Locality-Sensitive Hashing for Earthquake Detection: A Case Study of Scaling Data-Driven Science (Extended Version)

Kexin Rong*, Clara E. Yoon†, Karianne J. Bergen‡, Hashem Elezabi*,
Peter Bailis*, Philip Levis*, Gregory C. Beroza†
Stanford University

ABSTRACT

In this work, we report on a novel application of Locality Sensitive Hashing (LSH) to seismic data at scale. Based on the high waveform similarity between reoccurring earthquakes, our application identifies potential earthquakes by searching for similar time series segments via LSH. However, a straightforward implementation of this LSH-enabled application has difficulty scaling beyond 3 months of continuous time series data measured at a single seismic station. As a case study of a data-driven science workflow, we illustrate how domain knowledge can be incorporated into the workload to improve both the efficiency and result quality. We describe several end-to-end optimizations of the analysis pipeline from pre-processing to post-processing, which allow the application to scale to time series data measured at multiple seismic stations. Our optimizations enable an over 100× speedup in the end-to-end analysis pipeline. This improved scalability enabled seismologists to perform seismic analysis on more than ten years of continuous time series data from over ten seismic stations, and has directly enabled the discovery of 597 new earthquakes near the Diablo Canyon nuclear power plant in California and 6123 new earthquakes in New Zealand.

PVLDB Reference Format:
Kexin Rong, Clara E. Yoon, Karianne J. Bergen, Hashem Elezabi, Peter Bailis, Philip Levis, Gregory C. Beroza. Locality-Sensitive Hashing for Earthquake Detection: A Case Study Scaling Data-Driven Science [Innovative Systems and Applications]. *PVLDB*, 11 (5): xxxx-yyyy, 2018.
DOI: https://doi.org/TBD

1. INTRODUCTION

Locality Sensitive Hashing (LSH) [29] is a well studied computational primitive for efficient nearest neighbor search in high-dimensional spaces. LSH hashes items into low-dimensional spaces such that similar items have a higher collision probability in the hash table. Successful applications of LSH include entity resolution [66], genome sequence comparison [18], text and image search [41, 52], near duplicate detection [20, 46], and video identification [37].

*Department of Computer Science
†Department of Geophysics
‡Institute for Computational and Mathematical Engineering

DOI: https://doi.org/TBD

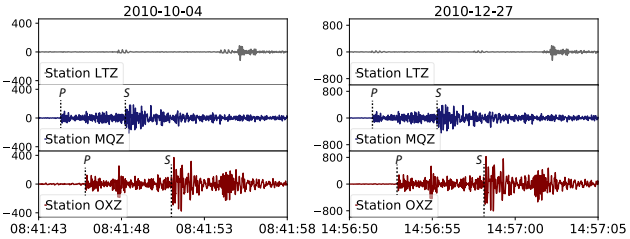

Figure 1: Example of near identical waveforms between occurrences of the same earthquake two months apart, observed at three seismic stations in New Zealand. The stations experience increased ground motions upon the arrivals of seismic waves (e.g., P and S waves). This paper scales LSH to over 30 billion data points and discovers 597 and 6123 new earthquakes near the Diablo Canyon nuclear power plant in California and in New Zealand, respectively.

In this paper, we present an innovative use of LSH—and associated challenges at scale—in large-scale earthquake detection across seismic networks. Earthquake detection is particularly interesting in both its abundance of raw data and scarcity of labeled examples:

First, seismic data is large. Earthquakes are monitored by seismic networks, which can contain thousands of seismometers that continuously measure ground motion and vibration. For example, Southern California alone has over 500 seismic stations, each collecting continuous ground motion measurements at 100Hz. As a result, the network has collected over ten trillion (10^{13}) data points in the form of time series in the past decade alone [5].

Second, despite large measurement volumes, only a small fraction of earthquake events are cataloged, or confirmed and hand-labeled. As earthquake magnitude (i.e., size) decreases, the frequency of earthquake events increases exponentially. Worldwide, major earthquakes (magnitude 7+) occur approximately once a month, while magnitude 2.0 and smaller earthquakes occur several thousand times a day. At low magnitudes, it is difficult to detect earthquake signals because earthquake energy approaches the noise floor, and conventional seismological analyses can fail to disambiguate between signal and noise. Nevertheless, detecting these small earthquakes is important in uncovering sources of earthquakes [24, 32], improving the understanding of earthquake mechanics [49, 59], and better predicting the occurrences of future events [38].

To take advantage of the large volume of unlabeled raw measurement data, seismologists have developed an unsupervised, data-driven earthquake detection method, Fingerprint And Similarity Thresholding (FAST), based on waveform similarity [25]. Specifically, seismic sources repeatedly generate earthquakes over the course of days, months or even years, and these earthquakes show near identical waveforms when recorded at the same seismic station, regardless of the earthquake's magnitude [27, 57]. Figure 1

illustrates this phenomenon by depicting two similar earthquakes that are two months apart, observed at three seismic stations in New Zealand. By applying LSH to identify similar waveforms from seismic data, seismologists were able to discover new, low-magnitude earthquakes without knowledge of prior earthquake events.

However, despite early success, seismologists had difficulty scaling their LSH-based analysis beyond 3-month of time series data (7.95×10^8 data points) at a single seismic station [24]. The FAST implementation faces severe scalability challenges. Contrary to what LSH theory suggests, the actual LSH runtime in FAST grows near quadratically with the input size due to correlations in the seismic signals: in an initial performance benchmark, the similarity search took 5 CPU-days to process 3 months of data, and, with a $5\times$ increase in dataset size, LSH query time increased by $30\times$. In addition, station-specific repeated background noise leads to an overwhelming number of similar but non-earthquake time series matches, both crippling throughput and seismologists' ability to sift through the output, which can number in the hundreds of millions of events. Ultimately, these scalability bottlenecks prevented seismologists from making use of the decades of data at their disposal.

In this paper, we show how systems, algorithms, and domain expertise can go hand-in-hand to deliver substantial scalability improvements for this seismological analysis. Via algorithmic design, optimization using domain knowledge, and data engineering, we scale the FAST workload to years of continuous data at multiple stations. In turn, this scalability has enabled new scientific discoveries, including previously unknown earthquakes near a nuclear reactor in San Luis Obispo, California, and in New Zealand.

Specifically, we build a scalable end-to-end earthquake detection pipeline comprised of three main steps. First, the fingerprint extraction step encodes time-frequency features of the original time series into compact binary fingerprints that are more robust to small variations. To address the bottleneck caused by repeating non-seismic signals, we apply domain-specific filters based on the frequency bands and the frequency of occurrences of earthquakes. Second, the search step applies LSH on the binary fingerprints to identify all pairs of similar time series segments. We pinpoint high hash collision rates caused by physical correlations in the input data as a core culprit of LSH performance degradation and alleviate the impact of large buckets by increasing hash selectivity while keeping the detection threshold constant. Third, the alignment step significantly reduces the size of detection results and confirms seismic behavior by performing spatiotemporal correlation with nearby seismic stations in the network [14]. To scale this analysis, we leverage domain knowledge of the invariance of the time difference between a pair of earthquake events across all stations at which they are recorded.

In summary, as an innovative systems and applications paper, this work makes several contributions:

- We report on a new application of LSH in seismology as well as a complete end-to-end data science pipeline, including non-trivial pre-processing and post-processing, that scales to a decade of continuous time series for earthquake detection.
- We present a case study for using domain knowledge to improve the accuracy and efficiency of the pipeline. We illustrate how applying seismological domain knowledge in each component of the pipeline is critical to scalability.
- We demonstrate that our optimizations enable a cumulative two order-of-magnitude speedup in the end-to-end detection pipeline. These quantitative improvements enable qualitative discoveries: we discovered 597 new earthquakes near the Diablo Canyon nuclear power plant in California and 6123 new earthquakes in New Zealand, allowing seismologists to determine the size and shape of nearby fault structures.

Beyond these contributions to a database audience, our solution is an open source tool, available for use by the broader scientific community. We have already run workshops for seismologists at Stanford [2] and believe that the pipeline can not only facilitate targeted seismic analysis but also contribute to the label generation for supervised methods in seismic data [50].

The rest of the paper proceeds as follows. We review background information about earthquake detection in Section 2 and discuss additional related work in Section 3. We give a brief overview of the end-to-end detection pipeline and key technical challenges in Section 4. Sections 5, 6 and 7 present details as well as optimizations in the fingerprint extraction, similarity search and the spatiotemporal alignment steps of the pipeline. We perform a detailed evaluation on both the quantitative performance improvements of our optimizations as well as qualitative results of new seismic findings in Section 8. In Section 9, we reflect on lessons learned and conclude.

2. BACKGROUND

With the deployment of denser and increasingly sensitive sensor arrays, seismology is experiencing rapid growth of high-resolution data [30]. Seismic networks consisting of up to thousands of sensors record years of continuous seismic data streams, typically at 100Hz frequencies. This massive volume of data has fueled strong interest in the seismology community to apply and develop scalable algorithms that automate and improve the monitoring and prediction of earthquake events [21, 40, 42].

In this work, we focus on the problem of detecting new, low-magnitude earthquakes from historical seismic data. Earthquakes, which are primarily caused by the rupture of geological faults, radiate energy that travels through the Earth in the form of seismic waves. Seismic waves induce ground motion that is recorded by seismometers. Modern seismometers typically include 3 components that measure simultaneous ground motion along the north-south, east-west, and vertical axes. Ground motions along each of these three axes are recorded as a separate *channel* of time series data.

Channels capture complementary signals for different seismic waves, such as the P-wave and the S-wave. The P-waves travel along the direction of propagation, like sound, while the S-waves travel perpendicular to the direction of propagation, like ocean waves. The vertical channel, therefore, better captures the up and down motions caused by the P-waves while the horizontal channels better capture the side to side motions caused by the S-waves. P-waves travel the fastest and are the first to arrive at seismic stations, followed by the slower but usually larger amplitude S-waves. Hence, the P-wave and S-wave of an earthquake typically register as two "big wiggles" on the ground motion measurements (Figure 1). These impulsive arrivals of seismic waves are example characteristics of earthquakes that seismologists look for in the data.

While it is easy for human eyes to identify large earthquakes on a single channel, accurately detecting small earthquakes usually requires looking at data from multiple channels or stations. These low-magnitude earthquakes pose a challenge for conventional methods for detection, which we outline below. Traditional energy-based earthquake detectors such as a short-term average (STA)/long-term average (LTA) identify earthquake events by their impulsive, high signal-to-noise seismic waves. However, these detectors are prone to high false positive and false negative rates at low magnitudes, especially with noisy backgrounds and/or weak signals as in a station that is distant from a low magnitude event [28]. Template matching, or the waveform cross-correlation with template waveforms of known earthquakes, has proven more effective for detecting known seismic signals in noisy data [15, 58]. However, the template match-

ing relies on template waveforms of prior events and is not suitable for discovering events from unknown sources.

As a result, almost all earthquakes greater than magnitude 5 are detected [26]. However, an estimated 1.5 million earthquakes with magnitude between 2 and 5 are not detected by conventional means, and 1.3 million of these are between magnitude 2 and 2.9. This is based on the magnitude frequency distribution of earthquakes [31]. We are interested in detecting these low-magnitude earthquakes missing from public earthquake catalogs to better understand earthquake mechanics and sources, which inform seismic hazard estimates and prediction [32, 38, 49, 59].

The earthquake detection pipeline we study in the paper is an unsupervised and data-driven approach that does not rely on supervised (i.e., labeled) examples of prior earthquake events, and is designed to complement existing, supervised detection methods. As in template matching, the method we optimize takes advantage of the high similarity between waveforms generated by reoccurring earthquakes. However, instead of relying on waveform templates from known events, the pipeline leverages the recurring nature of seismic activity to detect similar waveforms in time and across stations. To do so, the pipeline performs an all-pair time series similarity search, treating each segment of the input waveform data as a "template" for potential earthquakes. This pipeline will not detect an earthquake that occurs only once and is not similar enough to any other earthquakes in the input data, so, to improve detection recall, it is critical to be able to scale the analysis to input data with longer duration (e.g., years instead of weeks or months).

3. RELATED WORK

In this section, we address related work in earthquake detection, LSH-based applications and time series similarity search.

Earthquake Detection. The original FAST work appeared in the seismology community, and has proven a useful tool in scientific discovery [24, 25]. In this paper, we present FAST to a database audience for the first time, and report on both the pipeline composition and optimization from a computational perspective. The results presented in this paper are the result of over a year of collaboration between our database research group and the Stanford earthquake seismology research group. The optimizations we present in this paper and the resulting scalability results of the optimized pipeline have not previously been published. We believe this represents a useful and innovative application of LSH to a real domain science tool that will be of interest to both the database community and researchers of LSH and time-series analytics.

The problem of earthquake detection is decades old [6], and many classic techniques—many of which are in use today—were developed for an era in which humans manually inspected seismographs for readings [35, 67]. With the rise of machine learning and large-scale data analytics, there has been increasing interest in further automating these techniques. While FAST is optimized to find many small-scale earthquakes, alternative approaches in the seismology community utilize template matching [15, 58], social media [55], and machine learning including neural networks [8, 65]. Most recently, with sufficient training data, supervised approaches such as [50] have the advantage of being able to detect non repeating earthquake events. In contrast, our LSH-based detection method is unsupervised, and therefore does not rely on labeled earthquake events. We compare against two supervised methods [50, 56] and show that our unsupervised pipeline is able to detect qualitatively different events from the existing earthquake catalog.

Locality Sensitive Hashing. In this work, we perform a detailed case study of the practical challenges of applying LSH to the domain of seismology and propose domain-specific solutions to address these challenges. We do not contribute to the advance of the state-of-the-art LSH algorithms. Instead, we show that classic LSH techniques, with domain-specific optimizations, can lead to scientific discoveries, while posing substantial computational challenges at scale. Existing work shows that LSH performance is sensitive to key parameters such as number of hashes [23, 53]; we provide supporting evidence and detailed analysis on the performance implication of LSH parameters in our application domain. In addition to the core LSH techniques, we also present nontrivial preprocessing and postprocessing steps that enable an end-to-end detection pipeline, including spatiotemporal alignment of LSH matches.

While there are exciting opportunities to significantly speed up the similarity search on GPUs [34], this work focuses on CPU workloads. To preserve the integrity of the established workflow, we focus on optimizing the existing MinHash based LSH rather than replacing it with potentially more efficient LSH variants such as LSH forest [10] and multi-probe LSH [45]. While we share certain observations with prior work that parallelizes and distributes SimHash LSH (a different LSH family based on cosine similarity) [62], we focus on optimizing MinHash LSH for a single node and enables speedups via domain-specific optimizations. We also provide performance benchmarks against alternative similarity search algorithms such as set similarity joins [47] and an alternative LSH library based on recent theoretical advances in LSH for cosine similarity [7]. We believe the resulting experience report—and several key challenges we encountered including fingerprint non-uniformity and repeated noise—as well as our open source implementation will be valuable to researchers developing LSH techniques in the future.

Time Series Analytics. Time series analytics is a core topic in large-scale data analytics and data mining [39, 44, 69]. In our application domain, we utilize time series similarity search as a core workhorse in earthquake detection. There are a range of computational primitives that could be applied to this problem [22], including Euclidean distance and its variants [70], Dynamic Time Warping [51], and edit distance [63]. However, the input time series coming from seismometer measurements to our application is high frequency (e.g. 100Hz) and potentially noisy. Therefore, small time-shifts, outliers and scaling can result in large changes in time-domain metrics [19]. Instead, we encode time-frequency features of the input time series into binary vectors and focus on the Jaccard similarity. This feature extraction procedure is an adaptation of the Waveprint algorithm [9] originally designed for audio data; the key modification made for seismic data was to focus on frequency features that are the most discriminative from background noise, such that the average similarity between non-seismic signals are reduced [13]. An alternative binary representation models time series as points on a grid, and uses the non-empty grid cells as a set representation of the time series [48]. However, this representation does not take advantage of the physical properties distinguishing background from seismic signals.

4. PIPELINE OVERVIEW

In this section, we provide an overview of our unsupervised detection pipeline from feature extraction to post-processing. We provide a detailed description of each step in the pipeline—and our associated optimizations—in later sections, referenced inline.

The input of the detection pipeline consists of continuous ground motion measurements in the form of time series, collected from multiple stations in the seismic network. The output is a list of potential earthquakes, specified in the form of timestamps when the seismic wave arrives at each station. From there, seismologists can

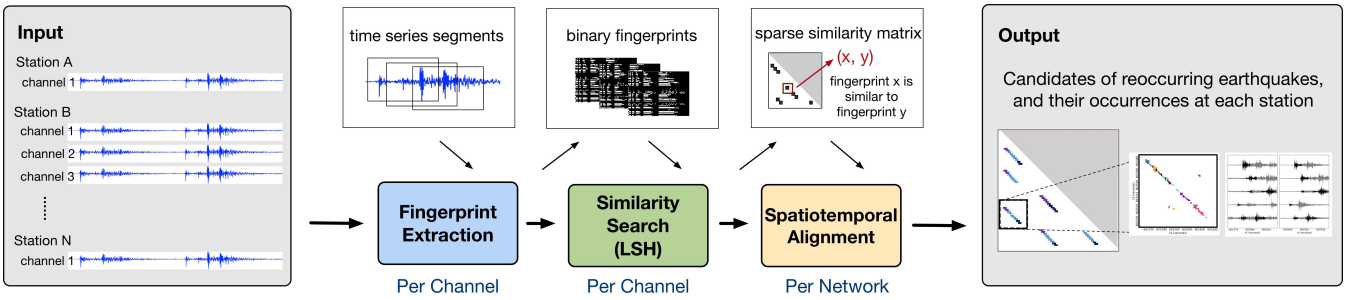

Figure 2: The three steps of the end-to-end earthquake detection pipeline: fingerprinting transforms time series into binary vectors (Section 5); similarity search identifies pairs of similar binary vectors (Section 6); alignment aggregates and reduces false positives in results (Section 7).

compare with public earthquake catalogs to identify new events, and visually inspect the measurements to confirm seismic findings.

Figure 2 gives an overview of the end-to-end detection pipeline, which includes three main steps: fingerprint extraction, similarity search, and spatiotemporal alignment. For each input time series, or continuous ground motion measurements from a seismic channel, the algorithm slices the input into short windows of overlapping time series segments and encodes time-frequency features of each window into a binary fingerprint; the similarity of the fingerprints resembles that of the original waveforms (Section 5). Given binary fingerprints generated from the first step, the algorithm then performs an all pairs similarity search via LSH to identify pairs of highly similar fingerprints (Section 6). The similarity search outputs a sparse similarity matrix for each input time series, where the non-zero entries of the matrix represent similar pairs of fingerprints. Like a traditional associator that maps earthquake detections at each station to a consistent seismic source, in the spatiotemporal alignment stage, the algorithm combines, filters and clusters the outputs from all seismic channels to generate a list of candidate earthquake detections with high confidence (Section 7).

A naïve implementation of the pipeline imposes serious scalability challenges. For example, we observed a degradation of LSH performance in our application domain caused by the non-uniformity and correlation in the binary fingerprints; the correlations induce undesired LSH hash collisions, which significantly increase the number of lookups per similarity search query (Section 6.3). The similarity search does not distinguish seismic from non-seismic signals. In the presence of repeating background signals, similar noise waveforms could outnumber similar earthquake waveforms, leading to more than an order of magnitude slow down in runtime and increase in output size (Section 6.5). As the input time series and the output of the similarity search becomes larger, the pipeline must adapt to data sizes that are too large to fit into main memory (Section 6.4, 7.2).

In this paper, we focus on single-machine, main-memory execution on commodity servers and multicore processors. We parallelize the pipeline within a given server but otherwise do not distribute the computation to multiple servers. In principle, the parallelization efforts extend to distributed execution. However, given the poor quadratic scalability of the unoptimized pipeline, distribution would not have been a viable option for scaling to our target of a decade of data. As a result of the optimizations described in this paper, we are able to scale to decades of data on a single node without requiring distribution. However, we view distributed execution as worthwhile extension for future work.

In the remaining sections of this paper, we describe the design decisions as well as performance optimizations for each pipeline component. Most of our optimizations focus on the all pairs similarity search, where our initial implementation exhibited near quadratic growth in runtime with the input size. We show in the evaluation that, these optimizations enable speedups of more than two orders of magnitude in the end-to-end detection pipeline.

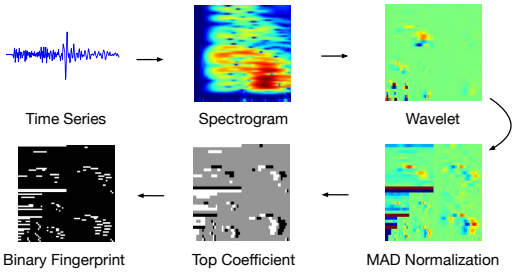

Figure 3: The fingerprinting algorithm encodes time-frequency features of the original time series into binary vectors.

5. FINGERPRINT EXTRACTION

In this section, we describe the fingerprint extraction step that encodes the time-frequency features of the input time series into compact binary vectors for similarity search. We begin with an overview of the existing fingerprinting algorithm [13] and discuss the benefits of using fingerprints in place of the time series (Section 5.1). We then describe a new optimization that parallelizes and accelerates the fingerprinting generation via sampling (Section 5.2).

5.1 Fingerprint Overview

The fingerprinting step transforms the continuous time series data into compact binary vectors (fingerprints) for similarity search. The fingerprints are compact encoding of the time-frequency features of the time series data. The Jaccard similarity between binary fingerprints, defined as the size of the intersection of the non-zero entries divided by the size of the union, approximates the pairwise waveform similarity of the original time series segments. Similar fingerprinting techniques have already seen a successful application on audio signals [9]. Compared to directly computing similarity on the time series, fingerprinting incorporates additional frequency-domain features into the detection and provides more robustness with respect to translation and small variations [13].

Figure 3 illustrates the individual steps for fingerprinting:

1. **Spectrogram** Compute the spectrogram, a time-frequency representation, of the time series data. Slice the spectrogram it into short overlapping segments using a sliding window and smooth by downsampling each segment into a spectral image of fixed dimensions.

2. **Wavelet Transform** Compute two-dimensional discrete Haar wavelet transform on each spectral image. The wavelet coefficients serve as a lossy compression of the spectral images.
3. **Normalization** Normalize each wavelet coefficient by its median and the median absolute deviation (MAD[1]) on the full, background dominated dataset.
4. **Top coefficient** Extract the top K most anomalous wavelet coefficients, or the largest coefficients after MAD normalization, from each spectral image. By selecting the most anomalous coefficients, we focus only on coefficients that are most distinct from coefficients that characterize noise, which empirically leads to better detection results.
5. **Binarize** Binarize the signs and positions of the top wavelet coefficients. Concretely, we encode the sign of each normalized coefficient using 2 bits: $-1 \rightarrow 01$, $0 \rightarrow 00$, $1 \rightarrow 10$.

5.2 Optimization: MAD via sampling

The fingerprint extraction is implemented via scientific modules such as scipy, numpy and PyWavelets in Python. While its runtime grows linearly with input size, it can take several days to fingerprint ten years of time series data.

In the unoptimized procedure, normalizing the wavelet coefficients requires two full passes on the data. The first pass calculates the median and the MAD for each coefficient across the whole population, and the second pass normalizes the wavelet representations of each fingerprint accordingly. Given the median and MAD for each wavelet coefficient, we can divide the input time series into partitions and normalize in parallel. Therefore, the calculation of the median and MAD remains the runtime bottleneck.

We accelerate the median and MAD calculation by sampling a small portion of the input data to approximate the true median and MAD. We randomly sample a continuous segment representing X% of the time series for each fixed interval (e.g. 1% every hour), and compute the median and the MAD from the sampled data.

The confidence interval for MAD with a sample size of n shrinks with $n^{1/2}$ [60]. In the evaluation, we investigate the trade-off between speed and accuracy for different sampling rates (Section 8.3). We empirically find that, on one month of input time series data, sampling provides an order of magnitude speedup with almost no loss in accuracy. For input time series of longer duration, sampling 1% or less of the input can suffice.

6. LSH-BASED SIMILARITY SEARCH

In this section, we present the pipeline's time series similarity search, based on LSH. We begin with a description of the baseline implementation of the similarity search (Section 6.1), upon which we build the optimizations. Our contributions include: a cache friendly hash signature generation procedure (Section 6.2), an empirical analysis of the impact of hash collisions and LSH parameters on query performance (Section 6.3), partition and parallelization of LSH that reduce the runtime and memory usage (Section 6.4), and finally, two domain-specific filters (an improved bandpass filter and a new occurrence filter) that improve both the performance and detection quality of the search (Section 6.5).

6.1 Similarity Search Overview

Reoccurring earthquakes generated from nearby seismic sources show near-identical waveforms at the same seismic station. Given continuous ground motion measurements from a seismic station, we utilize similarity search to identify all pairs of time series segments that are above a similarity threshold as candidate earthquake events.

[1]For $X = \{x_1, x_2, ..., x_n\}$, the MAD is defined as the median of the absolute deviations from the median: $MAD = median(|x_i - median(X)|)$

We perform an approximate similarity search via MinHash LSH and identify pairs of binary fingerprints whose Jaccard similarity exceeds a predefined threshold [17]. Intuitively, MinHash LSH performs a random projection of high-dimensional data into lower dimensional space, hashing similar items to the same hash table "bucket" with high probability. Instead of performing the naïve n^2 comparisons between all pairs of fingerprints, we need only compare fingerprints that share a hash bucket, significantly reducing the number of comparisons required. We refer to the ratio of average comparisons per query to the size of the dataset as *selectivity*, a machine-independent proxy for efficiency [23].

The MinHash value of a fingerprint is defined as the first non-zero element of the fingerprint under a given random permutation of fingerprint elements. For each fingerprint and each hash mapping, the minimum hash value is computed among all nonzero elements in the fingerprint. Let p denote the collision probability of the hash signature with a single MinHash value. The number of hash functions k effectively decreases the collision probability of the hash signature to p^k by combining every k MinHash values into a single hash signature ([43] provides an overview).

Hash table construction. We initialize each hash table with its corresponding set of hash signatures by mapping each hash signature to a list of fingerprints that share the same signature. Empirically, we find that using $t = 100$ hash tables suffices for our application, and there is little gain in further increasing the number of hash tables.

Search. The search queries the hash tables for each fingerprint's near neighbor candidates, or other fingerprints that share the query fingerprint's hash buckets. We keep track of the number of times the query fingerprint and candidates have matching hash signature in the hash tables, and identify candidates with matches above a threshold. The number of matches is also used as a proxy for the confidence of the similarity for the final step of the pipeline.

6.2 Optimization: Hash signature generation

In this subsection, we present both memory access pattern and algorithmic improvements to speed up the generation of hash signatures. We show that, together, the optimizations can lead to an over $3\times$ improvement in hash generation time (Section 8.1).

Similar to observations made for SimHash (a different hash family for angular distances) [62], a naïve implementation of the MinHash generation can suffer from poor memory locality due to the sparsity of input data. SimHash functions are evaluated as a dot product between the input and hash vectors, while MinHash functions are evaluated as a minimum of hash mappings corresponding to non-zero elements of the input. For sparse input, both functions access scattered, non-contiguous elements in the hash mapping array, causing an increase in cache misses. We improve the memory access pattern by blocking the access to the MinHash hash table. For each fingerprint, we use dimensions of the fingerprint, rather than hash functions, as the main loop. This way, the hash value lookups for each non-zero element in the fingerprint are blocked into a contiguous row in the hash mapping array. For our application, this loop order has the additional advantage of exploiting the high overlap (e.g. over 60% in one example) between neighboring fingerprints. The overlap means that previously accessed elements in hash mappings are likely to get reused, further improving the access pattern.

We further speed up the hash signature generation by replacing MinHash with Min-Max hash. In MinHash, only the minimum hash value is kept for each random hash mapping. Min-Max hash, on the other hand, keep both the min and the max value which means that it generates two hashes from each hash mapping. Therefore, to generate hash signatures of the same length, Min-Max hash re-

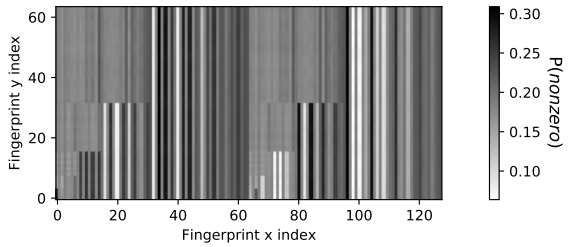

Figure 4: Probability that each element in the fingerprint is equal to 1, averaged over 15.7M fingerprints, each of dimension 8192, generated from a year of time series data. The heatmap shows that some elements of the fingerprint are much more likely to be non-zero compared to others.

duces the number of required hash mappings to half. Previous work showed the Min-Max hash is an unbiased estimator of pairwise Jaccard similarity, and achieves similar and sometimes smaller mean squared error (MSE) in estimating pairwise Jaccard similarity in practice [33]. We include pseudocode for the optimized hash signature calculation in Appendix D of extended Technical Report [54].

6.3 Optimization: Alleviating hash collisions

Perhaps surprisingly, our initial LSH implementation demonstrated poor scaling with the input size: with a 5× increase in put size, the runtime increased by 30×. In this subsection, we analyze the cause of LSH performance degradation and the performance implications of core LSH parameters in our application.

Cause of hash collisions. Poor distribution of hash signatures can lead to large LSH hash buckets and high query *selectivity*, significantly degrading the performance of LSH queries [10, 36]. For example, in the extreme case when the hash table contains a single hash bucket, the *selectivity* equals 1 and the LSH performance is equivalent to that of the naive $O(n^2)$ all pairs search.

We found that the large hash buckets reflect physical correlations in the data. Elements of the binary fingerprints encode physical properties of the waveform data, so the probability that each element in the fingerprint is non-zero is not uniform (Figure 4). Moreover, elements of the fingerprints are not guaranteed to be independent, meaning that the probability that two elements a_i, a_j both equal 1 can be much larger than the product of the probability that each element equals 1 ($\mathbb{P}[a_i = 1, a_j = 1] > \mathbb{P}[a_i = 1] \times \mathbb{P}[a_j = 1]$).

This correlation has a direct impact on the probability of collision of MinHash signatures. For example, if the hash signature contains k independent MinHash values of a fingerprint, and two of the non-zero elements responsible for the MinHash values are dependent, the hash signature has similar collision probability as the signature that contains only $k-1$ MinHash values. In other words, more fingerprints are likely to be hashed to the same bucket under this signature. For fingerprints shown in Figure 4, the largest 0.1% of the hash buckets contain an average of 32.9% of the total fingerprints for hash tables constructed with 6 hash functions.

Performance impact of LSH parameters. The precision and recall of the LSH is controlled via two key parameters: the number of hash functions k and the number of hash table matches m. Intuitively, using k hash functions is equivalent to requiring two fingerprints agree at k randomly selected non-zero positions. Therefore, the larger the number of hash functions, the lower the probability of collision. To improve recall, we increase the number of independent permutations to make sure that similar fingerprints can land in the same hash bucket with high probability.

Given two fingerprints with Jaccard similarity s, the probability

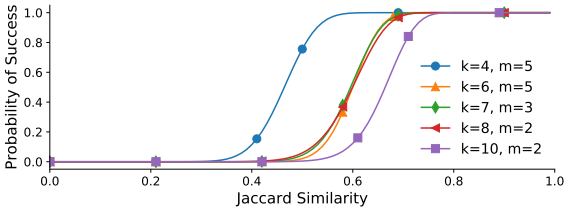

Figure 5: Theoretical probability of a successful search versus Jaccard similarity between fingerprints (k: number of hash functions, m: number of matches). Different LSH parameter settings can have near identical detection probability with vastly different runtime.

that with k hash functions, the fingerprints are hashed to the same bucket at least m times out of $t = 100$ hash tables is: $\mathbb{P}[s] = 1 - \sum_{i=0}^{m-1}[\binom{t}{i}(1-s^k)^{t-i}(s^k)^i]$. The probability of detection success as a function of Jaccard similarity has the form of an S-curve (Figure 5). The S-curve shifts to the right with the increase in the number of hash functions k or the m threshold, increasing the Jaccard similarity threshold for LSH. Figure 5 illustrates a near-identical probability of success curve under different parameter settings.

Due to the presence of correlation in the input data, LSH parameters that result in the same success probability curve can have vastly different runtime in practice. As we increase the number of hash functions, the expected average size of hash buckets decreases. However, we find that, for seismic data, raising the number of hash functions can lead to an order of magnitude speedup in the similarity search. We investigate the impact of LSH parameters in Section 8.3.

One caveat of lowering the number of matches is that the probability of spurious matches increases. These spurious matches can be suppressed by scaling up the number of matches required and the number of total hash tables, at the cost of larger memory usage. We illustrate the empirical trade-off between the decrease in similarity search time and the increase in output size in Section 8.1.

6.4 Optimization: Partitioning

In this subsection, we describe the partition and parallelization of the LSH that further reduce its runtime and memory footprint.

Partition. Using a 1-second lag for adjacent fingerprints results in around 300M total fingerprints for 10 years of time series data. Given a hash signature of 64 bits and 100 total hash tables, the total size of hash signature is approximately 250 GB. In addition, to avoid expensive disk I/O, we also need to keep all hash tables in memory for lookups. Taken together, this requires several hundred gigabytes of memory, which can exceed available main memory.

To scale to larger input data on a single node with the existing LSH implementation, we perform similarity search in partitions. We partition the hash signatures into equal-lengthed blocks and populate the hash tables with one partition at a time, while still keeping the lookup table of fingerprints to hash signatures in memory. During query, we output matches between fingerprints in the current partition with all other fingerprints and subsequently repeat this process for each partition. The partitioned search yields identical results to the original search, with the benefit that only a subset of the fingerprints are stored in the hash tables in memory. We can partition the lookup table of hash signatures similarly to further reduce memory. We illustrate the performance and memory trade-offs under different numbers of partitions in Section 8.3.

The idea of populating the hash table with a subset of the input could also be favorable for performing a small number of nearest neighbor queries on a large dataset, e.g., a thousand queries on a million items. There are two ways to execute the queries. We can hash the full dataset and then perform a thousand queries to retrieve

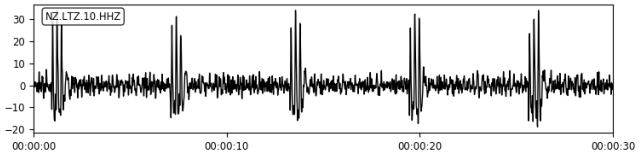

Figure 6: The short, three-spike pattern is an example of similar and repeating background signals not due to seismic activity. These repeating noise patterns cause scalability challenges for LSH.

near neighbor candidates in each query item's hash bucket; alternatively, we can populate the table with only query items and for every other item in the dataset, check whether it is hashed to an existing bucket in the table. While the two methods yield identical query results, the latter could be 8.6× faster since the cost of initializing the hash table dominates that of the search.

We can further improve LSH performance and reduce memory usage with the more space efficient variants such as multi-probe LSH [45]. However, given that the alignment step uses the number of hash buckets shared between fingerprints as a proxy for similarity, and that switching to a multi-probe implementation would alter this similarity measure, we preserve the original LSH implementation for backwards compatibility with FAST. We demonstrate the potential benefits of adopting multi-probe LSH in Section 8.4.

Parallelization. The hash generation procedure can be easily parallelized once the hash mappings are generated: we can partition the input fingerprints and calculate hash signatures in parallel. Similarly, the query procedure can be parallelized by running nearest neighbor queries for different fingerprints in parallel. The only critical section in the search is on outputting nearest neighbors to file, which we optimize by writing to binary files. We show in Section 8.3 that the total hash signature generation time and similarity search time reduces linearly with the number of processes.

6.5 Optimization: Domain-specific filters

Like many other sensor measurements, seismometer readings can be noisy. In this subsection, we address a practical challenge of the detection pipeline, where similar non-seismic signals dominate seismic findings in runtime and detection results. We show that by leveraging domain knowledge, we can greatly increase both the efficiency and the quality of the detection.

Filtering irrelevant frequencies. Some input time series contain station-specific narrow-band noise that repeats over time. Patterns of the repeating noise are captured in the fingerprints and are identified as near neighbor, or earthquake candidates in the similarity search.

To address this problem, we apply a bandpass filter to exclude frequency bands that show high average amplitudes and repeating patterns while containing low seismic activities. The bandpass filter is selected manually by examining short spectrogram samples, typically an hour long, of the input time series, based on seismological knowledge, and typical bandpass filter ranges span from 2 to 20Hz. Prior work [13, 14, 24, 25] proposes the idea of filtering irrelevant frequencies, but only on input time series. We extend the filter to the fingerprinting algorithm and cutoff spectrogram at the corner of the bandpass filter, which empirically improves detection performance. We perform a quantitative evaluation of the impact of bandpass filters on both the runtime and result quality (Section 8.2).

Removing correlated noise. Repeating non-seismic signals can also occur in frequency bands containing rich earthquake signals. Figure 6 shows an example of strong repeating background signals from a New Zealand seismic station. A large cluster of repeating signals with high pairwise similarity could produce nearest neighbor matches that dominate the similarity search, leading to a 10× increase in runtime and an over 100× increase in output size compared to results from similar stations. This poses both problems for computational scalability and for seismological interpretability.

We develop an occurrence filter for the similarity search by exploiting the rarity of the earthquake signals. Specifically, if a specific fingerprint is generating many nearest neighbor matches during a short duration of time, we can be fairly confident that it is not an earthquake signal. This observation generally holds except for special scenarios such as volcanic earthquakes [12].

During the similarity search, we dynamically generate a list of fingerprints to exclude from future search. If the number of near neighbor candidates a fingerprint generates is larger than a predefined percentage of the total fingerprints, we exclude this fingerprint as well as its near neighbors from future similarity search. To capture repeating noise over a short duration of time, the filter can be applied on top of the partitioned search. In this case, the filtering threshold is defined as the percentage of fingerprints in the current partition, rather than of the whole dataset. On the example dataset above, this approach filtered out around 30% of the total fingerprints. We evaluate the effect of the occurrence filter on different datasets under different filtering thresholds in Section 8.2.

7. SPATIOTEMPORAL ALIGNMENT

The LSH-based similar search outputs pairs of similar fingerprints (or waveforms) from the input data, without knowledge of whether the pairs correspond to actual earthquake events. In this section, we show that by incorporating domain knowledge, we are able to significantly reduce the size of the output and prioritize seismic findings in the similarity search results. We briefly summarize the aggregation and filtering techniques on the level of seismic channels, seismic stations and seismic networks introduced in a recent paper in seismology [14] (Section 7.1). We then describe the implementation challenges and our out-of-core adapatations enabling the algorithm to scale to large output volume (Section 7.2).

7.1 Alignment Overview

The similarity search computes a sparse similarity matrix $\mathcal{M}$, where the non-zero entry $\mathcal{M}[i, j]$ represents the similarity of fingerprints i and j. In order to identify weak events in low signal-to-noise ratio settings, seismologists set lenient detection thresholds for the similarity search, resulting in large output sizes in practice. For example, one year of time series data can easily generate 100G of output, or more than 5 billion pairs of similar fingerprints. Since it is infeasible for seismologists to inspect all results manually, we need to automatically filter and align the similar fingerprint pairs into a list of potential earthquakes with high confidence. Based on algorithms proposed in a recent work in seismology [14], we seek to reduce similarity search results at the level of seismic channels, stations and also across a seismic network. Figure 7 gives an overview of the spatiotemporal alignment procedure.

Channel Level. Seismic channels at the same station experience ground movements at the same time. Therefore, we can directly merge detection results from each channel of the station by summing the corresponding similarity matrix. Given that earthquake-triggered fingerprint matches tend to register at multiple channels whereas matches induced by local noise might only appear on one channel, we can prune detections by imposing a slightly higher similarity threshold on the combined similarity matrix. This is to make sure that we include either matches with high similarity, or weaker matches registered at more than one channel.

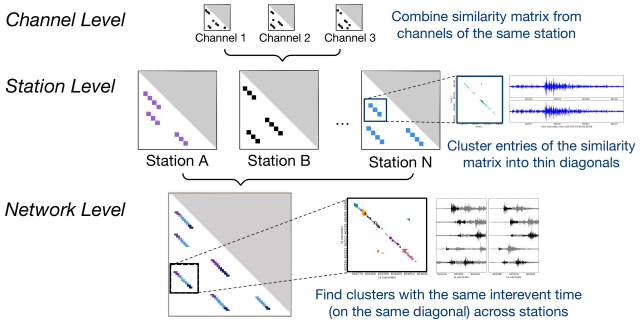

Figure 7: The alignment procedure combines similarity search outputs from all channels in the same station (Channel Level), groups similar fingerprint matches generated from the same pair of reoccurring earthquakes (Station Level), and checks across seismic stations to reduce false positives in the final detection list (Network Level).

Station Level. Given a combined similarity matrix for each seismic station, domain scientists have found that earthquake events can be characterized by thin diagonal shaped clusters in the matrix, which corresponds to a group of similar fingerprint pairs separated by a constant offset [14]. The constant offset represents the time difference, or the inter-event time, between a pair of reoccurring earthquake events. One pair of reoccurring earthquake events can generate multiple fingerprint matches in the similarity matrix, since event waveforms are longer than a fingerprint time window. We exclude "self-matches" generated from adjacent/overlapping fingerprints that are not attributable to reoccurring earthquakes. After grouping similar fingerprint pairs into clusters of thin diagonals, we reduce each cluster to a few summary statistics, such as the bounding box of the diagonal, the total number of similar pairs in the bounding box, and the sum of their similarity. Compared to storing every similar fingerprint pair, the clusters and summary statistics significantly reduce the size of the output.

Network Level. Earthquake signals also show strong temporal correlation across the seismic network, which we exploit to further suppress non-earthquake matches. Since an earthquake's travel time is only a function of its distance from the source but not of the magnitude, reoccurring earthquakes generated from the same source take a fixed travel time from the source to the seismic stations on each occurrence. Assume that an earthquake originated from source X takes δt_A and δt_B to travel to seismic stations A and B and that the source generates two earthquakes at time t_1 and t_2 (Figure 8). Station A experiences the arrivals of the two earthquakes at time $t_1+\delta t_A$ and $t_2+\delta t_A$, while station B experiences the arrivals at $t_1+\delta t_B$ and $t_2+\delta t_B$. The inter-event time Δt of these two earthquake events is independent of the location of the stations:

$$\Delta t = (t_2+\delta t_A)-(t_1+\delta t_A) = (t_2+\delta t_B)-(t_1+\delta t_B) = t_2-t_1.$$

This means that in practice, diagonals with the same offset Δt and close starting times at multiple stations can be attributed to the same earthquake event. We require a pair of earthquake events to be observed at more than a user-specified number of stations in order to be considered as a detection.

On a run with 7 to 10 years of time series data from 11 seismic stations (27 channels), the postprocessing procedure effectively reduced the output from more than 2 Terabytes of similar fingerprint pairs to around 30K timestamps of potential earthquakes.

7.2 Implementation and Optimization

The volume of similarity search output poses serious challenges for the alignment procedure, as we often need to process results

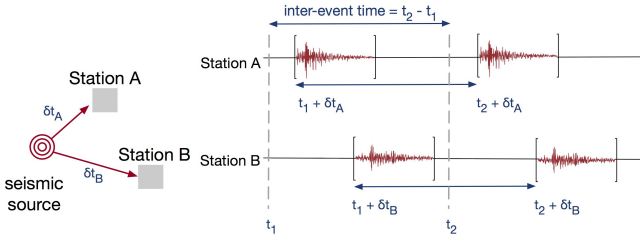

Figure 8: Earthquakes from the same seismic sources has a fixed travel time to each seismic station (e.g. δt_A, δt_B in the figure). The inter-event time between two occurrences of the same earthquake is invariant across seismic stations.

larger than the main memory of a single node. In this subsection, we describe our implementation and the new out-of-core adaptations of the algorithm that enable the scaling to large output volumes.

Similarity search output format. The similarity search produces outputs that are in the form of triplets. A triplet $(dt, idx1, sim)$ is a non-zero entry in the similarity matrix, which represents that fingerprint $idx1$ and $(idx1+dt)$ are hashed into the same bucket sim times (out of t independent trials). We use sim as an approximation of the similarity between the two fingerprints.

Channel. First, given outputs of similar fingerprint pairs (or the non-zero entries of the similarity matrix) from different channels at the same station, we want to compute the combined similarity matrix with only entries above a predefined threshold.

Naïvely, we could update a shared hashmap of the non-zero entries of the similarity matrix for each channel in the station. However, since the hashmap might not fit in the main memory on a single machine, we utilize the following sort-merge-reduce procedure instead:

1. In the sorting phase, we perform an external merge sort on the outputs from each channel, with dt as the primary sort key and $idx1$ as the secondary sort key. That is, we sort the similar fingerprint pairs first by the diagonal that they belong to in the similarity matrix, and within the diagonals, by the start time of the pairs.
2. In the merging phase, we perform a similar external merge sort on the already sorted outputs from each channel. This is to make sure that all matches generated by the same pair of fingerprint $idx1$ and $idx1+dt$ at different channels can be concentrated in consecutive rows of the merged file.
3. In the reduce phase, we traverse through the merged file and combine the similarity score of consecutive rows of the file that share the same dt and $idx1$. We discard results that have combined similarity smaller than the threshold.

Station. Given a combined similarity matrix for each seismic station, represented in the form of its non-zero entries sorted by their corresponding diagonals and starting time, we want to cluster fingerprint matches generated by potential earthquake events, or cluster non-zero entries along the narrow diagonals in the matrix.

We look for sequences of detections (non-zero entries) along each diagonal dt, where the largest gap between consecutive detections is smaller than a predefined gap parameter. Empirically, permitting a gap help ensure an earthquake's P and S wave arrivals are assigned to the same cluster. Identification of the initial clusters along each diagonal dt requires a linear pass through the similarity matrix. We then interactively merge clusters in adjacent diagonals $dt-1$ and $dt+1$, with the restriction that the final cluster has a relatively narrow width. We store a few summary statistics for each cluster (e.g. the cluster's bounding box, total number of entries) as well

as prune small clusters and isolated fingerprint matches, which significantly reduces the output size.

The station level clustering dominates the runtime in the spatiotemporal alignment. In order to speed up the clustering, we partition the similarity matrix according to the diagonals, or ranges of dts of the matched fingerprints, and perform clustering in parallel on each partition. A naïve equal-sized partition of the similarity matrix could lead to missed detections if a cluster split into two partitions gets pruned in both due to the decrease in size. Instead, we look for proper points of partition in the similarity matrix where there is a small gap between neighboring occupied diagonals. Again, we take advantage of the ordered nature of similarity matrix entries. We uniformly sample entries in the similarity matrix, and for every pair of neighboring sampled entries, we only check the entries in between for partition points if the two sampled entries lie on diagonals far apart enough to be in two partitions. Empirically, a sampling rate of around 1% works well for our datasets in that most sampled entries are skipped because they are too close to be partitioned.

Network. Given groups of potential events at each station, we perform a similar summarization across the network in order to identify subsets of the events that can be attributed to the same seismic source. In principle, we could also partition and parallelize the network detection. In practice, however, we found that the summarized event information at each station is already small enough that it suffices to compute in serial.

8. EVALUATION

In this section, we perform both quantitative evaluation on performances of the detection pipeline, as well as qualitative analysis of the detection results. Our goal is to demonstrate that:

1. Each of our optimizations contributes meaningfully to the performance improvement; together, our optimizations enable an over 100× speed up in the end-to-end detection pipeline.
2. Incorporating domain knowledge in the pipeline improves both the performance and the quality of the detection.
3. The improved scalability of the pipeline enables new scientific discoveries on two public datasets: we discovered 597 new earthquakes from a decade of seismic data near the Diablo Canyon nuclear power plant in California, as well as 6123 new earthquakes from a year of seismic data from New Zealand.

Dataset. We evaluate on two public datasets used in seismological analyses with our domain collaborators. The first dataset includes 1 year of 100Hz time series data (3.15 billion points per station) from 5 seismic stations (LTZ, MQZ, KHZ, THZ, OXZ) in New Zealand. We use the vertical channel (usually the least noisy) from each station [3]. The second dataset of interest includes 7 to 10 years of 100Hz time series data from 11 seismic stations and 27 total channels near the Diablo Canyon power plant in California [4].

Experimental Setup. We report results from evaluating the pipeline on a server with 512GB of RAM and two 28-thread Intel Xeon E5-2690 v4 2.6GHz CPUs. Our test server has L1, L2, L3 cache sizes of 32K, 256K and 35840K. We report the runtime averages from multiple trials.

8.1 End-to-end Evaluation

In this subsection, we report the runtime breakdown of the baseline implementation of the pipeline, as well as the effects of applying different optimizations.

To evaluate how our optimizations scale with data size, we evaluate the end-to-end pipeline on 1 month and 1 year of time series data from station LTZ in the New Zealand dataset. We applied a bandpass filter of 3-20Hz on the original time series to exclude noisy low-frequency bands. For fingerprinting, we used a sliding window with length of 30 seconds and slide of 2 seconds, which results in 1.28M binary fingerprints for 1 month of time series data (15.7M for one year), each of dimension 8192; for similarity search, we use 6 hash functions, and require a detection threshold of 5 matches out of 100 hash tables. We further investigate the effect of varying these parameters in the microbenchmarks in Section 8.3.

Figure 9 shows the cumulative runtime after applying each optimization. Cumulatively, our optimizations scale well with the size of the dataset, and enable an over 100× improvement in end-to-end processing time. We analyze each of these components in turn:

First, we apply a 1% occurrence filter (+ occur filter, Section 6.5) during similarity search to exclude frequent fingerprint matches generated by repeating background noise. This enables a 2-5× improvement in similarity search runtime while reducing the output size by 10-50×, reflected in the decrease in postprocessing time.

Second, we further reduce the search time by increasing the number of hash functions to 8 and lowering the detection threshold to 2 (+ increase #funcs, Section 6.3). While this increases the hash signature generation and output size, it enables around 10× improvement in search time for both datasets.

Third, we reduce the hash signature generation time by improving the cache locality and reducing the computation by with Min-Max hash instead of MinHash (+ locality MinMax, Section 6.2), which leads to a 3× speedup for both datasets.

Fourth, we speed up fingerprinting by 2× by estimating MAD statistics with a 10% sample (+ MAD sample, Section 5.2).

Finally, we enable parallelism and run the pipeline with 12 processes (Section 5.2, 6.4, 7.2). As a result, we see an almost linear decrease in runtime in each part of the pipeline. We acknowledge that due to the overall lack of data dependencies in this scientific pipeline, parallelizations can already improve significant speedups.

The improved scalability enables us to scale analytics from 3 months to over 10 years of data. We discuss qualitative detection results from both datasets in Section 8.5.

8.2 Effect of domain-specific optimizations

Here, we investigate the effect of applying domain-specific optimizations to the pipeline. We demonstrate that incorporating domain knowledge could improve both performance and result quality.

Occurrence filter. We evaluate the effect of applying the frequency based filter during similarity search on the five stations from the New Zealand dataset. For this evaluation, we use a partition size of 1 month as the duration for the frequency threshold; a >1% threshold indicates that a fingerprint matches over 10K other fingerprints in the same month. We report the total percentage of fingerprints that are filtered under varying frequency thresholds in Table 1. We also evaluate the accuracy of the occurrence filter by comparing the timestamps of filtered fingerprints with the catalog of the arrival time of known earthquakes at each station. We report in Table 1 false positive rate, or the number of filtered earthquakes over the total number of cataloged events, of the filter under varying thresholds.

The results show that as occurrence filter becomes stronger, the percentage of filtered fingerprints and the false positive rate both increase. For seismic stations suffering from correlated noise, the occurrence filter can effectively eliminate a significant amount of fingerprints from the similarity search. For station LTZ, a >1% threshold filters out up to 30% of the total fingerprints without any false positives, which results in a 4× improvement in runtime. For other stations, the occurrence filter has little influence on the results. This is expected since these other stations do not have repeating noise signals such as those at station LTZ (Figure 6). In practice,

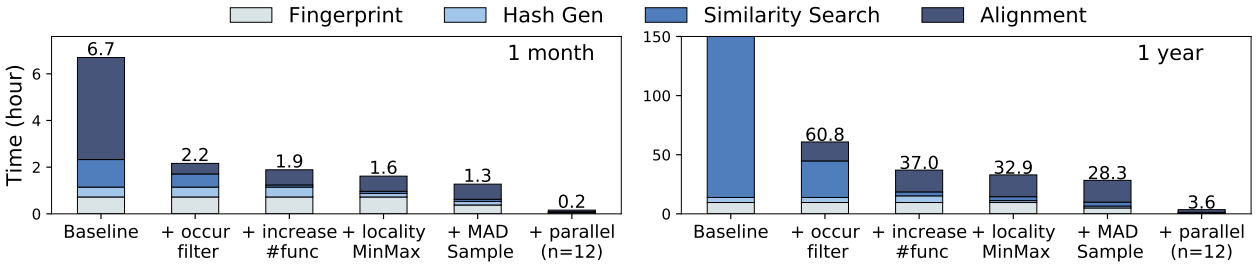

Figure 9: Factor analysis of processing 1 month (left) and 1 year (right) of 100Hz data from LTZ station in the New Zealand dataset. We show that each of our optimization contributes to the performance improvements, and enabled an over 100× speed up end-to-end.

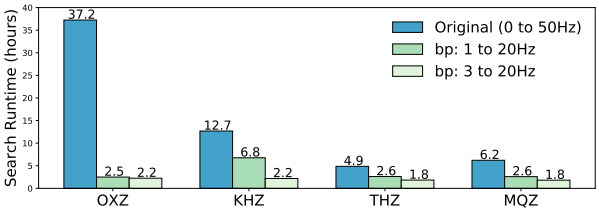

Figure 10: LSH runtime under different band pass filters. Matches of noise in the non-seismic frequency bands can lead to significant increase in runtime for unfiltered time series.

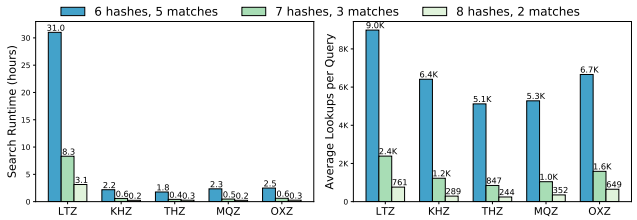

Figure 11: Effect of LSH parameters on similarity search runtime and average query lookups. Increasing the number of hash functions significantly decreases average number of lookups per query, which results in a 10× improvement runtime.

correlated noise is rather prevalent in seismic data. In the Diablo Canyon dataset for example, we applied the occurrence filter on three out of the eleven seismic stations in order for the similarity search to finish in a tractable time.

Bandpass filter. We compare similarity search on the same dataset (Nyquist frequency 50Hz) before and after applying bandpass filters. The first bandpass filter (bp: 1-20Hz) is selected as most seismic signals are under 20Hz; the second (bp: 3-20Hz) is selected after manually looking at samples spectrograms of the dataset and excluding noisy low frequencies. Figure 10 reports the similarity search runtime for fingerprints generated with different bandpass filters. Overall, similarity search suffers from additional matches generated from the noisy frequency bands outside the interests of seismology. For example, at station OXZ, removing the bandpass filter leads to a 16× increase in runtime and a 209× increase in output size.

We compare detection recall on 8811 catalog earthquake events for different bandpass filters. The recall for the unfiltered data (0-50Hz), the 1-20Hz and 3-20Hz bandpass filters are 20.3%, 23.7%, 45.2%, respectively. The overall low recall is expected, as we only used 4 (out of over 50) stations in the seismic network that contributes to the generation of catalog events. Empirically, a narrow, domain-informed bandpass filter focuses the comparison of fingerprint similarity only on frequencies that are characteristics of seismic events, leading to improved similarity between earthquake events and therefore increased recall. We provide guidelines for setting the bandpass filter in the extended report ([54], Appendix C).

8.3 Effect of pipeline parameters

In this section, we evaluate the effect of the space/quality and time trade-offs for core pipeline parameters.

MAD sampling rate. We evaluate the speed and quality trade-off for calculating the median and MAD of the wavelet coefficients for fingerprints via sampling. We measure the runtime and accuracy on the 1 month dataset in Section 8.1 (1.3M fingerprints) under varying sampling rates. Overall, runtime and accuracy both decrease with sampling rate as expected. For example, a 10% and 1% sampling rate produce fingerprints with 99.7% and 98.7% accuracy, while enabling a near linear speedup of 10.5× and 99.8×, respectively. Below 1%, runtime improvements suffer from a diminishing return, as the IO begins to dominate the MAD calculation in runtime–on this dataset, a 0.1% sampling rate only speeds up the MAD calculation by 350×. We include additional results of this trade-off in [54].

LSH parameters. We report runtime of the similarity search under different LSH parameters in Figure 11. As indicated in Figure 5, the three sets of parameters that we evaluate yield near identical probability of detection given Jaccard similarity of two fingerprints. However, by increasing the number of hash functions and thereby increasing the selectivity of hash signatures, we decrease the average number of lookups per query by over 10x. This results in around 10x improvement in similarity search time.

Number of partitions. We report the runtime and memory usage of the similarity search with varying number of partitions in Figure 12. As the number of partitions increases, the runtime increases slightly due to the overhead of initialization and deletion of hash tables. In contrast, memory usage decreases as we only need to keep a subset of the hash signatures in the hash table at any time. Overall, by increasing the number of partitions from 1 to 8, we are able to decrease the memory usage by over 60% while incurring less than 20% runtime overhead. This allows us to run LSH on larger datasets with the same amount of memory.

Parallelism. Finally, to quantify the speedups from parallelism, we report the runtime of LSH hash signature generation and similarity search using a varying number of processes. For hash signature generation, we report time taken to generate hash mappings as well as the time taken to compute Min-Max hash for each fingerprint. For similarity search, we use the fixed hash signature as input for each station and vary the number of processes assigned during the search. We show the results in Figure 13. Overall, hash signature generation scales almost perfectly (linearly) up to 32 processes, while similarity search scales slightly worse; both experience significant

	LTZ (1548 events)			MQZ (1544 events)			KHZ (1542 events)			THZ (1352 events)			OXZ (1248 events)		
Thresh	FP	Filtered	Time	FP	Filtered	Time	FP	Filtered	Time	FP	Filtered	Time	FP	Filtered	Time
>5.0%	0	0.09	149.3	0	0	2.8	0	0	2.2	0	0	2.4	0	0	2.6
>1.0%	0	30.1	31.0	0	0	2.7	0	0	2.3	0	0	2.3	0	0	2.6
>0.5%	0	31.2	32.1	0	0.09	2.8	0	0	2.4	0	0	2.4	0.08	0.08	2.7
>0.1%	0	32.1	28.6	0.07	0.3	2.7	0	0.03	2.4	0	0.02	2.3	0.08	0.17	2.6

Table 1: The table shows that the percentage of fingerprints filtered (Filtered) and the false positive rate (FP) both increase as the occurrence filter becomes stronger (from filtering matches above 5.0% to above 0.1%). The runtime (in hours) measures similarity search time.

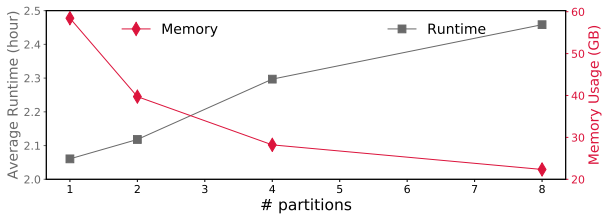

Figure 12: Runtime and memory usage for similarity search under a varying number of partitions. By increasing the number of search partitions, we are able to decrease the memory usage by over 60% while incurring less than 20% runtime overhead.

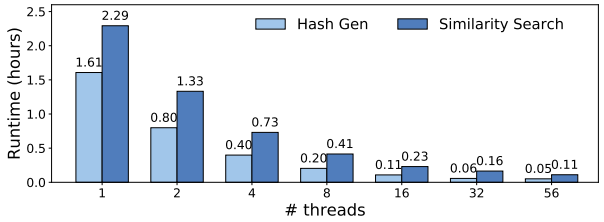

Figure 13: Hash generation scales near linearly up to 32 threads.

performance degradation running with all available threads.

8.4 Comparison with Alternatives

In this section, we evaluate against alternative similarity search algorithms and supervised methods. We include additional experiment details in the extended technical report ([54], Appendix A).

Alternative Similarity Search Algorithms. We compare the single-core query performance of our MinHash LSH to 1) an alternative open source LSH library FALCONN [1] 2) four state-of-the-art set similarity join algorithms: PPJoin [68], GroupJoin [16], AllPairs [11] and AdaptJoin [64]. We use 74,795 fingerprints with dimension 2048 and 10% non-zero entries, and a Jaccard similarity threshold of 0.5 for all libraries. Compared to exact algorithms like set similarity joins, the LSHs incur a 6% false negative rate. However, MinHash LSH enables a 24× to 65× speedup against FALCONN and 63× to 197× speedup against set similarity joins (Table 2). Characteristics of the input fingerprints contribute to the performance differences: the fixed number of non-zero entries in fingerprints makes pruning techniques in set similarity joins based on set length irrelevant; our results corroborate with previous findings that MinHash outperforms SimHash on binary, sparse input [61].

Supervised Methods. We report results evaluating two supervised models: WEASEL [56] and ConvNetQuake [50] on the Diablo Canyon dataset. Both models were trained on labeled catalog events (3585 events from 2010 to 2017) and randomly sampled noise windows at station PG.LMD. We also augment the earthquake training examples by 1) adding earthquake examples from another station PG.DCD 2) perturbing existing events with white noise 3) shifting

Algorithm	Average Query time	Speedup
MinHash LSH	36 μs	–
FALCONN vanilla LSH	.87ms	24×
FALCONN multi-probe LSH	2.4ms	65×
AdaptJoin [64]	2.3ms	63×
AllPairs [11]	7.1ms	197×
GroupJoin [16]	5.7ms	159×
PPJoin [68]	5.5ms	151×

Table 2: Single core per-datapoint query time for LSH and set similarity joins. MinHash LSH incurs a 6.6% false negative rate while enabling up to 197× speedup.

	WEASEL [56]	ConvNetQuake [50]
Test Catalog Acc. (%)	90.8	90.6
Test FAST Acc. (%)	68.0	70.5
True Negative Rate (%)	98.6	92.2
False Positive Rate (%)	90.0±5.88	90.0±5.88

Table 3: Supervised methods trained on catalog events exhibit high false positive rate and a 20% accuracy gap between predictions on catalog and FAST detected events.

the location of the earthquake event in the window. Table 3 reports test accuracy of the two models on a sample of 306 unseen catalog events and 449 new events detected by our pipeline (FAST events), as well as the false positive rate estimated from manual inspection of 100 random earthquake predictions. While supervised methods achieve high accuracy in classifying unseen catalog and noise events, they exhibit a high false positive rate (90±5.88%) and miss 30-32% of new earthquake events detected by our pipeline. The experiment suggests that unsupervised methods like our pipeline are able to detect qualitatively different events from the existing catalog, and that supervised methods are complements, rather than replacements, of unsupervised methods for earthquake detection.

8.5 Qualitative Results

We first report our findings in running the pipeline over a decade (06/2007 to 10/2017) of continuous seismic data from 11 seismic stations (27 total channels) near the Diablo Canyon nuclear power plant in central California. The chosen area is of special interest as there are many active faults near the power plant. Detecting additional small earthquakes in this region will allow seismologists to determine the size and shape of nearby fault structures, which can potentially inform seismic hazard estimates.

We applied station-specific bandpass filters between 3 and 12 Hz to remove repeating background noise from the time series. In addition, we applied the occurrence filter on three out of the eleven seismic stations that experienced corrupted sensor measurements. The number of input binary fingerprints for each seismic channel ranges from 180 million to 337 million; the similarity search runtime ranges from 3 hours to 12 hours with 48 processes.

Among the 5048 detections above our detection threshold, 397 detections (about 8%) were false positives, confirmed via visual inspection: 30 were duplicate earthquakes with a lower similarity,

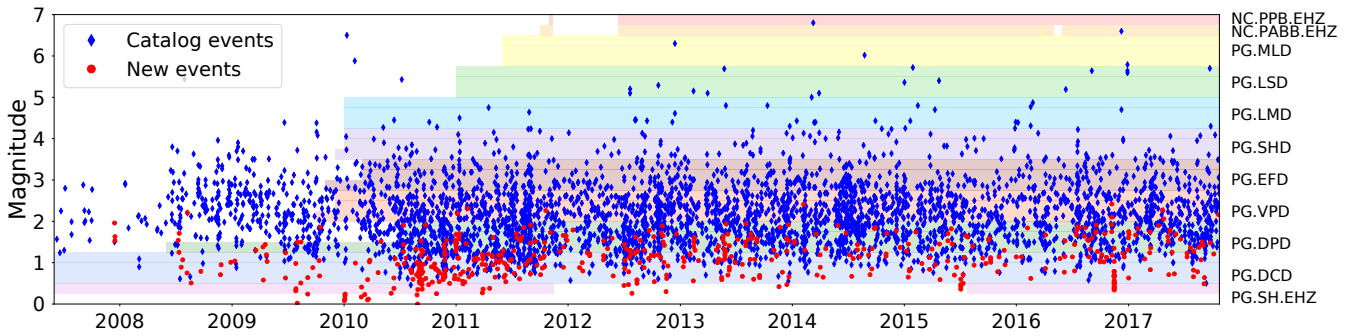

Figure 14: The left axis shows origin times and magnitude of detected earthquakes, with the catalog events marked in blue and new events marked in red. The colored bands in the right axis represent the duration of data used for detection collected from 11 seismic stations and 27 total channels. Overall, we detected 3957 catalog earthquakes (diamond) as well as 597 new local earthquakes (circle) from this dataset.

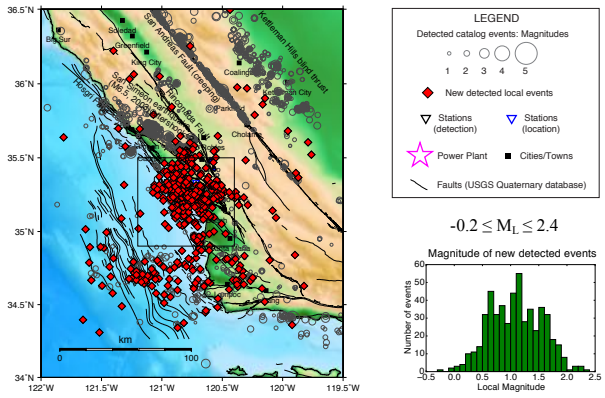

Figure 15: Overview of the location of detected catalog events (gray open circles) and new events (red diamonds). The pipeline was able to detect earthquakes close to the seismic network (boxed) as well as all over California.

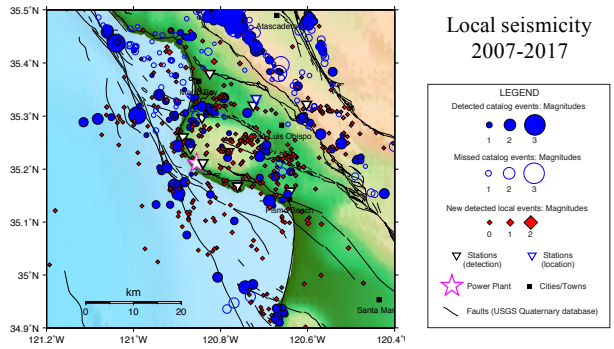

Figure 16: Zoom in view of locations of new detected earthquakes (red diamonds) and cataloged events (blue circles) near the seismic network (box in Figure 15). The new local earthquakes contribute detailed information about the structure of faults.

18 were catalog quarry blasts, 5 were deep teleseismic earthquakes (large earthquakes from >1000 km away). There were also 62 non-seismic signals detected across the seismic network; we suspect that some of these waveforms are sonic booms.

Overall, we were able to detect and locate 3957 catalog earthquakes, as well as 597 new local earthquakes. Figure 14 shows an overview of the origin time of detected earthquakes, which is spread over the entire ten-year span. The detected events include both low-magnitude events near the seismic stations, as well as larger events that are farther away. Figure 15 visualizes the locations of both catalog events and newly detected earthquakes, and Figure 16 zooms in on earthquakes in the vicinity of the power plant. Despite the low rate of local earthquake activity (535 total catalog events from 2007 to 2017 within the area shown in Figure 16), we were able to detect 355 new events that are between -0.2 and 2.4 in magnitude and located within the seismic network, where many active faults exist. We missed 261 catalog events, almost all of which originated from outside the network of our interest. Running the detection pipeline at scale enables scientists to discover earthquakes from unknown sources. These new detected local events will be used to determine the details of active fault structures near the power plant.

We are also actively working with our domain collaborators on additional analysis of the New Zealand dataset. The pipeline detected 11419 events, including 4916 catalog events, 355 teleseismic events, 6123 new local earthquakes and 25 false positives (noise waveforms) verified by the seismologists. We are preparing these results for publication in seismological venues, and expect to further improve the detection results by scaling up the analysis to more seismic stations over a longer duration of time.

9. CONCLUSION

In this work, we reported on a novel application of LSH to large-scale seismological data, as well as the challenges and optimizations required to scale the system to over a decade of continuous sensor data. This experience in scaling LSH for large-scale earthquake detection illustrates both the potential and the challenge of applying core data analytics primitives to data-driven domain science on large datasets. On the one hand, LSH and, more generally, time series similarity search, is well-studied, with scores of algorithms for efficient implementation: by applying canonical MinHash-based LSH, our seismologist collaborators were able to meaningfully analyze more data than would have been feasible via manual inspection. On the other hand, the straightforward implementation of LSH in the original FAST detection pipeline failed to scale beyond a few months of data. The particulars of seismological data—such as frequency imbalance in the time series and repeated background noise—placed severe strain on an unmodified LSH implementation and on researchers attempting to understand the output. As a result, the seismological discoveries we have described in this paper would not have been possible without domain-specific optimizations to the detection pipeline. We believe that these results have important implications for researchers studying LSH (e.g., regarding the importance of skew resistance) and will continue to bear fruit as we scale the system to even more data and larger networks.

10. REFERENCES

[1] FALCONN - FAst Lookups of Cosine and Other Nearest Neighbors. https://github.com/falconn-lib/falconn.

[2] FAST Detection Pipeline. https://github.com/stanford-futuredata/FAST.

[3] GeoNet. https://www.geonet.org.nz/data/tools/FDSN.

[4] NCEDC. http://service.ncedc.org/.

[5] SCEDC (2013): Southern California Earthquake Center. Caltech. Dataset. doi:10.7909/C3WD3xH1.

[6] D. L. Anderson. *Theory of the Earth*. Blackwell scientific publications, 1989.

[7] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal lsh for angular distance. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1225–1233. Curran Associates, Inc., 2015.

[8] W. Astuti, R. Akmeliawati, W. Sediono, and M. Salami. Hybrid technique using singular value decomposition (svd) and support vector machine (svm) approach for earthquake prediction. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(5):1719–1728, 2014.

[9] S. Baluja and M. Covell. Audio fingerprinting: Combining computer vision & data stream processing. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 2, pages II–213. IEEE, 2007.

[10] M. Bawa, T. Condie, and P. Ganesan. Lsh forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web*, pages 651–660. ACM, 2005.

[11] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 131–140, New York, NY, USA, 2007. ACM.

[12] A. F. Bell, S. Hernandez, H. E. Gaunt, P. Mothes, M. Ruiz, D. Sierra, and S. Aguaiza. The rise and fall of periodic 'drumbeat' seismicity at tungurahua volcano, ecuador. *Earth and Planetary Science Letters*, 475:58 – 70, 2017.

[13] K. Bergen, C. Yoon, and G. C. Beroza. *Scalable Similarity Search in Seismology: A New Approach to Large-Scale Earthquake Detection*, pages 301–308. Springer International Publishing, Cham, 2016.

[14] K. J. Bergen and G. C. Beroza. Detecting earthquakes over a seismic network using single-station similarity measures. *Geophysical Journal International*, 213(3):1984–1998, 2018.

[15] D. Bobrov, I. Kitov, and L. Zerbo. Perspectives of cross-correlation in seismic monitoring at the international data centre. *Pure and Applied Geophysics*, 171(3):439–468, Mar 2014.

[16] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *Proc. VLDB Endow.*, 6(1):1–12, Nov. 2012.

[17] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society.

[18] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.

[19] F.-P. Chan, A.-C. Fu, and C. Yu. Haar wavelets for efficient similarity search of time-series: with and without time warping. *IEEE Transactions on knowledge and data engineering*, 15(3):686–705, 2003.

[20] O. Chum, J. Philbin, and A. Zisserman. Near Duplicate Image Detection: min-Hash and tf-idf Weighting.

[21] G. Corts et al. Using principal component analysis to improve earthquake magnitude prediction in japan. jzx049:1–14, 10 2017.

[22] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, Aug. 2008.

[23] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling lsh for performance tuning. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 669–678, New York, NY, USA, 2008. ACM.

[24] C. E. Yoon, Y. Huang, W. L. Ellsworth, and G. C. Beroza. Seismicity during the initial stages of the guy-greenbrier, arkansas, earthquake sequence: Induced seismicity in guy, arkansas. 11 2017.

[25] C. E Yoon, O. O'Reilly, K. Bergen, and G. C Beroza. Earthquake detection through computationally efficient similarity search. 1:e1501057–e1501057, 12 2015.

[26] E. R. Engdahl, R. van der Hilst, and R. Buland. Global teleseismic earthquake relocation with improved travel times and procedures for depth determination. *Bulletin of the Seismological Society of America*, 88(3):722–743, 1998.

[27] R. J. Geller and C. S. Mueller. Four similar earthquakes in central california. *Geophysical Research Letters*, 7(10):821–824, 1980.

[28] S. J. Gibbons and F. Ringdal. The detection of low magnitude seismic events using array-based waveform correlation. *Geophysical Journal International*, 165(1):149–166, 2006.

[29] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[30] Y. J. Gu, A. Okeler, S. Contenti, K. Kocon, L. Shen, and K. Brzak. Broadband seismic array deployment and data analysis in alberta. *CSEG Recorder, September*, pages 37–44, 2009.

[31] B. GUTENBERG and C. F. RICHTER. Magnitude and energy of earthquakes. *Annals of Geophysics*, 9(1):1–15, 1956.

[32] Y. Huang and G. C. Beroza. Temporal variation in the magnitude-frequency distribution during the guy-greenbrier earthquake sequence. *Geophysical Research Letters*, 42(16):6639–6646, 2015. 2015GL065170.

[33] J. Ji, J. Li, S. Yan, Q. Tian, and B. Zhang. Min-max hash for jaccard similarity. In *2013 IEEE 13th International Conference on Data Mining*, pages 301–309, Dec 2013.

[34] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[35] M. Joswig. Pattern recognition for earthquake detection. *Bulletin of the Seismological Society of America*, 80(1):170, 1990.

[36] B. Kang and K. Jung. Robust and efficient locality sensitive hashing for nearest neighbor search in large data sets. Citeseer.

[37] Z. Kang, W. T. Ooi, and Q. Sun. Hierarchical, non-uniform locality sensitive hashing and its application to video identification. In *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)*, volume 1, pages 743–746 Vol.1, June 2004.

[38] A. Kato and S. Nakagawa. Multiple slow-slip events during a foreshock sequence of the 2014 iquique, chile mw 8.1 earthquake. *Geophysical Research Letters*, 41(15):5420–5427, 2014. 2014GL061138.

[39] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery*, 7(4):349–371, 2003.

[40] Q. Kong, R. M. Allen, L. Schreier, and Y.-W. Kwon. Myshake: A smartphone seismic network for earthquake early warning and beyond. *Science Advances*, 2(2), 2016.

[41] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2130–2137, Sept 2009.

[42] R. Kulkarni. A review of application of data mining in earthquake prediction. 01 2012.

[43] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[44] T. W. Liao. Clustering of time series data–a survey. *Pattern recognition*, 38(11):1857–1874, 2005.

[45] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 950–961. VLDB Endowment, 2007.

[46] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 141–150, New York, NY, USA, 2007. ACM.

[47] W. Mann, N. Augsten, and P. Bouros. An empirical evaluation of set similarity join techniques. *Proc. VLDB Endow.*, 9(9):636–647, May 2016.

[48] J. Peng, H. Wang, J. Li, and H. Gao. Set-based similarity search for

time series. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2039–2052. ACM, 2016.

[49] Z. Peng and P. Zhao. Migration of early aftershocks following the 2004 parkfield earthquake. *Nature Geoscience*, 2:877 EP –, 11 2009.

[50] T. Perol, M. Gharbi, and M. Denolle. Convolutional neural network for earthquake detection and location. *Science Advances*, 4(2), 2018.

[51] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

[52] B. Rao and E. Zhu. Searching web data using minhash lsh. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2257–2258, New York, NY, USA, 2016. ACM.

[53] B. Rao and E. Zhu. Searching web data using minhash lsh. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2257–2258. ACM, 2016.

[54] K. Rong, C. E. Yoon, K. J. Bergen, H. Elezabi, P. Bailis, P. Levis, and G. C. Beroza. Locality-sensitive hashing for earthquake detection: A case study scaling data-driven science (extended version). `http://kexinrong.github.io/quake.pdf`.

[55] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM, 2010.

[56] P. Schäfer and U. Leser. Fast and accurate time series classification with weasel. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 637–646, New York, NY, USA, 2017. ACM.

[57] D. P. Schaff and G. C. Beroza. Coseismic and postseismic velocity changes measured by repeating earthquakes. *Journal of Geophysical Research: Solid Earth*, 109(B10):n/a–n/a, 2004. B10302.

[58] D. P. Schaff and F. Waldhauser. One magnitude unit reduction in detection threshold by cross correlation applied to parkfield (california) and china seismicity. 2010.

[59] D. R. Shelly, D. P. Hill, F. Massin, J. Farrell, R. B. Smith, and T. Taira. A fluid-driven earthquake swarm on the margin of the yellowstone caldera. *Journal of Geophysical Research: Solid Earth*, 118(9):4872–4886, 2013.

[60] W. Shi and B. M. G. Kibria. On some confidence intervals for estimating the mean of a skewed population. *International Journal of Mathematical Education in Science and Technology*, 38(3):412–421, 2007.

[61] A. Shrivastava and P. Li. In defense of minhash over simhash. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 33, Reykjavik, Iceland, 2014.

[62] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *Proc. VLDB Endow.*, 6(14):1930–1941, Sept. 2013.

[63] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings 18th International Conference on Data Engineering*, pages 673–684, 2002.

[64] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: An adaptive framework for similarity join and search. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 85–96, New York, NY, USA, 2012. ACM.

[65] J. Wang and T.-l. Teng. Identification and picking of s phase using an artificial neural network. *Bulletin of the Seismological Society of America*, 87(5):1140, 1997.

[66] Q. Wang, M. Cui, and H. Liang. Semantic-aware blocking for entity resolution. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):166–180, Jan 2016.

[67] M. Withers, R. Aster, C. Young, J. Beiriger, M. Harris, S. Moore, and J. Trujillo. A comparison of select trigger algorithms for automated global seismic phase and event detection. *Bulletin of the Seismological Society of America*, 88(1):95, 1998.

[68] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, 36(3):15:1–15:41, Aug. 2011.

[69] Z. Xing, J. Pei, and E. Keogh. A brief survey on sequence classification. *SIGKDD Explor. Newsl.*, 12(1):40–48, Nov. 2010.

[70] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 385–394, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

False Negative (%)	Query time (ms)	# Hash Tables	# Probes
6.7	0.87	85	85
6.5	2.4	50	120
0.54	2.4	50	400
0.36	2.0	200	200

Table 4: Average query time and false negative rate under different FALCONN parameter settings.

APPENDIX

A. COMPARISON TO ALTERNATIVES

A.1 Exact Similarity Search Algorithms

In this subsection, we investigate the performance and accuracy tradeoff between using MinHash LSH and exact algorithms for similarity search. We focus the comparison on set similarity joins, a line of exact join algorithms that identifies all pairs of sets above a similarity threshold from two collections of sets [47]. State-of-the-art set similarity joins avoid exhaustively computing all pairs of set similarities via a filter-verification approach, such that only "promising" candidates that survive the filtering and verification are examined for the final join.

We report single-core query time of our MinHash LSH implementation and four state-of-the-art algorithms for set similarity joins: PPJoin [68], GroupJoin [16], AllPairs [11] and AdaptJoin [64]. For the set similarity joins, we use an open-source implementation (C++) from a recent benchmark paper, which is reported to be faster than the original implementations on almost all data points tested [47].

We use a set of fingerprints generated from 20 hours of continuous time series data, which includes 74,795 input fingerprints with dimension 2048 and 10% non-zero entries. For set similarity joins, we transform each binary fingerprint into a set of integer tokens of the non-zero entries, with the tokens chosen such that larger integer tokens are more frequent than smaller ones.

We found that with a Jaccard similarity threshold of 0.5, the MinHash LSH incurs a 6.6% false negative rate while enabling 63× to 200× speedups compared to set similarity join algorithms (Table 2). Among the four tested algorithms, AdaptJoin achieves the best query performance as a result of the small candidate set size enabled by its sophisticated filters. This is different from the benchmark paper's observation that expensive filters do not pay off and often lead to the slowest runtime [47]. One important difference in our experiment is that the input fingerprints have a fixed number of non-zero entries; as a result, the corresponding input sets have equal length. Therefore, filtering and pruning techniques based on set length do not apply to our dataset.

A.2 Alternative LSH library

In this subsection, we compare the query performance of our similarity search to an alternative and more advanced open source LSH library. We were unable to find an existing high-performance implementation of LSH for Jaccard similarity, so we instead compare to FALCONN [1], a popular library based on recent theoretical advances in LSH family for cosine similarity [7].

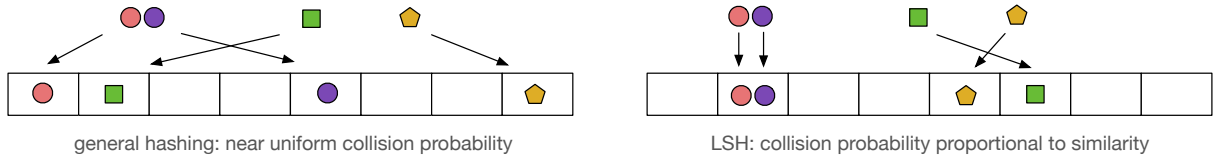

Figure 17: Locality-sensitive hashing hashes similar items to the same hash "bucket" with high probability.

We exclude hash table construction time, and compare single-core query time of FALCONN and our MinHash LSH. We use the cross-polytope LSH family and tune the FALCONN parameters such that the resulting false negative rate is similar to that of the MinHash LSH (6.6%). With "vanilla" LSH, FALCONN achieves an average query time of 0.87ms (85 hash tables); with multi-probe LSH, FALCONN achieves an average query time of 2.4ms (50 hash tables and 120 probes). In comparison, our implementation has an average query time of 36 μs (4 hash functions, 100 hash tables), which is 24$\times$ and 65$\times$ faster than FALCONN with vanilla and multi-probe LSH. We report the runtime and false negative rate under additional FALCONN parameter settings in Table 4. Notably, in multi-probe LSH, adding additional probes reduces the false negative rate with very little runtime overhead. We consider using multi-probe LSH to further reduce the memory usage as a valuable area of future work.

The performance difference reflects a mismatch between our sparse, binary input and FALCONN's target similarity metrics in cosine distance. Our results corroborate previous findings that MinHash outperforms SimHash on binary, sparse input data [61].

A.3 Supervised Methods

In this subsection, we report results from using supervised models for earthquake detection on the Diablo Canyon dataset.

Models. We focus the evaluation on two supervised models: WEASEL [56] and ConvNetQuake [50]. The former is a time series classification model that leverages statistics tests to select discriminative bag-of-pattern features on Fourier transforms; it outperforms the state-of-the-art non-ensemble classifiers in accuracy on the UCR time series benchmark. The latter is a convolutional neural network model with 8 strided convolution layers followed by a fully connected layer; it has successfully detected uncataloged earthquakes in Central Oklahoma.

Data. Same as the qualitative study in Section 8.5, we focus on the area in the vicinity of the Diablo Canyon nuclear power plant in California. We use catalog earthquake events located in the region specified by Figure 15 as ground truth. We perform classification on the continuous ground motion data recorded at station PG.LMD, which has the largest number of high-quality recordings of catalog earthquake signals, and use additional data from station PG.DCD (station that remained active for the longest) for augmentation. Both stations record at 100Hz on 3 channels, capturing ground motion along three directions: EHZ channel for vertical, EHN channel for North-South and EHE channel for East-West motions. We use the vertical channel for WEASEL, and all three channels for ConvNetQuake.

Preprocessing and Augmentation. We extract 15-second long windows from the input data streams, which include windows containing earthquake events (positive examples) as well as windows containing only seismic noise (negative examples). This window length is consistent with that used for fingerprinting.

We adopt the recommended data preprocessing and augmentation procedures for the two models. For WEASEL, we z-normalize each 15-second window of time series by subtracting the mean and dividing by the standard deviation. For ConvNetQuake, we divide the input into monthly streams and preprocess each stream by subtracting the mean and dividing by the absolute peak amplitude; we generate additional earthquake training examples by perturbing existing ones with zero-mean Gaussian noise with a standard deviation of 1.2. For both models, we further augment the earthquake training set with examples of catalog events recorded at an additional station.

In order to prevent the models from overfitting to location of the earthquake event in the time window (e.g. a spike in the center of the window indicates earthquakes), we generate 6 samples for each catalog earthquake event with the location of the earthquake event shifted across the window. Specifically, we divide the 15-second time window into five equal-length regions, and generate one training example from each catalog event with the event located at a random position within each region; we generate an additional example with earthquake event located right in the center of the window. We report prediction accuracy averaged on samples located in each of the five regions for each event. We further analyze the impact of this augmentation in the results section below.

Train/Test Split. We create earthquake (positive) examples from the arrival times from the Northern California Seismic Network (NCSN) catalog [4]. Together, the catalog yields 3585 and 1388 catalog events for PG.LMD and PG.DCD, respectively, from 2007 to 2017. We select a random 10% of the catalog events from PG.LMD as the test set, which includes 306 events from 8 months. We create a second test set containing 449 new earthquake events detected by our pipeline. Both test sets exhibit similar magnitude distribution, with majority of the events centered around magnitude 1. The training set includes the remaining catalog events at PG.LMD, as well as additional catalog events at PG.DCD.

For negative examples, we randomly sample windows of seismic noise located between two catalog events at station PG.LMD. For training, we select 28,067 windows of noise for WEASEL, and 874,896 windows for ConvNetQuake; ConvNetQuake requires a much larger training set to prevent overfitting. For testing, we select 85,060 windows of noise from September, 2016 for both models.

Finally, we generate 15-second non-overlapping windows from one month of continuous data (December, 2011) in the test set. We then select 100 random windows that the model classifies as earthquakes for false positive evaluation.

Results. We report the two models' best classification accuracy on test noise events (true negative rate), catalog events and FAST events in Table 3. The additional training data from PG.DCD boosts the classification accuracy for catalog and FAST events by up to 4.3% and 3.2%. If the model is only trained on samples with the earthquake event in the center of the window, the accuracy further degrades for over 6% for WEASEL and over 20% for ConvNetQuake, indicating that the models are not robust to translation.

Overall, the 20% gap in prediction accuracy between catalog events and FAST events suggests that models trained on the former do not generalized as well to the latter. Since the two test sets have similar magnitude distributions, the difference indicates that

Stages	Fingerprint	Hash Gen	Search	Alignment
Baseline	9.58	4.28	149	>1 mo (est.)
+ occur filter	9.58	4.28	**30.9** (-79%)	**16.02**
+ #n func	9.58	**5.63** (+32%)	**3.35** (-89%)	**18.42** (+15%)
+ locality Min-Max	9.58	**1.58** (-72%)	3.35	18.42
+ MAD sample	**4.98** (-48%)	1.58	3.35	18.42
+ parallel (n=12)	**0.54** (-89%)	**0.14** (-91%)	**0.62** (-81%)	**2.25** (-88%)

Table 5: Factor analysis (runtime in hours, and relative improvement) of each optimization on 1 year of data from station LTZ. Each optimization contributes meaningfully to the speedup of the pipeline, and together, the optimizations enable an over 100× end-to-end speedup.

Sampling Rate	Accuracy (%)	Speedup
0.001	94.9	350×
0.01	98.7	99.8×
0.1	99.5	10.5×
0.5	99.7	2.2×
0.9	99.9	1.1×

Table 6: Speedup and quality of different MAD sampling rate compared to no sampling on 1.3M fingerprints. Sampling enables a 100x speed up in MAD calculation with 98.7% accuracy. Below 1%, runtime improvements suffer from a diminishing return, as the IO begins to dominate the MAD calculation in runtime.

FAST events might be sufficiently different from the existing catalog events in training set that they are not detected effectively.

In addition, we report the false positive rate evaluated on a random sample of 100 windows predicted as earthquakes by each model. The ground truth is obtained via our domain collaborators' manual inspection. WEASEL and ConvNetQuake exhibit a false positive rate of 90% with a 95% confidence interval of 5.88%. In comparison, our end-to-end pipeline has only 8% false positives.

Discussion. The fact that unsupervised method like our pipeline is able to find qualitatively different events than those in the existing catalog suggests that, for the earthquake detection problem, supervised and unsupervised methods are not mutually exclusive, but complementary to each other. In areas with rich historical data, supervised models showed promising potential for earthquake classification [50]. However, in cases where there are not enough events in the area of interest for training, we can still obtain meaningful detections via domain-informed unsupervised methods. In addition, unsupervised methods can serve as a means for label generation to improve the performance of supervised methods.

B. ADDITIONAL EVALUATIONS

This section contains additional evaluation results for the factor analysis in Section 8.1, the microbenchmarks of pipeline parameters in Section 8.3 as well as a figure illustrating the key idea behind locality-sensitive hashing.

In Table 5, we report the runtime and relative improvement of each optimization in the factor analysis in Section 8.1 on 1 year of time series data at station LTZ in the New Zealand dataset.

In Table 6, we report the relative speed up in MAD calculation time as well as the average overlap between the binary fingerprints generated using the sampled MAD and the original MAD as a metric for accuracy. The results illustrate that runtime reduces linearly with sampling rate, as expected. At lower rates, I/O begins to dominate MAD calculation runtime so the runtime improvements suffer from diminishing return.

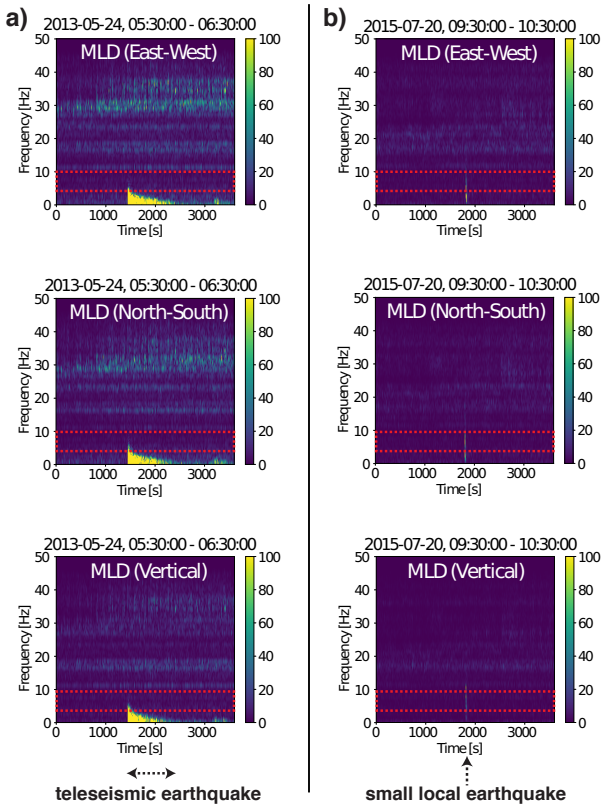

Figure 18: Example hour-long spectrograms from the three components of continuous seismic data, sampled at 100 Hz, at station MLD from the Diablo Canyon, California, data set: East-West (top row), North-South (center row), vertical (bottom row). For this station, a 4-10 Hz bandpass filter (dotted red rectangle) was applied before entering the processing pipeline. (a) Example of signals that should be excluded by the bandpass filter: a magnitude 8.3 teleseismic earthquake from the Sea of Okhotsk (bordered by Japan and Russia) starting at time 1400 seconds, and persistent repeating noise throughout the entire hour at higher frequencies. (b) Example of a signal that should be included in the bandpass filter: a small local earthquake, with magnitude 1.7, at time 1800 seconds.

Finally, Figure 17 illustrates the key difference between LSH and general hashing: LSH hash functions preserve the distance of items in the high dimensional space, such that similar items are mapped to the same "bucket" with high probability.

C. BANDPASS FILTER GUIDELINES

Figure 18 illustrates the process of selecting the bandpass filter on an example data set. The provided examples are hour-long spectrograms computed from the three components of continuous seismic data at station MLD from the Diablo Canyon, California, data set.

Figure 18a shows examples of signals that should be excluded by the bandpass filter. The high-amplitude signal starting at time 1400 seconds is from a magnitude 8.3 teleseismic earthquake near Japan and Russia, with a long duration of over 10 minutes and predominantly lower frequency content (below 4 Hz). Generally, we are not interested in detecting large teleseismic earthquakes, because they are already detected and cataloged by global seismic networks (and shaking is usually felt near their origin). There is

also persistent repeating noise throughout the entire hour at higher frequencies: it is especially prominent at 30-40 Hz on the East-West and North-South channels, but there are several bands of repeating noise, starting at a low of 12 Hz. We commonly observe repeating noise at lower frequencies (0-3 Hz) at most seismic stations, which is also seen in Figure 18a after the teleseismic earthquake. It is essential to exclude as much of this persistent repeating noise from the bandpass filter as possible; otherwise, most of the fingerprints would match each other based on similar noise patterns, degrading both detection performance and runtime.

Figure 18b shows an example of a small (magnitude 1.7) local earthquake signals, at time 1800 seconds, that we would like to detect, and therefore should be included by the bandpass filter. A small local earthquake is much shorter in duration, typically a few seconds long, and has higher frequency content, up to 10-20 Hz, compared to a teleseismic earthquake. We choose the widest possible bandpass filter to keep as much of the desired local earthquake signal as we can, while excluding frequencies with persistent repeating noise.

Figure 18a and b show spectrograms from two different days (2013-05-24 and 2015-07-20) at one example seismic station. In general, we recommend randomly sampling and examining short spectrogram sections throughout the entire duration of available continuous seismic data, and at each seismic station used for detection, as the amplitudes and frequencies of the repeating noise can vary significantly over time and at different stations. Anthropogenic (cultural) noise levels are often higher during the day than at night, and higher during the workweek than on the weekend. Sometimes it is difficult to select a frequency range that does not contain any persistent repeating noise; in this case, we advise excluding frequency bands with the highest amplitudes of repeating noise.

D. HASH SIGNATURE GENERATION

We present pseudocode for the optimized hash signature generation procedure in Algorithm 1.

Algorithm 1 Optimized and parallelized Min-Max hash generation

```
function SINGLE_HASH(d, t, k, seed)                    ▷ Get all hash mappings
    for x ∈ {1, 2, ..., d} do
        for y ∈ {1, 2, ..., t * k} do
            hash[i][j] = MURMURHASH(i, seed + j)
    return hash
function MINMAX_BATCH(fp, hash) ▷ Get hash signature for given batch
    for x ∈ {1, 2, ..., fp.size()} do
        for y ∈ {1, 2, ..., d} do
            if fp[x][y] == 1 then
                for i ∈ {1, 2, ..., t} do
                    for j ∈ {1, 2, ..., ⌈k/2⌉ } do
                        minvals[i][j] = min(hash[y][i][j], minvals[i][j])
                        maxvals[i][j] = max(hash[y][i][j], maxvals[i][j])
        for i ∈ {1, 2, ..., t} do
            minmaxhash[x][i] = HASH_COMBINE(minvals[i], maxvals[i])
    return minmaxhash

function GEN_SIGNATURE(fp, nprocs)                          ▷ main function
    hash = SINGLE_HASH(d, t, k/2, seed)
    fp_partition = PARTITION(fp, nprocs)
    for i ∈ {1,2, ..., nprocs} do in parallel
        MINMAX_BATCH(fp_partition[i], hash)
```