bank of template waveforms [11, 18]. In the time domain, the matched filter output for a template $h_i(t)$ and detector data $d(t)$ is given by

$$x_i(t) = 2 \int_{-\infty}^{\infty} \hat{h}_i(\tau) \hat{d}(t + \tau) d\tau, i \in [0, 2M - 1], \quad (1)$$

where M is the number of templates, the whitened data

$$\hat{d}(\tau) = \int_{-\infty}^{\infty} df \frac{\tilde{d}(f)}{\sqrt{S_n(|f|)}} e^{2\pi i f \tau}, \quad (2)$$

and the whitened templates $\hat{h}_i(\tau)$ are defined similarly. The LLOID algorithm uses real templates for filtering, and to account for both phases of the matched filter response, both the real and imaginary parts of each template are filtered using Eq. 1. For a template bank containing M template waveforms, this results in $2M$ real template filters.

Modern template banks can contain millions of waveforms, with low-mass binary templates spanning hundreds of seconds. Direct computation of these cross-correlations for all templates is therefore computationally prohibitive, particularly in low-latency applications where results must be produced in near real time.

The LLOID algorithm addresses this challenge by compressing the template bank using SVD and processing each time slice at the minimum sampling rate required to capture its frequency content according to the Nyquist theorem [18]. By representing templates with a reduced set of orthogonal basis waveforms and employing a multirate filtering strategy, LLOID significantly reduces the computational cost while preserving sensitivity to gravitational-wave signals.

1. SVD bank construction

The SVD bank construction follows the LLOID algorithm, combining time slicing and SVD to compress the template bank. Templates are first divided into groups [11, 14]. Within each group, templates are whitened and then divided into time slices. The time slices within a group share the same time boundaries across templates. Each slice is downsampled to the Nyquist rate corresponding to its highest frequency content, ensuring it is sampled at the minimum rate needed for that slice.

For each time slice, the set of templates is arranged into a matrix, with real and imaginary components stacked, and then decomposed using SVD. This produces a set of orthonormal basis waveforms ranked by their significance. Only the leading bases are retained, as higher-order components contribute negligibly to the accuracy required for matched filtering:

$$\hat{h}_i^s[k] \approx \sum_{l=0}^{L^s-1} v_{il}^s \sigma_l^s u_l^s[k], s \in [0, S-1], \quad (3)$$

where S is the number of time slices, with 0 representing the latest time slice. $u_l^s[k]$ are the orthonormal basis waveforms, v_{il}^s are the reconstruction coefficients for template i , σ_l^s are the singular values, ordered by their contribution to reconstructing $\hat{h}_i^s[t]$, and L^s is the number of basis components in the time slice. The SVD procedure reduces the number of waveform filters by ap-

proximately two orders of magnitude in the GstLAL O4 template bank [14].

2. Filtering workflow

a. Downsampling of whitened data Each template time slice requires a specific sample rate, so the input detector data is downsampled to match the rate of each time slice. This produces multiple downsampled streams that correspond to the different time slice rates.

Downsampling is implemented using a sinc-windowed sinc kernel:

$$g[k] = \begin{cases} \frac{\sin(\pi(k-c)/f)}{\pi(k-c)/f} \cdot \frac{\sin(\pi(k-c)/c)}{\pi(k-c)/c}, & k \neq c, \\ 1, & k = c, \end{cases} \quad (4)$$

where f is the downsampling factor (restricted to powers of two) and c is the half length of the filter. The total kernel length is $2c + 1$, and for downsampling $c = fN_{\text{down}}$, where N_{down} is the half filter length at the target rate, which is typically set to 32 samples in GstLAL, and determines the latency introduced by the downsampling filter.

b. Filtering and reconstruction After the input data for a given time slice s is downsampled to match its sample rate, it is filtered by cross-correlating with the basis templates $u_l^s[n]$:

$$y_l^s[k] = \sum_{n=0}^{N^s-1} u_l^s[n] d^s[k+n], \quad (5)$$

where $y_l^s[k]$ is the filtered output for the basis component l in time slice s , and N^s is the length of the basis template for that time slice.

Once all basis components are filtered, the original templates are reconstructed by combining these components using their singular values σ_l^s and reconstruction coefficients ν_{il}^s :

$$y_i^s[k] = \sum_{l=0}^{L^s-1} \sigma_l^s \nu_{il}^s y_l^s[k]. \quad (6)$$

This reconstruction produces the SNR time series for all templates at the corresponding sample rate of the time slice.

c. Upsampling SNR segments After filtering and reconstructing each downsampled time slice, the outputs must be upsampled to the original sample rate. Standard upsampling typically inserts zeros between samples and convolves with a kernel, but SGNL (following GstLAL) uses a polyphase method to avoid multiplying by zeros.

The upsampling kernel is also a sinc-windowed sinc kernel, as defined in Eq. 4, with the total kernel length defined as $2fN_{\text{up}} + 1$, where N_{up} is the half-length at the original sample rate for the upsampling kernel, and is set

to 8 samples. To perform polyphase interpolation, the kernel is divided into f sub-kernels:

$$z_j[k] = g[kf + j], \quad j = 0, \dots, f-1, \quad k = 0, \dots, 2c/f. \quad (7)$$

Each sub-kernel is convolved directly with the input, and the outputs are interleaved to produce the final upsampled SNR time series:

$$y[i] = \begin{cases} x[i/f], & i \bmod f = 0, \\ \sum_{k=0}^{2c/f} x[\lfloor i/f \rfloor - k] z_{i \bmod f}[k], & \text{otherwise.} \end{cases} \quad (8)$$

where $\lfloor \cdot \rfloor$ denotes floor division.

d. Combining SNR segments The SNR segments are combined recursively, starting from the lowest-rate time slice. Each segment is upsampled to the next higher sample rate and then added to the SNR contribution from the higher-rate slice. This process is repeated through all slices to construct the full SNR time series. Each time slice has a known time delay from the LLOID decomposition, and as long as these delays exceed the combined lengths of the downsampling and upsampling filters, no additional latency is introduced.

3. Extensions in SGNL

a. Pre-synchronization of time slices In SGNL, time slices are pre-synchronized before filtering rather than only aligned after the LLOID pipeline output. For each time slice s , SGNL identifies exactly which segment of the downsampled input $x^s[k]$ is needed so that, after filtering, reconstruction, and upsampling, the slice's contribution aligns precisely with the high-rate SNR output. This avoids trimming outputs or waiting for extra buffers.

Two main factors determine the segment:

1. The time delay z^s of the time slice s .
2. The cumulative half-lengths of all upsampling kernels between the slice's rate f^s and the maximum rate f^0 .

Using these offsets, SGNL identifies the exact portion of each downsampled stream to process, avoiding unnecessary reads and eliminating added latency.

The recursive reconstruction of the SNR contribution from slice s can be expressed as:

$$\begin{aligned} \rho^s[k] = & (H^\uparrow \rho^{s+1})[k] + \sum_{l=0}^{L^s-1} \sigma_l^s \nu_{il}^s \sum_{n=0}^{N^s-1} u_l^s[n] \\ & \times d^s \left[k + n - z^s + \sum_{f^s \leq f < f^0} \frac{N_{\text{up}}}{f} \right], \end{aligned} \quad (9)$$

where H^\uparrow is the upsampling operator that maps a slice to the next-higher sample rate with Eq. 8, and all other symbols are as defined in previous sections.

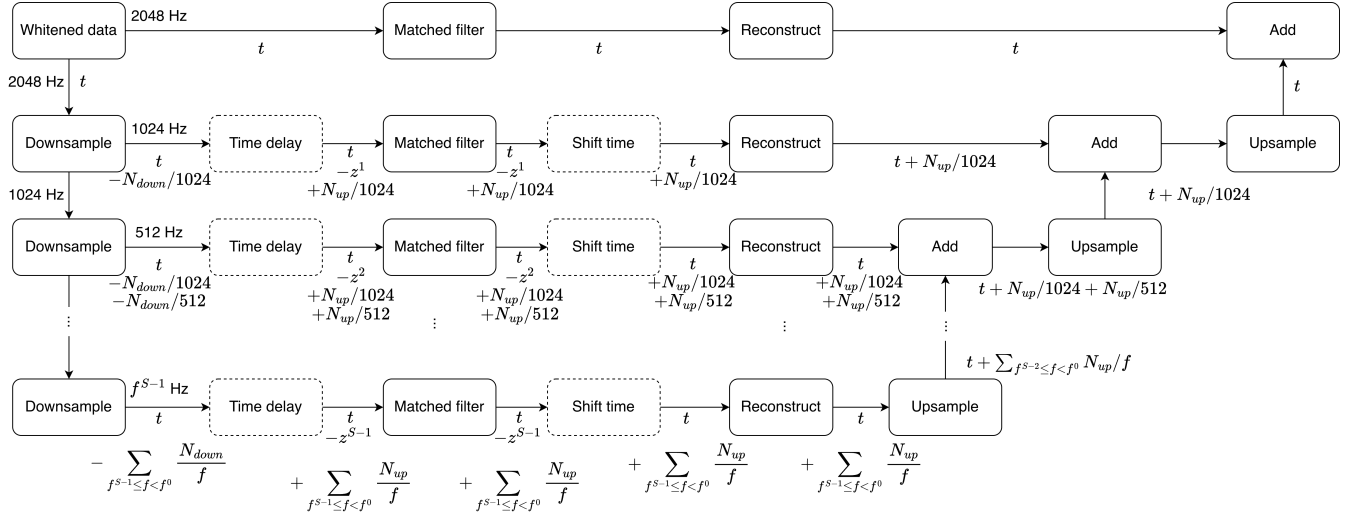


FIG. 4: Diagram of the LLOID algorithm pipeline. The time evolution of the latest data sample at each processing stage is indicated alongside the transition arrows. Whitened data are downsampled to match the sample rates of each SVD bank. After downsampling, the output lags the input by N_{down}/f seconds. Each time slice is then filtered using the appropriately delayed, downsampled data, with an additional shift equal to the upsampling kernel length. Following filtering, the output timestamp is advanced by the time delay associated with that slice. The filtered outputs are multiplied by the SVD reconstruction matrices to produce the physical SNR streams at each sample rate. The lowest-rate SNR stream is recursively upsampled and added to the higher-rate streams, where each upsampling stage introduces a latency of N_{up}/f seconds. This pre-synchronization of delays and time shifts ensures that the upsample-and-add process remains fully aligned, eliminating additional buffering and enabling the LLOID algorithm to operate with zero latency.

This backward bookkeeping ensures that each slice contributes to the output fully synchronized, improving temporal alignment across the pipeline. Fig. 4 shows the LLOID algorithm as implemented in the SGNL pipeline.

b. Multidimensional filtering In SGNL, filtering can be performed on multiple SVD banks in parallel by stacking the bases and reconstruction matrices into tensor objects, which are multidimensional arrays. The retained bases and their reconstruction coefficients after SVD are grouped by sample rate. During initialization, bases from slices with the same rate are stacked into a tensor, as are their reconstruction coefficients. Each unique sample rate has a corresponding downsampled data stream, and the relevant time segments for each slice are aligned and stacked accordingly, as detailed in Sec. II C 3 a. This tensorized layout enables SGNL to perform filtering and matrix multiplication across multiple slices and SVD banks in parallel with PyTorch’s multibatch and multichannel operations, thereby optimizing GPU performance.

D. Event identification

1. Trigger identification

After the LLOID filtering algorithm, the $2M$ real template filters produce $2M$ SNR time series, with the real and imaginary components corresponding to the SNR outputs of each of the M template waveforms. For each

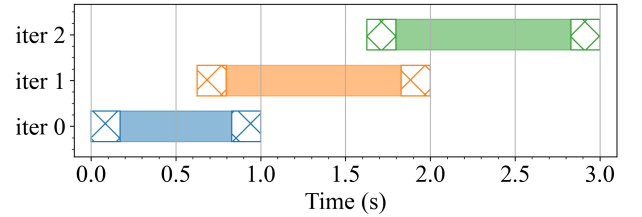


FIG. 5: Overlapping and streaming behavior during the trigger and coincidence identification stage. Blue, orange, and green rectangles indicate iterations 0, 1, and 2 of the pipeline, respectively. Shaded regions mark the trigger-finding windows, and cross-hatched regions show the padding regions used for the χ^2 calculation (350 samples in this example, corresponding to an autocorrelation length of 701 samples). The trigger-finding windows overlap by the sum of the maximum light-travel time between detectors (0.0273 s for an LIGO Hanford (H), LIGO Livingston (L), and Virgo (V) search) and a 0.005 s coincidence buffer. Each trigger finding window therefore spans one second plus the overlap interval. Input data are in one-second strides, thus the first window is shortened to produce output as soon as data become available, minimizing additional latency.

detector and template, the trigger is defined as the maximum modulus of the complex SNR that exceeds a threshold of $\rho = 4$. The corresponding time, SNR, and phase are recorded for each trigger. This trigger identification

process is performed within each one-second segment, including a small buffer to account for edge effects during coincidence formation (see Sec. IID 3 and Fig. 5). This maximization is implemented using PyTorch operations and executed on the GPU when GPU mode is enabled.

2. Signal-Based Consistency Test

For each SNR peak identified during the trigger identification stage, a signal-consistency statistic is computed as

$$\xi_j^2 = \frac{\int_{-\delta t}^{\delta t} dt |z_j(t) - z_j(0)R_j(t)|^2}{\int_{-\delta t}^{\delta t} dt (2 - 2|R_j(t)|^2)}, j \in [0, M-1], \quad (10)$$

where ξ_j^2 quantifies the consistency of the SNR peak with the expected complex template autocorrelation. Here, $z_j(t)$ denotes the complex matched-filter SNR time series, $z_j(0)$ its peak value at the trigger time, and $R_j(t)$ the complex template autocorrelation function, normalized such that $R_j(0) = 1$ [11]. The integration half-width δt corresponds to half the full autocorrelation length. The full autocorrelation length is 351 samples for high-mass templates and 701 samples for low-mass templates. This defines the segment of the SNR time series used in the ξ_j^2 computation and contributes to the overall pipeline latency. The streaming and overlap behavior are illustrated in Fig. 5, where the cross-hatched region represents the padding interval for the autocorrelation length calculation and indicates the associated processing latency.

To compute this statistic, the autocorrelation functions are stacked across SVD banks, and the corresponding segment of the SNR time series, centered on the trigger, is extracted with the same length as the autocorrelation across templates. The calculation is performed using tensor operations and executed on the GPU when GPU mode is enabled. Evaluated for each trigger, ξ_j^2 provides a waveform-consistency measure that complements SNR and detector coincidence in event ranking.

3. Coincidence Formation

To suppress false alarms, candidate events must occur in temporal coincidence across multiple detectors. The event time for each single-detector trigger is defined as the time of the SNR peak. For every trigger in one detector, the pipeline checks if triggers in the other detectors are within a coincidence window that accounts for the gravitational-wave travel time between sites. An additional buffer of 0.005 s is included to accommodate timing variations. To minimize edge effects, the trigger-finding windows overlap by the sum of the maximum light-travel time and the coincidence buffer. Fig. 5 illustrates the small overlap between adjacent trigger-finding windows (shaded regions). The coincidence check is implemented

using tensor operations and executes efficiently on the GPU when GPU mode is enabled.

4. Clustering

After coincidence formation, each event may consist of a variable number of coincident triggers across detectors. Within each subbank, events are then clustered, and only the event with the highest network SNR is retained, ensuring that at most one event is kept per subbank. In GPU mode, the selected events are transferred to the CPU for subsequent processing. The clustering uses the same time window as the trigger-finding stage, removing closely spaced redundant events.

5. Significance estimation

SGNL refactored GstLAL's likelihood ratio calculations for ranking statistics into a modular library, STRIKE [27]. This calculation requires collecting background noise distributions for each detector and for each SVD bin. Background collection uses the SNR and ξ^2 values of triggers that exceed the SNR threshold during the trigger-identification stage. We only include single-detector triggers that occurred while multiple detectors were operating.

In multi-SVD bank mode, the background is stored in a multi-dimensional tensor initialized at startup. Each SVD bank accumulates its background triggers independently. For each trigger-finding window, binning is performed in parallel across SVD banks. The resulting counts are continuously added to the persistent background tensor. These operations run on the GPU when GPU mode is enabled. The background is snapshotted every four hours, and is smoothed by the Gaussian kernel as kernel density estimation. At this point, the background tensor is transferred to the CPU.

For each event that survives the clustering stage (Sec. IID 4), the likelihood ratio is calculated using the snapshotted background tensor of the specific bin that the event belongs to. The persistent background tensor on the GPU (in GPU mode) continues collecting new background data.

Each snapshot of the background is saved to disk and also posted via a Bottle [28] route. A companion program queries this background data through Bottle services, draws samples from the SNR- ξ^2 distributions from each SVD bin across all inspiral programs, and constructs a marginalized background model across SVD bins. The resulting marginalized background is saved to disk. Inspirational jobs check for updated files and reload them in memory at every snapshot interval. FAR assignments for events with likelihood ratio evaluations are then calculated using the marginalized background.

Events and their associated triggers are saved to an in-memory SQLite database. At each snapshot interval,

the triggers are also saved to disk.

6. Event candidate alerting

After the likelihood ratio is calculated and FAR is assigned, events that pass the upload threshold are internally aggregated across subbanks to select a local candidate within a program. If the FAR is below the public alert threshold, the event with the maximum SNR is selected. If the FAR is above the threshold, the event with the minimum FAR is chosen. The local candidate is sent through Kafka. An auxiliary program further aggregates local candidates across all inspiral programs and sends the global candidate to GraceDB [12].

During internal aggregation, the local candidate may include triggers from only a subset of the observing detectors. For example, if H, L, and V are observing, the local candidate might initially contain triggers only from H and L. To provide complete SNR time-series information for downstream processing, we search for a corresponding trigger in the missing detector that lies within the light-travel-time window, and add it to the local candidate. The SNR time series for each trigger forming the local candidate are then extracted and stored in the local-candidate metadata before it is sent out. This ensures full SNR time series information is available for downstream skymap generation.

7. Monitoring metrics

Metrics from each event, including SNR, likelihood ratio, and FAR, and latency at various pipeline stages, are sent to Kafka topics. Metrics from multiple inspiral programs are aggregated and stored in an InfluxDB [29]. Grafana [30] panels are then used to visualize these metrics over time and monitor the pipeline’s performance and stability.

III. MOCK DATA CHALLENGE RESULTS

To assess the performance of the SGNL online analysis and compare it with GstLAL, we ran an analysis on an MDC. The MDC consists of 40 days of O3 H, L, and V data, from Jan 05 23:59:42 GMT 2020 to Feb 14 23:59:42 GMT 2020. The data are replayed with timestamps shifted to the present to simulate live streams. Alongside the strain data, additional channels containing simulated CBC waveforms for binary neutron stars (BNS), neutron star black hole binaries (NSBH), and binary black holes (BBH) were streamed in parallel. Injections were added roughly every minute, for a total of about 50,000 injections. A detailed description of the MDC and the injection set can be found in [31].

The GstLAL MDC results used for comparison in this paper were taken from the MDC iteration spanning Jan

02 18:39:42 UTC 2024 to Feb 11 18:39:42 UTC 2024, which is produced with GstLAL in its configuration deployed in the second part of O4. The O4 template bank is split into two “checkerboarded” halves [14]. This checkerboarding process takes every other adjacent template in the full bank, forming two banks that cover the same parameter space. The GstLAL MDC follows the O4 configuration and processes the two checkerboarded banks at different computing sites to provide redundancy against computing cluster failures.

For the SGNL MDC in this work, we used one checkerboarded half of the full GstLAL template bank. We ran on 38 NVIDIA A2 16 GB GPUs, with half of the GPUs processing strain data and the other half processing strain data with injections added. The filtering algorithm is conducted with float16 precision. One inspiral program processes ~ 35 -60 SVD banks. The filtering, trigger generation, and coincidence formation are done on the GPU.

In this section we aim to compare the performance between SGNL and GstLAL on the MDC. We will first analyze the event recovery of known gravitational-wave events identified in O3 within the duration of the MDC data. Next, we will analyze the injection recovery comparison in terms of the sensitive spacetime volume and injection parameters. Finally, we will highlight the difference in latency performance between the pipelines.

A. Gravitational wave events

There are nine gravitational-wave events previously published in GWTC-3 [7] that are in the MDC data. Table I summarizes the SNR and FAR results for all nine gravitational-wave events in the MDC for both GstLAL and SGNL. The preferred event was selected with the criteria of the maximum SNR event among the events with FAR below the public alert threshold, otherwise the minimum FAR event was selected. During the time of the GstLAL MDC, GstLAL’s SNR optimizer was also uploading events to GraceDB, for the purpose of this study we exclude the SNR optimizer events when selecting the preferred event. The number of events versus inverse false-alarm rate (IFAR) plots for the SGNL MDC are shown in Fig. 6.

For eight of the nine gravitational-wave events, SGNL recovered network SNRs within $< 1\%$ of GstLAL’s values and obtained consistent FARs. The remaining event, GW200202_154313, showed a lower SNR and a FAR that was orders of magnitude higher in the SGNL MDC compared to GstLAL. Investigation revealed that the GstLAL-preferred event (hereafter E1) came from the checkerboard half of the template bank that SGNL did not use. GstLAL also uploaded a second candidate (E2) from the checkerboard used by SGNL, with an SNR of 10.57 and a FAR of $4.87 \times 10^{-2} \text{ yrs}^{-1}$. E2 has a lower SNR and higher FAR than E1.

We therefore compared the SGNL event with E2. They

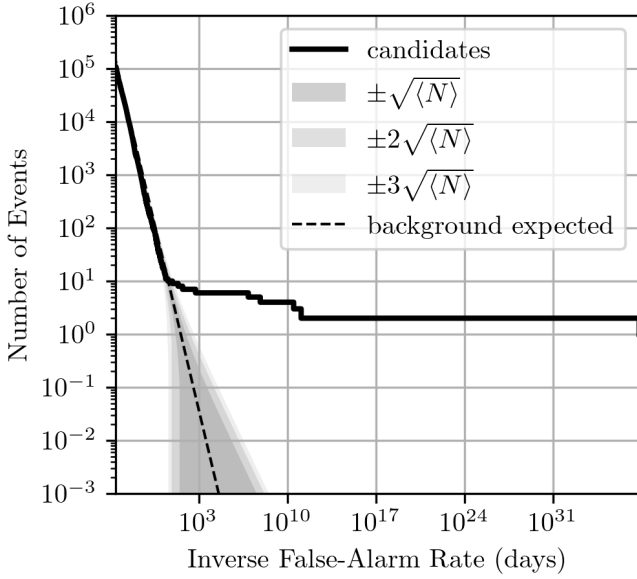


FIG. 6: Number of events versus IFAR in days. Dashed line: expected background distribution. Shaded areas: 1, 2, and 3 σ uncertainties. Black line: number of candidates observed.

were recovered with different templates, although both templates belong to the same SVD subbank. Offline re-filtering showed that this difference comes from the trigger-finding window boundaries: the SNR maximization step selects different triggers when the boundaries shift, especially for triggers close to threshold. This accounts for the SNR difference between the SGNL event and E2, and contributes to the FAR discrepancy as well.

A further contribution to the FAR difference comes from SGNL adopting recent GstLAL updates, including a different KDE smoothing kernel for the background distribution. For this event, the L1 trigger lay near the edge of the background, where the choice of kernel has a larger impact. In addition, SGNL experienced intermittent data-delivery issues in the hours preceding the event, resulting in frequent dropped segments of L1 data. These gaps reduced the cadence of PSD updates, which affected the background accumulation leading up to the event. Although this event is not recovered as significantly in SGNL as it is in GstLAL, injection studies demonstrate that the search sensitivity between SGNL and GstLAL is consistent within statistical and systematic uncertainties (see Sec. III C). This indicates that the behavior of this particular event does not reflect a systematic issue.

B. Retraction

In addition to the known gravitational-wave events, we identify “retraction-level candidates,” defined as events with a FAR below one per year for the MDC [12]. Both GstLAL and SGNL found one such event, corresponding to the same GPS time in the original O3 data. Gst-

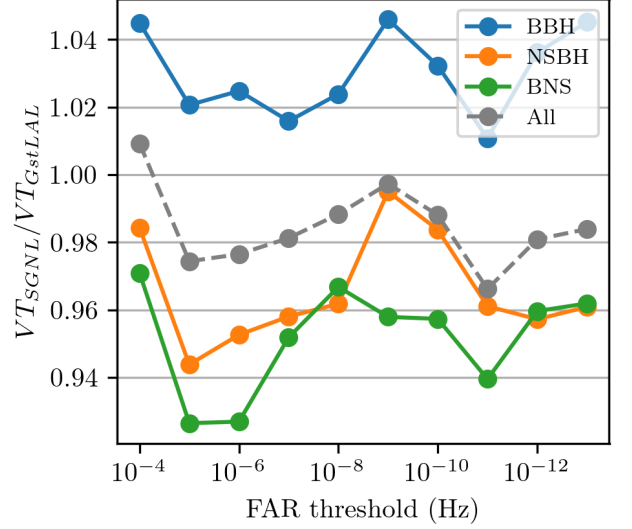


FIG. 7: $\langle VT \rangle$ ratio between SGNL and GstLAL at the end of the MDC for each source class. VT_{SGNL} is the $\langle VT \rangle$ for the SGNL analysis, while VT_{GstLAL} is the $\langle VT \rangle$ for the GstLAL analysis. BBH $\langle VT \rangle$ is shown in blue, NSBH $\langle VT \rangle$ in orange, BNS $\langle VT \rangle$ in green, and the combined $\langle VT \rangle$ across all sources is shown in the grey dashed line. SGNL analyzes a single checkerboard for this study, whereas GstLAL analyzes the full template bank (two checkerboards). A single checkerboard has a sensitivity of 97% w.r.t. the full template bank [14].

LAL recovered this event in the L1 detector with an SNR of 15.07, and SGNL found it in the same detector with an SNR of 15.06. The FAR values are 2.56×10^{-6} and 1.24×10^{-8} per year for GstLAL and SGNL, respectively, both below the one-per-year threshold. This event also appeared in previous GstLAL MDC studies [12]. As shown in Fig. 3 of [12], the corresponding spectrogram exhibits scattering glitches, and because the event was recovered in only one detector while all three were operational, it is likely of terrestrial origin. Since the purpose of this MDC comparison is to assess consistency between GstLAL and SGNL, the recovery of this retraction event by SGNL is expected.

C. Recovered injections

1. Search Sensitivity

We use the spacetime volume, or sensitivity volume-time (VT), to quantify the analysis sensitivity. The comoving volume is defined as [32, 33]

$$V_c = 4\pi D_H \int_0^z dz \frac{(1+z)D_A^2}{E(z)}, \quad (11)$$

where z is the redshift, D_H is the Hubble distance, D_A is the angular distance, and $E(z)$ is the Hubble parameter.

TABLE I: Comparison of gravitational-wave event candidates previously reported in GWTC-3 within the MDC time span, between GstLAL and SGNL. The “found inst.” column lists the instruments that identified each event with a trigger SNR ≥ 4.0 . All reported SNRs are network SNRs.

Name	Found Inst.	GstLAL MDC		SGNL MDC	
		SNR	FAR (yrs ⁻¹)	SNR	FAR (yrs ⁻¹)
GW200112.155838	L1	18.46	2.05×10^{-8}	18.47	2.11×10^{-8}
GW200115.042309	H1L1	11.48	3.23×10^{-5}	11.50	1.29×10^{-5}
GW200128.022011	H1L1	9.97	8.72×10^{-5}	9.98	2.26×10^{-3}
GW200129.065458	H1L1V1	26.30	5.51×10^{-36}	26.26	5.71×10^{-36}
GW200202.154313	H1L1	11.09	6.18×10^{-2}	10.14	3.32×10^2
GW200208.130117	H1L1	10.56	1.47×10^{-5}	10.54	1.02×10^{-4}
GW200208.222617	H1L1	8.03	1.28×10^3	8.03	8.12×10^2
GW200209.085452	H1L1	9.97	9.62×10^{-1}	9.96	9.21×10^{-1}
GW200210.092254	H1L1	9.27	7.57×10^2	9.21	6.71×10^2

Multiplying the comoving volume by the total observing time gives the searched spacetime volume. The injection VT, $\langle VT \rangle_{\text{inj}}$, is calculated using the maximum redshift of the injections and the time span they cover. The analysis VT, $\langle VT \rangle$, is defined as the fraction of recovered injections multiplied by the injection VT [12]:

$$\langle VT \rangle = \frac{\text{found}}{\text{total}} \langle VT \rangle_{\text{inj}}. \quad (12)$$

The VT ratio between analyses is defined as the ratio of the number of recovered injections.

Fig. 7 shows the VT ratio between the SGNL analysis and the GstLAL analysis at different FAR thresholds, which are calculated as the ratio of number of found injections, with the “found” criteria being the injections that are below the given FAR threshold. The VT ratios are calculated for difference source classes, BNS, NSBH, and BBH. The single checkerboard has a sensitivity of 97% w.r.t. the full template bank [14]. Other contributions to the discrepancies are the lack of the sub-sample interpolation in SNR, the difference in analysis operating time, and the occurrence of dropped data. We conclude that the $\langle VT \rangle$ of SGNL is consistent with GstLAL within statistical and systematic uncertainties.

D. Upload latency

Fig. 8 shows the histogram of GraceDB reporting latencies for GstLAL and SGNL. The GraceDB reporting latency is defined as the time when GraceDB receives an event minus the event coalescence time. For GstLAL, the median latency is 9.3 seconds, while SGNL has a median of 5.4 seconds, representing a 42% reduction. GstLAL’s latency histogram shows two modes: one around 8–9 seconds and another around 11–12 seconds. This is caused by the downstream event aggregation process, where the aggregator waits two seconds to combine events [12].

SGNL’s latencies peak around 5–6 seconds, with the shortest latency at 3.77 seconds. SGNL lacks a second

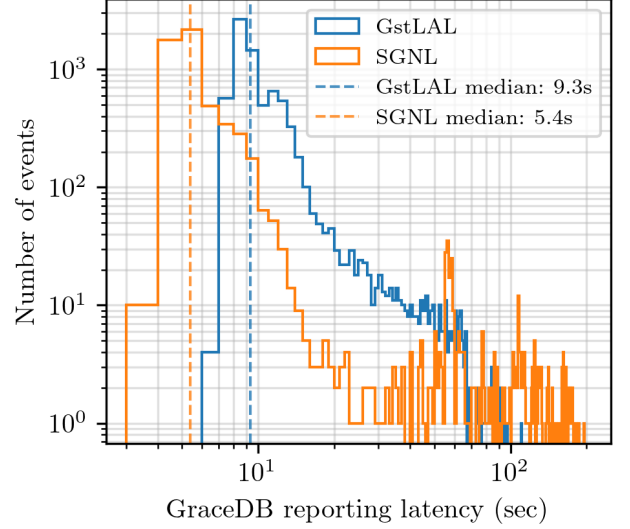


FIG. 8: Histograms of GraceDB reporting latencies. The blue histogram shows latencies for the GstLAL analysis, while the orange histogram shows latencies for the SGNL analysis. Dashed lines indicate the median latency for each pipeline: blue for GstLAL and orange for SGNL.

mode because each inspiral job processes about 50 SVD banks and performs internal aggregation to select the best event (see Sec. IID 6). Fewer events therefore reach the downstream aggregator.

A small bump around 60 seconds is due to a ~ 40 -hour period of unstable $h(t)$ data streaming and a 60-second timeout in the data consumer (see Sec. IIB 1). A few SGNL events have latencies up to 180 seconds. This is caused by occasional GraceDB instability during the study, which delayed uploads until several retries later.

IV. CONCLUSION

We have presented SGNL, a scalable, low-latency gravitational-wave detection pipeline for compact binary mergers. SGNL modernizes the GstLAL framework by leveraging a Python-based streaming architecture, SGN, enabling modular, flexible, and GPU-accelerated processing while preserving the physics-proven methods of matched filtering and likelihood-ratio ranking. Key innovations include pre-synchronization of LLOID time slices, multidimensional tensorized filtering, and optimized data streaming, all of which contribute to low-latency operation without sacrificing accuracy.

Performance evaluation using a MDC demonstrates that SGNL reliably recovers known gravitational-wave events and simulated injections, with sensitivities comparable to GstLAL, within expected uncertainties using a single checkerboard. Notably, SGNL reduced GstLAL's median latency from 9.3 seconds to 5.4 seconds, which is a 42% reduction.

Looking ahead, SGNL offers a flexible platform for future enhancements, including expanded parameter-space coverage and integration with machine learning techniques. Its design ensures that low-latency detection and multimessenger alerts can be performed efficiently, supporting rapid follow-up and maximizing the scientific output of the global gravitational-wave network.

ACKNOWLEDGMENTS

This material is based upon work supported by NSF's LIGO Laboratory which is a major facility fully funded by the National Science Foundation. This research

has made use of data, software and/or web tools obtained from the Gravitational Wave Open Science Center (<https://www.gw-openscience.org/>), a service of LIGO Laboratory, the LIGO Scientific Collaboration and the Virgo Collaboration. We especially made heavy use of the LVK Algorithm Library [25]. LIGO Laboratory and Advanced LIGO are funded by the United States National Science Foundation (NSF) as well as the Science and Technology Facilities Council (STFC) of the United Kingdom, the Max-Planck-Society (MPS), and the State of Niedersachsen/Germany for support of the construction of Advanced LIGO and construction and operation of the GEO600 detector. Additional support for Advanced LIGO was provided by the Australian Research Council. Virgo is funded, through the European Gravitational Observatory (EGO), by the French Centre National de Recherche Scientifique (CNRS), the Italian Istituto Nazionale di Fisica Nucleare (INFN) and the Dutch Nikhef, with contributions by institutions from Belgium, Germany, Greece, Hungary, Ireland, Japan, Monaco, Poland, Portugal, Spain.

The authors are grateful for computational resources provided by the the Pennsylvania State University's Institute for Computational and Data Sciences gravitational-wave cluster, and the LIGO Lab cluster at the LIGO Laboratory, and supported by the National Science Foundation awards OAC-2103662, PHY-2308881, PHY-2011865, OAC-2201445, OAC-2018299, PHY-0757058, PHY-0823459, PHY-2207728, PHY-2513124, PHY-2110594, and PHY-2513358. CH Acknowledges generous support from the Eberly College of Science, the Department of Physics, the Institute for Gravitation and the Cosmos, and the Institute for Computational and Data Sciences.

[1] The LIGO Scientific Collaboration, the Virgo Collaboration, the KAGRA Collaboration, A. G. Abac, I. Abouelfettouh, F. Acernese, K. Ackley, C. Adamcewicz, S. Adhicary, D. Adhikari, N. Adhikari, R. X. Adhikari, V. K. Adkins, S. Afroz, A. Agapito, D. Agarwal, M. Agathos, N. Aggarwal, S. Aggarwal, O. D. Aguiar, I. L. Ahrend, L. Aiello, A. Ain, P. Ajith, T. Akutsu, S. Albanesi, W. Ali, S. Al-Kersh, C. Alléné, A. Allocca, S. Al-Shammari, P. A. Altin, S. Alvarez-Lopez, W. Amar, O. Amarasingham, A. Amato, F. Amicucci, C. Amra, A. Ananyeva, S. B. Anderson, W. G. Anderson, M. Andia, M. Ando, M. Andrés-Carcasona, T. Andrić, J. Anglin, S. Ansoldi, J. M. Antelis, S. Antier, M. Aoumi, E. Z. Appavuravther, S. Appert, S. K. Apple, K. Arai, A. Araya, M. C. Araya, M. Arca Sedda, J. S. Areeda, N. Aritomi, F. Armato, S. Armstrong, N. Arnaud, M. Arogeti, S. M. Aronson, K. G. Arun, G. Ashton, Y. Aso, L. Asprea, M. Assiduo, S. Assis de Souza Melo, S. M. Aston, P. Astone, F. Attadio, F. Aubin, K. AultO'Neal, G. Avallone, E. A. Avila, S. Babak, C. Badger, S. Bae, S. Bagnasco, L. Baiotti, R. Bajpai, T. Baka, A. M. Baker, K. A. Baker, T. Baker,

G. Baldi, N. Baldicchi, M. Ball, G. Ballardín, S. W. Ballmer, S. Banagiri, B. Banerjee, D. Bankar, T. M. Baptiste, P. Baral, M. Baratti, J. C. Barayoga, B. C. Barish, D. Barker, N. Barman, P. Barneo, F. Barone, B. Barr, L. Barsotti, M. Barsuglia, D. Barta, A. M. Bartoletti, M. A. Barton, I. Bartos, A. Basalae, R. Bassiri, A. Basti, M. Bawaj, P. Baxi, J. C. Bayley, A. C. Baylor, P. A. Baynard, II, M. Bazzan, V. M. Bedakihale, F. Beirnaert, M. Bejger, D. Belardinelli, A. S. Bell, D. S. Bellie, L. Bellizzi, W. Benoit, I. Bentara, J. D. Bentley, M. Ben Yaala, S. Bera, F. Bergamin, B. K. Berger, S. Bernuzzi, M. Beroiz, C. P. L. Berry, D. Bersanetti, T. Bertheas, A. Bertolini, J. Betzwieser, D. Beveridge, G. Bevilacqua, N. Bevins, R. Bhandare, R. Bhatt, D. Bhattacharjee, S. Bhattacharyya, S. Bhaumik, V. Biancalana, A. Bianchi, I. A. Bilenko, G. Billingsley, A. Binetti, S. Bini, C. Binu, S. Biot, O. Birnholtz, S. Biscoveanu, A. Bisht, M. Bitossi, M. A. Bizouard, S. Blaber, J. K. Blackburn, L. A. Blagg, C. D. Blair, D. G. Blair, N. Bode, N. Boettner, G. Boileau, M. Boldrini, G. N. Bolingbroke, A. Bolliand, L. D. Bonavena, R. Bondarescu, F. Bondu, E. Bonilla, M. S. Bonilla, A. Bonino,

- R. Bonnand, A. Borchers, S. Borhanian, V. Boschi, S. Bose, V. Bossilkov, Y. Bothra, A. Boudon, L. Bourg, M. Boyle, A. Bozzi, C. Bradaschia, P. R. Brady, A. Branch, M. Branchesi, I. Braun, T. Briant, A. Brillet, M. Brinkmann, P. Brockill, and E. Brockmueller, GWTC-4.0: Updating the Gravitational-Wave Transient Catalog with Observations from the First Part of the Fourth LIGO-Virgo-KAGRA Observing Run, arXiv e-prints , arXiv:2508.18082 (2025), arXiv:2508.18082 [gr-qc].
- [2] J. Aasi *et al.* (LIGO Scientific Collaboration), Advanced LIGO, Classical and Quantum Gravity **32**, 074001 (2015).
- [3] F. Acernese *et al.*, Advanced Virgo: a second-generation interferometric gravitational wave detector, Classical and Quantum Gravity **32**, 024001 (2014).
- [4] T. Akutsu *et al.*, Overview of KAGRA: Detector design and construction history, Progress of Theoretical and Experimental Physics **2021**, 05A101 (2020).
- [5] B. P. Abbott *et al.* (LIGO Scientific Collaboration and Virgo Collaboration), GWTC-1: A Gravitational-Wave Transient Catalog of Compact Binary Mergers Observed by LIGO and Virgo during the First and Second Observing Runs, Phys. Rev. X **9**, 031040 (2019).
- [6] R. Abbott *et al.* (LIGO Scientific Collaboration and Virgo Collaboration), GWTC-2: Compact Binary Coalescences Observed by LIGO and Virgo during the First Half of the Third Observing Run, Phys. Rev. X **11**, 021053 (2021).
- [7] R. Abbott *et al.* (LIGO Scientific Collaboration, Virgo Collaboration, and KAGRA Collaboration), GWTC-3: Compact Binary Coalescences Observed by LIGO and Virgo during the Second Part of the Third Observing Run, Phys. Rev. X **13**, 041039 (2023).
- [8] L.-V.-K. Collaboration, Gravitational-wave candidate event database, <https://gracedb.ligo.org/superevents/public/>.
- [9] C. S. Unnikrishnan, IndIGO and Ligo-India Scope and Plans for Gravitational Wave Research and Precision Metrology in India, International Journal of Modern Physics D **22**, 1341010 (2013), arXiv:1510.06059 [physics.ins-det].
- [10] C. S. Unnikrishnan, LIGO-India: A decadal assessment on its scope, relevance, progress and future, International Journal of Modern Physics D **33**, 2450025 (2024).
- [11] C. Messick *et al.*, Analysis framework for the prompt discovery of compact binary mergers in gravitational-wave data, Phys. Rev. D **95**, 042001 (2017).
- [12] B. Ewing *et al.*, Performance of the low-latency GstLAL inspiral search towards LIGO, Virgo, and KAGRA's fourth observing run, Phys. Rev. D **109**, 042008 (2024).
- [13] L. Tsukada *et al.*, Improved ranking statistics of the GstLAL inspiral search for compact binary coalescences, Phys. Rev. D **108**, 043004 (2023).
- [14] S. Sakon *et al.*, Template bank for compact binary mergers in the fourth observing run of Advanced LIGO, Advanced Virgo, and KAGRA, Phys. Rev. D **109**, 044066 (2024).
- [15] P. Joshi, L. Tsukada, C. Hanna, S. Adhicary, D. Mukherjee, W. Niu, S. Sakon, D. Singh, P. Baral, A. Baylor, K. Cannon, S. Caudill, B. Cousins, J. D. E. Creighton, B. Ewing, H. Fong, R. N. George, P. Godwin, R. Harada, Y.-J. Huang, R. Huxford, J. Kennington, S. Kuwahara, A. K. Y. Li, R. Magee, D. Meacher, C. Messick, S. Morisaki, A. Pace, C. Posnansky, A. Ray, S. Sachdev, S. Schmidt, U. Shah, R. Tapia, K. Ueno, A. Viets, L. Wade, M. Wade, Z. Yarbrough, and N. Zhang, New Methods for Offline GstLAL Analyses, arXiv e-prints , arXiv:2506.06497 (2025), arXiv:2506.06497 [gr-qc].
- [16] K. Cannon, S. Caudill, C. Chan, B. Cousins, J. D. Creighton, B. Ewing, H. Fong, P. Godwin, C. Hanna, S. Hooper, R. Huxford, R. Magee, D. Meacher, C. Messick, S. Morisaki, D. Mukherjee, H. Ohta, A. Pace, S. Privitera, I. de Ruiter, S. Sachdev, L. Singer, D. Singh, R. Tapia, L. Tsukada, D. Tsuna, T. Tsutsui, K. Ueno, A. Viets, L. Wade, and M. Wade, GstLAL: A software framework for gravitational wave discovery, SoftwareX **14**, 100680 (2021).
- [17] B. P. Abbott *et al.* (LIGO Scientific Collaboration and Virgo Collaboration), GW170817: Observation of Gravitational Waves from a Binary Neutron Star Inspiral, Phys. Rev. Lett. **119**, 161101 (2017).
- [18] K. Cannon, R. Cariou, A. Chapman, M. Crispin-Ortuzar, N. Fotopoulos, M. Frei, C. Hanna, E. Kara, D. Keppel, L. Liao, S. Privitera, A. Searle, L. Singer, and A. Weinstein, TOWARD EARLY-WARNING DETECTION OF GRAVITATIONAL WAVES FROM COMPACT BINARY COALESCENCE, The Astrophysical Journal **748**, 136 (2012).
- [19] GStreamer, <https://gstreamer.freedesktop.org/documentation/index.html>.
- [20] SGNL, <https://git.ligo.org/greg/sgnl> ().
- [21] SGN, <https://git.ligo.org/greg/sgn> ().
- [22] Y.-J. Huang, C. Hanna, B. Ewing, P. Godwin, J. Gonçalves, R. Magee, C. Messick, L. Tsukada, Z. Yarbrough, P. Joshi, J. Kennington, W. Niu, J. Rollins, and U. Shah, Scalable matched-filtering pipeline for gravitational-wave searches of compact binary mergers, Phys. Rev. D **112**, 082002 (2025).
- [23] SGN-TS, <https://git.ligo.org/greg/sgn-ts> ().
- [24] SGN-LIGO, <https://git.ligo.org/greg/sgn-ligo> ().
- [25] LIGO Scientific Collaboration, LIGO Algorithm Library - LALSuite, free software (GPL) (2018).
- [26] L. Tsukada, K. Cannon, C. Hanna, D. Keppel, D. Meacher, and C. Messick, Application of a zero-latency whitening filter to compact binary coalescence gravitational-wave searches, Phys. Rev. D **97**, 103009 (2018).
- [27] STRIKE, <https://git.ligo.org/greg/strike>.
- [28] Bottle: Python web framework, <https://bottlepy.org/docs/dev/>.
- [29] InfluxDB, <https://www.influxdata.com/>.
- [30] Grafana, <https://grafana.com/>.
- [31] S. S. Chaudhary, A. Toivonen, G. Waratkar, G. Mo, D. Chatterjee, S. Antier, P. Brockill, M. W. Coughlin, R. Essick, S. Ghosh, S. Morisaki, P. Baral, A. Baylor, N. Adhikari, P. Brady, G. C. Davies, T. D. Canton, M. Cavaglia, J. Creighton, S. Choudhary, Y.-K. Chu, P. Clearwater, L. Davis, T. Dent, M. Drago, B. Ewing, P. Godwin, W. Guo, C. Hanna, R. Huxford, I. Harry, E. Katsavounidis, M. Kovalam, A. K. Li, R. Magee, E. Marx, D. Meacher, C. Messick, X. Morice-Atkinson, A. Pace, R. D. Pietri, B. Piotrkowski, S. Roy, S. Sachdev, L. P. Singer, D. Singh, M. Szczepanczyk, D. Tang, M. Trevor, L. Tsukada, V. Villa-Ortega, L. Wen, and D. Wysocki, Low-latency gravitational wave alert products and their performance at the time of the fourth LIGO-

- Virgo-KAGRA observing run, Proceedings of the National Academy of Sciences **121**, e2316474121 (2024), <https://www.pnas.org/doi/pdf/10.1073/pnas.2316474121>.
- [32] D. W. Hogg, Distance measures in cosmology (1999), arXiv:astro-ph/9905116.
- [33] H.-Y. Chen, D. E. Holz, J. Miller, M. Evans, S. Vitale, and J. Creighton, Distance measures in gravitational-wave astrophysics and cosmology, Class. Quant. Grav. **38**, 055010 (2021), arXiv:1709.08079 [astro-ph.CO].