REPORT
doi:10.26443/seismica.v2i2.979

Seismology in the cloud: guidance for the individual researcher

Z. Krauss [*1], **Y. Ni** [2], **S. Henderson** [2,3], **M. Denolle** [2]

[1]School of Oceanography, University of Washington, Seattle, WA, USA, [2]Department of Earth and Space Sciences, University of Washington, Seattle, WA, USA, [3]eScience Institute, University of Washington, Seattle, WA, USA

Author contributions: *Conceptualization*: Z. Krauss, M. Denolle. *Data Curation*: Z. Krauss, Y. Ni. *Formal Analysis*: Z. Krauss, Y. Ni. *Funding Acquisition*: Z. Krauss, Y. Ni, M. Denolle. *Investigation*: Z. Krauss, Y. Ni. *Methodology*: Z. Krauss, Y. Ni, S. Henderson, M. Denolle. *Project Administration*: Z. Krauss, M. Denolle. *Resources*: S. Henderson, M. Denolle. *Software*: Z. Krauss, Y. Ni, S. Henderson. *Supervision*: M. Denolle, S. Henderson. *Validation*: Z. Krauss, Y. Ni, S. Henderson, M. Denolle. *Visualization*: Z. Krauss. *Writing – original draft*: Z. Krauss. *Writing – review & editing*: Z. Krauss, Y. Ni, S. Henderson, M. Denolle.

Abstract The commercial cloud offers on-demand computational resources that could be revolutionary for the seismological community, especially as seismic datasets continue to grow. However, there are few educational examples for cloud use that target individual seismological researchers. Here, we present a reproducible earthquake detection and association workflow that runs on Microsoft Azure. The Python-based workflow runs on continuous time-series data using both template matching and pre-trained machine learning models. We provide tutorials for constructing cloud resources (both storage and computing) through a desktop portal and deploying the code both locally and remotely on the cloud resources. We apply the cloud-based workflow to one year of continuous data from a mid-ocean ridge to demonstrate the construction of two earthquake catalogs, one through template matching and one with a pre-trained machine learning model. We report on scaling of compute times and costs to show that CPU-only processing is generally inexpensive, and can be faster and simpler than using GPUs. Overall, we find that the commercial cloud presents a steep learning curve but is cost-effective. This report is intended as an informative starting point for any researcher considering migrating their own processing to the commercial cloud.

Production Editor:
Gareth Funning
Handling Editor:
Yen Joe Tan
Copy & Layout Editor:
Théa Ragon

Received:
May 22, 2023
Accepted:
August 10, 2023
Published:
August 25, 2023

Glossary for frequently-used terms

Commercial cloud – computational resources that are available for use remotely through a pay-as-you-go system. Cloud providers deliver access to these resources, which are physically maintained in large centers of computing servers, through the internet. Major cloud providers include Microsoft Azure, Amazon Web Services, and Google Cloud Platform.

CPU – *"Central Processing Unit"*. This is the part of a computer that interprets instructions to perform computational tasks. The CPU of a computer typically has multiple "CPU cores", which can each perform one task at a time. In this report, we use the word "CPU" to mean an individual CPU core, such that one CPU can perform one task at a time.

GPU – *"Graphics Processing Unit"*. This is a specialized computing processor that is designed to handle many specific small tasks at once, more efficiently than a CPU. In machine learning, they are commonly used to greatly speed up the training and application of neural networks, which can be applied to earthquake detection.

Parallelization – the act of splitting up a computational task into multiple independent steps, and running these steps on different CPUs or GPUs at the same time, to greatly decrease the overall time needed for computation.

Virtual machine – a resource provided by the commercial cloud that mimics the functionality of a typical computer with chosen amounts of CPUs, GPUs, and memory, but effectively is a barebone operating system which is isolated from the total resources of a larger physical server.

Motivation

Major recent advances in seismological research have been driven by seismic datasets that are dense in both space and time. These include, to name a few, the discovery of slow earthquakes and tectonic tremor (Obara, 2002; Rogers and Dragert, 2003), constraints on the propagation of large earthquakes using back projection (Ishii et al., 2005), and the imaging of shallow Earth properties through the cross-correlation of the ambient seismic field (Shapiro et al., 2005). Recognizing the value of such datasets, community and institutional seismic networks are rapidly increasing the rate and volume of public seismic data. Today, the amount of seismic data available on the IRIS DMC approaches 1 PetaByte, with new technologies able to collect this amount annually (Lindsey et al., 2017). This growth has made seismological research a big-data field that requires methodological advancement and computational infrastructure to maximize discovery (Quinteros et al., 2021).

*Corresponding author: zkrauss@uw.edu

The realization that tectonic events share fundamentally similar physical processes has led seismologists to develop supervised techniques to search the data based on similarity with past events. The technique of template matching, also referred to as matched filter, scans continuous data and detects events using a correlation coefficient with respect to a template (Gibbons and Ringdal, 2006; Turin, 1960). Template matching (TM) is robust at detecting new occurrences of previously seen phenomena and finding events buried in noise (Ross et al., 2019; Shelly et al., 2007). Open-source software to implement TM that makes use of the easily parallelizable nature of the technique is available (Beaucé et al., 2017; Chamberlain et al., 2017), but computational requirements can still be prohibitive when the number of templates is large.

In recent years, seismological research has seen a rapid and massive adoption of statistical and machine learning algorithms in automating seismological research workflows (Mousavi et al., 2020; Perol et al., 2018; Walter et al., 2021; Yoon et al., 2015; Zhu et al., 2023). The focus has largely been on developing and testing workflows on curated earthquake data sets (Michelini et al., 2021; Mousavi et al., 2019; Münchmeyer et al., 2022; Ni et al., 2023). Thus far, only several studies have used machine learning methods to detect new events in continuous seismic records (Tan et al., 2021; Scotto di Uccio et al., 2023). This may be in part because machine-learning models yield inconsistent predictions (Park et al., 2023); however, a more likely barrier to the adoption of machine learning techniques is the high entry cost associated with the computational skills and resources needed to deploy on continuous data.

With the coincident increase in both dataset sizes and the computational cost of state-of-the-art earthquake detection techniques, there is a growing need for seismological workflows to deploy on the cloud (Arrowsmith et al., 2022). Large seismic datasets are well suited for data sharing on cloud object storage, and some institutions such as the USGS and the Southern California Earthquake Data Center have begun migrating raw data and data products to cloud storage permanently (Schovanec et al., 2021; Yu et al., 2021). Despite large archives being on the cloud for a few years, few studies have leveraged them (Clements and Denolle, 2023). Some authors have demonstrated the overall great horizontal scaling performance of cloud computing and developed workflows that stream data using webservices (MacCarthy et al., 2020; Zhu et al., 2023), but the deployment strategies used (e.g., Docker, Dask, Kubernetes) are difficult for researchers to learn and deploy. Most of the seismological community faces a steep learning curve to shifting their functioning local workflows towards cloud computing, limiting its widespread adoption.

This report is distinct from other published seismological cloud-based workflows (Zhu et al., 2023) in that we aim to help the average seismological researcher build their own cloud computing version of their local processing algorithms from the ground up. We demonstrate this through the example of building an earthquake catalog from continuous data. We first develop a workflow to run locally using parallel processing in Python. The workflow applies the two most-used contemporary techniques of supervised earthquake detection: template matching and pre-trained models from machine learning, including earthquake detection, phase picking, and association. Then, we describe what is needed to migrate this workflow to the cloud, including constructing cloud storage, code containers, and cloud computing pools. We use Microsoft Azure because of resources available to us through our home institution, but we report that the core framework is similar to other major cloud providers, provided that researchers adapt for provider-specific storage and compute systems. We describe the workflow in detail and provide Jupyter notebooks and instructional materials through a GitHub page (Krauss et al., 2023a). We attempt to follow the guidelines of FAIR4RS (Barker et al., 2022; Wilkinson et al., 2016), which recommends that published software and metadata be human and machine findable, accessible via GitHub, interoperable, and usable & reusable.

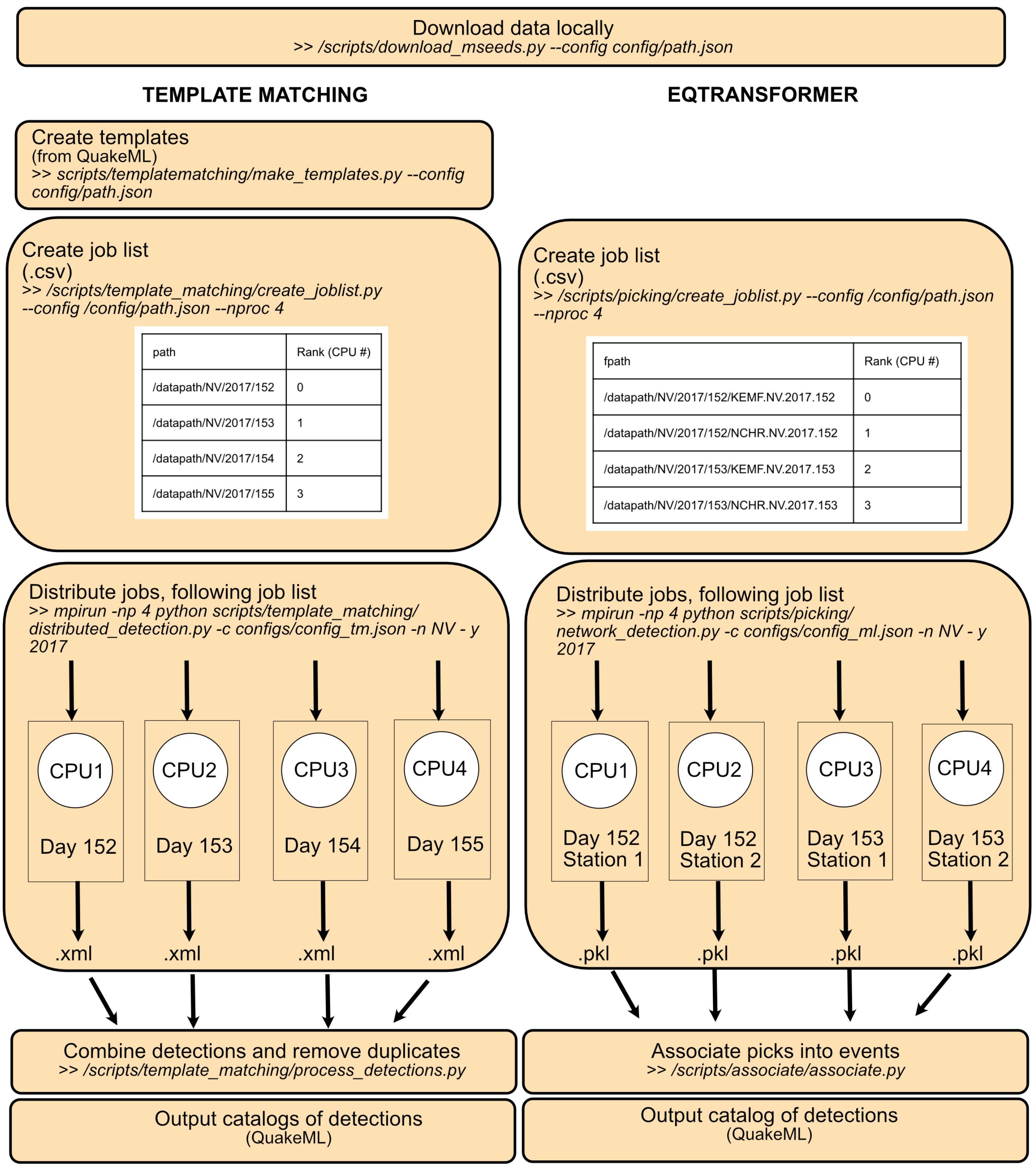
We provide cost and timing context for what users may expect for typical seismic workflows by documenting scaling performance, including a comparison between strategies that use either CPU or GPU computing (see Glossary). We also present results of the earthquake catalog workflow on the tectonically active Endeavour segment of the Juan de Fuca mid-ocean ridge, which act as a proof-of-concept of how template matching and machine learning techniques can be used together construct a detailed earthquake catalog.

Local Workflow

First, we describe how to build the workflow for a local implementation: this represents the first-step case of most individual researchers. We design a workflow to have two separate but operationally equivalent "branches" for the two earthquake detection methods of interest (Figure 1). The format and file structure of the input seismic waveform data is the same for both branches. Both branches produce earthquake detections in the same output format, QuakeML, an XML representation of earthquake metadata widely used by the seismic community (Schorlemmer et al., 2011).

TM is performed using EQcorrscan, an open-source Python toolbox for earthquake detection via the cross-correlation of waveform data with earthquake templates (Chamberlain et al., 2017). Detection parameters including filter bands, template lengths, and the type and magnitude of the detection threshold are specified in the config file as described in Section 2.1. After TM detection, redundant events are removed between templates by identifying events that occur within a given time threshold of each other (e.g., 1 s), and keeping only the event with the highest detection value.

For a comparison to machine learning-based earthquake detection methods, the other branch uses SeisBench, an open-source flexible Python framework for deploying seismological machine learning models (Woollam et al., 2022). Our example workflow uses


Download data locally
TEMPLATE MATCHING
EQTRANSFORMER
Create templates (from QuakeML)
Create job list (.csv)
Distribute jobs, following job list
CPU1
CPU2
CPU3
CPU4
Day 152
Day 153
Day 154
Day 155
Day 152 Station 1
Day 152 Station 2
Day 153 Station 1
Day 153 Station 2
.xml
.pkl
Combine detections and remove duplicates
Associate picks into events
Output catalogs of detections (QuakeML)
Output catalog of detections (QuakeML)

```
>> /scripts/download_mseeds.py --config config/path.json
```

```
>> scripts/templatematching/make_templates.py --config config/path.json
```

```
>> /scripts/template_matching/create_joblist.py --config /config/path.json --nproc 4
```

path	Rank (CPU #)
/datapath/NV/2017/152	0
/datapath/NV/2017/153	1
/datapath/NV/2017/154	2
/datapath/NV/2017/155	3

```
>> /scripts/picking/create_joblist.py --config /config/path.json --nproc 4
```

fpath	Rank (CPU #)
/datapath/NV/2017/152/KEMF.NV.2017.152	0
/datapath/NV/2017/152/NCHR.NV.2017.152	1
/datapath/NV/2017/153/KEMF.NV.2017.153	2
/datapath/NV/2017/153/NCHR.NV.2017.153	3

```
>> mpirun -np 4 python scripts/template_matching/distributed_detection.py -c configs/config_tm.json -n NV - y 2017
```

```
>> mpirun -np 4 python scripts/picking/network_detection.py -c configs/config_ml.json -n NV - y 2017
```

```
>> /scripts/template_matching/process_detections.py
```

```
>> /scripts/associate/associate.py
```

Figure 1 Flowchart of our workflow for performing earthquake detection on seismic waveform data, showing the two branches, TM and EQT, side by side. Each script's name, data format, filename extension, and unix commands are described for transparency and reusability.

EQTransformer (Mousavi et al., 2020) for phase picking and GaMMA (Zhu et al., 2022) for phase association; however, employing a different machine learning model would be straightforward through the flexibility of SeisBench. We do not perform model training, but only model inference: we use the pre-trained EQTransformer model provided by SeisBench, which is identical to the original model weights from Mousavi et al. (2020), to make P- and S-wave picks on all stations. To sweep continuously through the data, we use a sliding window of length 60 s with a step of 30 s and a detection threshold of 0.1 for both P- and S-waves. We then perform phase association using GaMMA, assuming 7.0 and 4.0 km/s as constant velocity for P- and S-waves, respectively, and requiring only one station with picks to form an event. These specifications are made in the config file required for the code execution.

Local set-up

We retrieve seismic waveform data from the IRIS DMC (Trabant et al., 2012) for a given time period using the ObsPy Python toolbox (Beyreuther et al., 2010). Three-component waveform data are resampled to a common sampling rate and stored in day-long chunks for all channels from a single station in the same way they are stored natively within the DMC. We organize these data in a descending folder structure first by network, then year, then day of the year, with individual MiniSEED files saved for each station. The naming of these files is important to parallelization later in the workflow. The data download can be performed using the `/scripts/download_mseeds.py` script (Figure 1), with date limits, network name, station and channel codes, and the output data directory specified within the config file (Notebook S1).

The TM branch requires an input of earthquake templates as an EQcorrscan Tribe object, stored in TAR format. An example script that constructs these templates from waveform data, using a starting earthquake catalog in QuakeML format, can be found at `/scripts/template_matching/make_templates.py`.

Finally, to ensure the portability of the code across machines, we specify all file paths, detection parameters, and machine characteristics in JSON format in a config file in the `/configs/` folder. Notebook S1 details the construction of these files. The config files are referenced in the call to the scripts that execute earthquake detection (Figure 1).

Single-node parallelization

A single computer, or node, typically has several CPU cores (hereafter referred to simply as CPUs) and is the smallest level of parallelization. Earthquake detection can be easily parallelized such that the computation is performed simultaneously for different time periods on different CPUs. For the EQT branch, which performs phase picking on one station at a time, we consider one day of earthquake detection on one station to constitute a single job. For the TM branch, which jointly performs phase picking and association on data from a complete network of stations, we consider one day of earthquake detection on all stations to constitute a single job.

When deploying our workflow locally, we organize the distribution of jobs across available CPUs by first creating a job list in CSV format (Figure 1). The job list is a simple table that ties the file path of MiniSEED data to a CPU number (Figure 1). The script to create this job list, `create_joblist.py`, is different for the TM and EQT branches, and can be found in the `/scripts/template_matching/` and `/scripts/picking/` directories, respectively. The job list CSV file can be created by running the script from the command line with additional arguments that specify how many CPUs to parallelize jobs across, and a path to the JSON config file which contains date limits and the path to the waveform data (Figure 1, Notebook S2).

We then distribute the jobs across CPUs in parallel using Open MPI (O-MPI), an open-source message passing interface (Gabriel et al., 2004). When O-MPI is called at the start of a Unix-style command, the command is simultaneously deployed separately to as many CPUs as specified. In our local workflow, we use O-MPI to run a distributing script `distributed_detection.py` (Figure 1, Notebook S2). As the distributing script runs on each individual CPU, it reads in the created job list and filters the job list to include only those assigned to the current CPU. The distributing script then loops over the filtered list and completes the job by running the script in which the actual detection is performed, `detection.py`, on the specified data path. This operation is the same for both the template matching and machine-learning workflows, with the template matching workflow parallelized over days only and the machine-learning workflow parallelized over both days and stations.

The TM branch outputs a catalog of detected earthquakes with P- and S-wave picks for each day of detection in QuakeML format. These daily catalogs can be converted to a summative earthquake catalog, also in QuakeML format, using `/scripts/template_matching/combine_catalogs.py`, which collates and removes duplicate detections between templates (Notebook S3). The EQT branch outputs a list of P- and S-wave picks for each day of detection in Python Pickle format. These picks can be associated into individual earthquakes using `/scripts/association/associate.py`, which produces an equivalent summative earthquake catalog in QuakeML format (Notebook S3).

Cloud Workflow

The value of the commercial cloud lies in the ability of an individual to pay for the use of computational instances of flexible size at any time in a “pay-as-you-go” structure. However, these virtual machines (VMs) are effectively a blank computer with a user-specified configuration. The configuration of a VM consists of user-specified CPU cores, RAM, and local storage. VMs can be chosen with a range of Operating Systems and environments. We recommend choosing a blank environment and installing only the necessary dependencies.

Using a VM in a way that mimics local workflows requires an ecosystem of cloud resources: a storage container that the VM can read from and write to, a packaged software environment with all desired scripts and their dependencies, and an overarching set of networking permissions that allows all resources to work in tandem (Figure 2). In this section, we describe the construction of an example set of these resources on Microsoft Azure, point to additional materials that further detail how to construct them, and describe how to use the constructed resources to process seismic data in parallel.

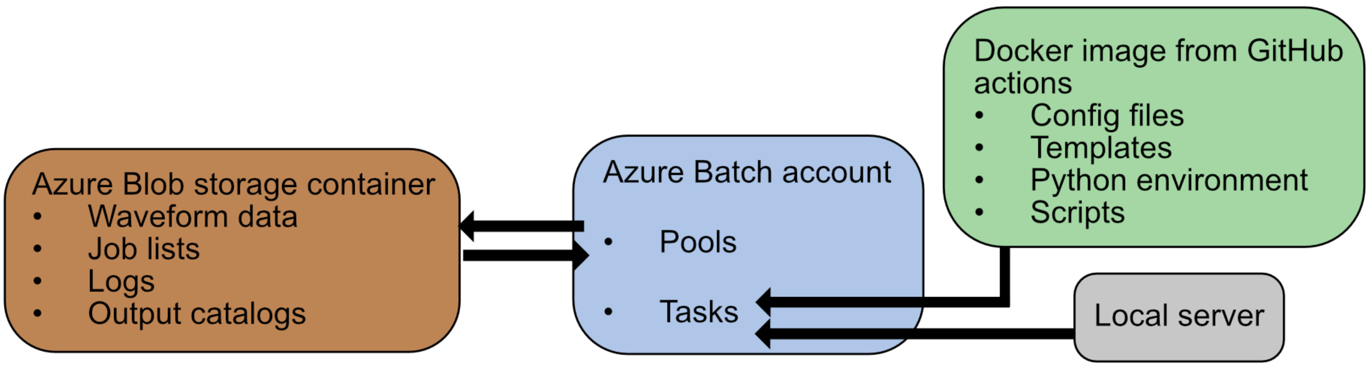


Figure 2 Map of how cloud resources relate to processes run on the local server and the containerized code.

Resource set-up

Containerizing the code base

In order to transition our local research workflow from our computer to the cloud, we first need to encapsulate all dependencies needed for the scripts to successfully execute. Containers are a standard way to define a minimal reproducible computing environment, such that you can guarantee your analysis will run on any other computer. There are various software systems to define and build containers. Docker is a widely-used system of software tools to create, store and execute code as containers. The container image is defined by a Dockerfile, which specifies the operating system software dependencies and command to execute. Images are typically stored on a server that can be accessed over the internet so that any computer can "pull" the image and start the container that executes your code. This allows for a high degree of parallelism because there can be thousands of containers each executing on different cloud-hosted virtual machines.

Because our code is hosted in a GitHub repository named "seismicloud" (https://github.com/Denolle-Lab/seismicloud, Krauss et al., 2023a), we use GitHub Actions continuous integration to build the container. Every time the repository is updated, the script docker.yml rebuilds a container image based on the Dockerfile in the repository. Specifically, the container image installs all Python dependencies (ObsPy, EqCorrscan, etc.) in a basic Linux operating system and copies the current version of the seismicloud codebase, including all scripts and data files. The container image is stored on the publicly-accessible GitHub Container registry. Any machine can create a container of the seismicloud workflow by accessing the URL and corresponding GitHub commit (e.g. `docker run ghcr.io/denolle-lab/seismicloud:latest`).

Initiating an Azure cloud account

Creating a user account on Azure is free, but a form of payment, or "subscription", must be tied to the account in order to create resources. For NSF-funded projects with provisions for cloud computing, this is sometimes done through a supported service, CloudBank (Norman et al., 2021). Once a subscription has been set up and users gain access to the Azure portal, users should begin by creating a budget for their subscription with automatic alerts for when spending has reached a given percent of the allocated funding (Tutorial S1). The next step is to create a "resource group", which will be used to tie the cloud resources users create to the subscription. Finally, a "virtual network" is created, which is a set of permissions that allows you to access the created resources and also allows those resources to access each other. Multiple users on the same subscription who will be carrying out separate processing should create their own separate resource groups and virtual networks. All of these actions are performed on the Azure portal, which is accessed through a web browser (Tutorial S1). We note that we built all of our cloud resources exclusively in the West-US 2 region, which was closest to our local servers in Seattle, Washington.

Storage container

The commercial cloud also provides opportunities for data storage that follow pay-as-you-go pricing. Cloud storage tends to use the model of object storage, which is designed to sustain high frequency data query. In Microsoft Azure, this is called Azure Blob storage. Blob storage containers are easily read from and written to by virtual machines so long as the storage container and virtual machine are under the same virtual network and necessary permissions are specified (see Tutorial S2). The cost of the container is dependent on how much data is actively stored and how frequently data is transferred into or out of the container. For Azure blob storage at the time of writing, the cost to store 1 TB of data is approximately $150 USD per month. Given these costs, typical individual researchers may not want to use Blob as a backup hard drive, but instead use it as temporary storage during computation.

For our workflow, we created one Azure Blob storage container to store raw waveform data and the outputs of processing, including earthquake catalogs, job lists, and processing logs. This choice was driven by our desire to centralize storage for parallelized computation. Tutorial S2 details the creation of the Blob storage container through the Azure portal, including the settings needed to facilitate mounting of the container to virtual machines later in the workflow. To load waveform data to the Blob storage container, we downloaded it on our local servers and then copied it to the storage container using Azure's command line utility AzCopy (see Tutorial S2). Waveform data could also be down-

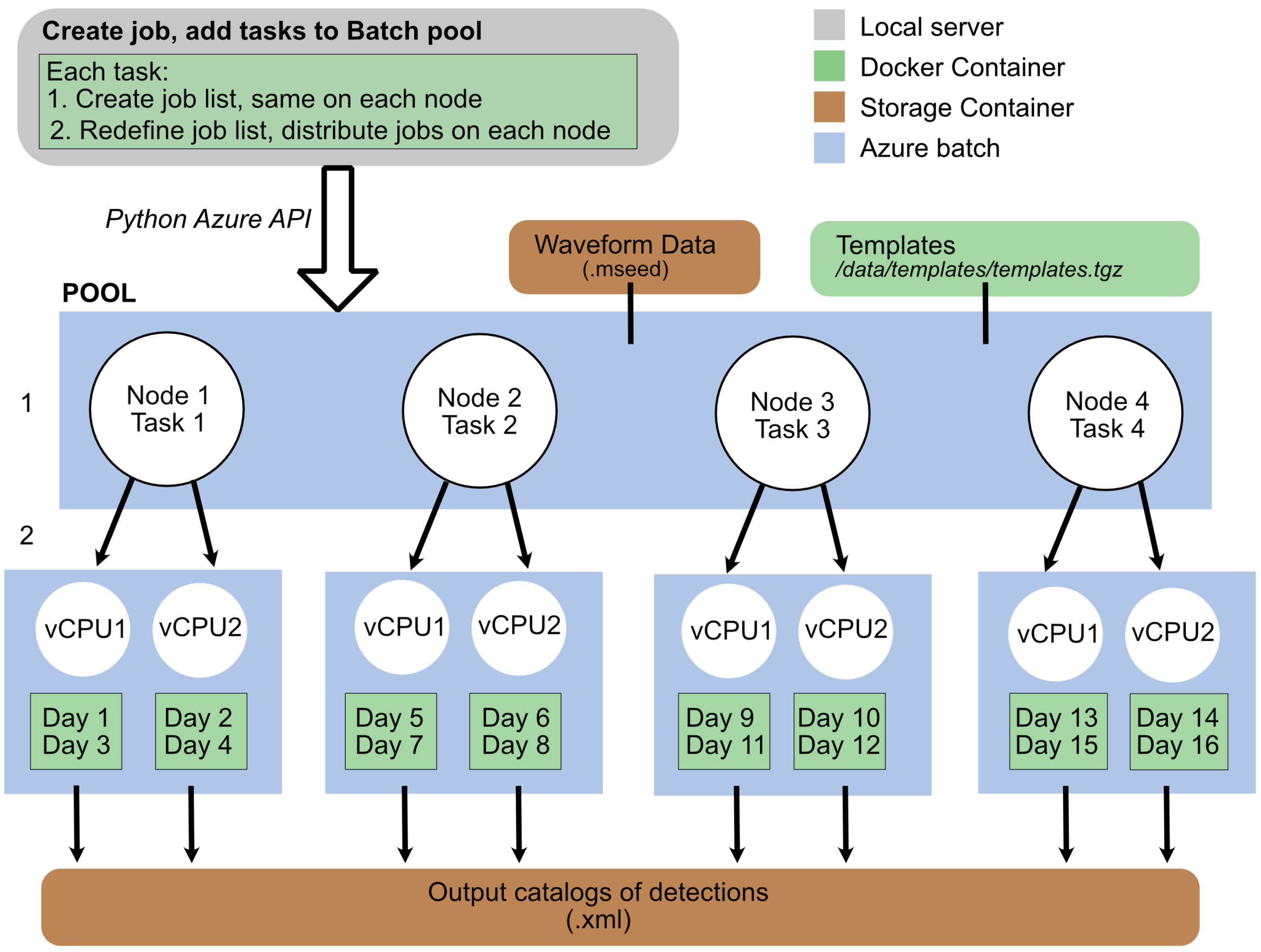


Figure 3 Example flowchart of how Azure cloud Pool-based Parallelization works for the Template Matching workflow, following the color scheme of Figure 2. The EQTransformer workflow is identical, except that the data paths are specific to both the day of the year and the seismic station.

loaded from webservices and stored directly on the storage container by running the `download_waveforms.py` script within the Docker container on a virtual machine with the Blob storage container mounted.

Pool-based parallelization

A powerful option offered by the commercial cloud is the ability to deploy tasks to not just one virtual machine, but pools that can be scaled to include many virtual machines, all managed from one account. In Microsoft Azure, this is performed through a resource called Batch. Batch accounts manage two separate entities: (1) Pools, groups of virtual machines (each called a node when within a Pool), each of which has an identical size and computing environment, and (2) Jobs, sets of commands that are passed to nodes within a Pool, which run with specified settings: inside of a Docker image, for example. Creating a Pool that runs correctly with the user's chosen Docker image and is connected to the user's desired Blob storage container is complicated, but once the Pool has been created, its specifications are saved and you can resize it as needed to run operations on-demand.

Users begin by creating a Batch account which will manage both Pools and Jobs through the Azure portal (see Tutorial S3). A Pool can then be created with a chosen node/virtual machine size, number of nodes, and region. Tutorial S3 details this process. To ensure that the nodes can execute commands within our Docker image, the operating system of the Pool must be specified as Docker compatible. We mount the Blob storage container to each node on the Pool through a start-up command line task. After a Pool has been created, the status of the nodes can be monitored from the Azure portal within the Batch account. The Pool can be resized at any time to contain anywhere from 0 up to the user's given quota of nodes, with all nodes created in the same manner specified when the Pool was created. If quotas, or limits imposed by Azure on how many CPUs can be given to one user in a given region, restrict the Pool the user wants to build, they can be increased through support requests (Tutorial S3). We note that starting quotas for a new user are often quite low, e.g. only 4 vCPUs in a given region.

Once a Pool has been created and the nodes on the Pool have successfully mounted the Blob storage container, we create and send Jobs to the Pool from a local server (e.g., a laptop) using Azure Python libraries.

Notebook S4 details this process. We use the Azure Python libraries to connect to both the Blob storage container and the Batch account using the account names and keys. We then create an empty Job on the Batch account that is tied to the already-created Pool. Next, we decide how many "Tasks" we need to send to the Job based on the number of nodes within the pool, N. Tasks are a single unit of computation, or one command line sent to one node. If all nodes are in use, the Job will keep Tasks in a queue until one becomes free. In our case, we create N Tasks such that the Tasks and nodes are matched 1:1 and there is no queuing. Each Task contains two command line operations: (1) creating a job list in CSV format following Figure 1, and (2) distributing the jobs across CPUs on the node using O-MPI. These commands contain an argument that indicates to the node which index, 1-N, they correspond to. During the creation of each Task, we specify that the commands are run within our Docker image; each node runs a separate Docker container, such that the first task run on each node needs to additionally pull down the container image.

The scripts called in the command lines are found in the `/batch_scripts/` directory of the Docker image. These Batch-specific scripts are very similar to the scripts called during local parallelization and are named in the same way (Figure 1), but are slightly modified to accommodate two levels of parallelization: across nodes, and across CPUs on each node (Figure 3). We avoid the need for inter-node communication by first creating a CSV job list that assigns each job to a node in the Pool rather than an individual CPU, such that each node creates the exact same job list. After the initial job list is created, however, each node then filters the job list following the number, 1-N, of the node, which is specified in the creation of the Task. The filtered, shortened job list is then redefined such that each job on the list is assigned a CPU number instead. The process then runs the same way as the single-instance parallelization, where the jobs on the job list are distributed across CPUs on the nodes using O-MPI (Figure 3). The outputs from all nodes, including job lists, logs, and catalog/pick outputs, are written to the mounted Blob storage container (Figure 3). These outputs can be downloaded locally also using Azure's command line utility AzCopy, as detailed in Tutorial S2.

We choose to use Batch services rather than a workload manager such as SLURM (Yoo et al., 2003), as is typically used in high performance computing (HPC), for several reasons. To use such a system on the cloud would require running a persistent machine to handle orchestrations, which would add cost and complexity. With Batch services, orchestration is instead done by the cloud provider at the time of job submission. Batch services also tend to be focused on independent containerized workflows whereas HPC scheduling systems are designed for non-containerized workflows on closely networked hardware.

Computational Performance

To understand how compute time and costs scale with different Pool sizes and types, we ran both the TM and EQT branches on one year of raw waveform data, ~600 GB in our case. We used data from 2017 for the Ocean Networks Canada NEPTUNE array (network code NV), a cabled 4-station ocean bottom seismometer network on a mid-ocean ridge (Heesemann et al., 2014). These data are locally available from the IRIS DMC. We performed limited preprocessing, only resampling all streams to a common sampling rate of 200 Hz. For the TM branch, we ran detection with a set of 53 templates chosen as a representative sample across the months of 2017 following Krauss and Wilcock (2022).

We constructed two separate Azure Batch Pools for the TM and EQT branches following the steps outlined in Tutorial S3. For the TM Pool, we used a memory-optimized instance type, the `standard_e4_v3`, to accommodate the intensive memory needs of cross-correlation. We found that we needed a minimum memory size of 32 GB, which is paired with 4 CPUs in Azure's virtual machine options, to avoid memory errors from EQcorrscan. For the EQT Pool, we used a compute-optimized instance type, the `standard_f4s`, also with 4 CPUs. The price of equivalent type but larger (more CPUs) virtual machines increases linearly. This is an advantage of using the cloud: it is the same cost to run 60 equivalent machines for one minute as it is to run 1 machine for 60 minutes.

We ran earthquake detection for the entire year of 2017 for both the TM and EQT branches on their corresponding Pools with increasing Pool sizes, from 1-64 nodes, or 4-256 CPUs, and documented compute time and associated costs (Figure 4). The compute times reported in Figure 4 include only the time needed to pull the image once and then run the tasks described in Figure 3, and do not include the additional time needed for waveform download, template construction, Pool start-up, or pick and catalog post-processing. Pool start-up times normally do not exceed 10 minutes, though this varies based on current user traffic in the region. Since start-up processes are run in parallel across nodes on the Pool, overhead start-up times do not tend to increase with number of CPUs or GPUs. The times shown are the mean of the compute time of all Tasks sent to the Pool, which vary slightly due to detections per day and data gaps. The costs associated with each earthquake detection run were calculated by multiplying compute time and number of nodes by Azure's per-hour pricing of the corresponding virtual machine type at the time of writing.

We find that the compute times and costs of both TM and EQT detection scale similarly with numbers of CPUs, with TM detection taking on average 38% of the time needed for EQT detection (Figure 4a). They notably do not scale as one divided by the number of cores despite our distributed memory parallelization scheme, because we do not use inter-node communication. We do not parallelize across all CPUs in the Pool, but only across groups of CPUs. So, if one day of detection takes longer on Node 4 than on Node 5, the next detection

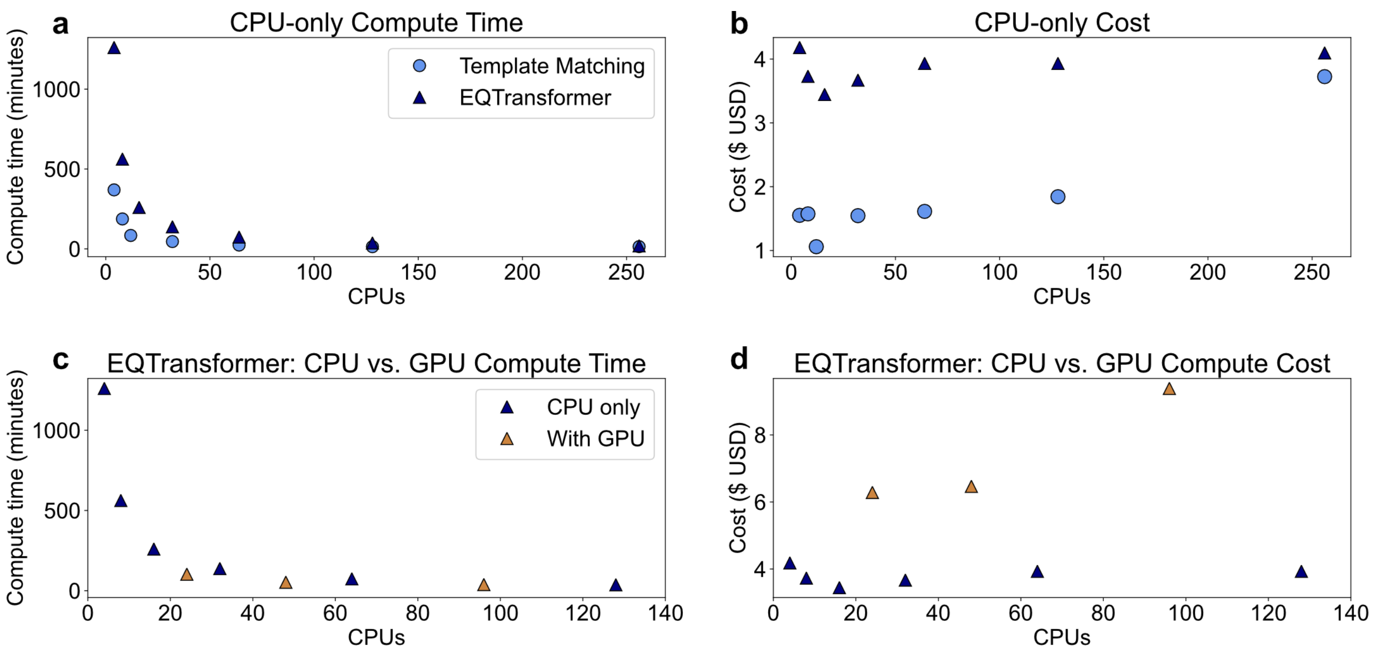


Figure 4 Scaling relationships between compute time, cost, and number of CPUs used for both the Template Matching and EQT workflows.

slated for Node 4 cannot be moved to Node 5 even if Node 5 is idle.

For all Pool sizes tested, we find that the cost of computation is less than $5 USD (Figure 4b). Both the TM and EQT tests show that the minimum cost is associated with a Pool size of 4 nodes, or 16 CPUs, with costs then increasing with increasing Pool size. The cost for the TM tests are lower than the EQT tests due to lower compute times; the virtual machine type used for the TM Pool, `standard_e4_v3`, was slightly more expensive than that used for the EQT pool, the `standard_f4s`, at $0.25 USD/hour and $0.19 USD/hour, respectively.

Constructing our workflow to run only with CPUs and not GPUs was a decision we made to simplify the setup of our Pools and to minimize the size of our Docker container. By not requiring the CUDA libraries for our Docker container, we reduced the size of the image from 6 GB to 1.4 GB. We also avoided the need for additional complexity in the parallelization method, such that we did not need to distribute tasks across both CPUs and GPUs and manage the communication between them. However, for methods that use deep learning networks such as EQT, GPUs are typically employed during model training to significantly accelerate computation time.

In order to investigate how the speed-up offered by GPUs compares to the lower cost of CPU-only instances, we ran three tests of the EQT branch on an available local server that had 4 A100 GPUs with 80 GB of RAM each. Microsoft Azure has an equivalent virtual machine size, the NC24ads, NC48ads, and NC96ads, with 24, 48, and 96 vCPUs, and 1, 2, and 4 A100 80 GB GPUs, respectively. We evenly distributed the jobs among the GPUs. We recorded the compute time for the 2017 data for three separate tests running locally, using (1) 24 CPUs and 1 A100 GPU, (2) 48 CPUs and 2 A100 GPUs, and (3) 96 CPUs and 4 A100 GPUs (Figure 4c). Using the timing information and pricing information from Azure, we calculated the equivalent cost to run on the cloud using the same computing set-up (Figure 4d). The cost of running on GPU instances is > 3x that of CPU-only instances, with the F4s instance (24 vCPUs) costing $1.19/hour in the West-US 2 region and the NC instance (24 vCPUs) costing $3.80/hour.

For our use case, we find that the same computational speed-up from GPUs can be achieved with only CPUs for a fraction of the cost. For instance, running the EQT branch on one year of data using 128 CPUs from the F4s instance had a compute time of 37 minutes, while the same test ran using 96 CPUs and 4 A100 GPUs took 38 minutes (Figure 4c). These two tests had associated costs of $3.93 USD and $9.38 USD, respectively. While the cost of both of these tests is relatively inexpensive, we point out that the speed-up offered by GPUs is marginal in comparison to the extra time and effort needed to address the complications of parallelizing across GPUs and to increase the size and complexity of the Docker container. It should also be noted that these results are application dependent: we do not attempt to parallelize the TM branch using GPUs with the capabilities of the Fast Matched Filter method (Beaucé et al., 2017) in order to avoid adding CUDA dependencies to the containers.

Example Results

We report results of the TM and EQT detection workflows applied to one year of seismic waveform data. We do not intend for these results to represent a thorough comparison of the two methods because we did not iterate on thresholding parameters for either the TM or EQT branches of detection. This section instead serves as a demonstration of the results of our described workflow.

We apply the TM and EQT detection workflows to waveform data from 2017 for the NV network, a cabled

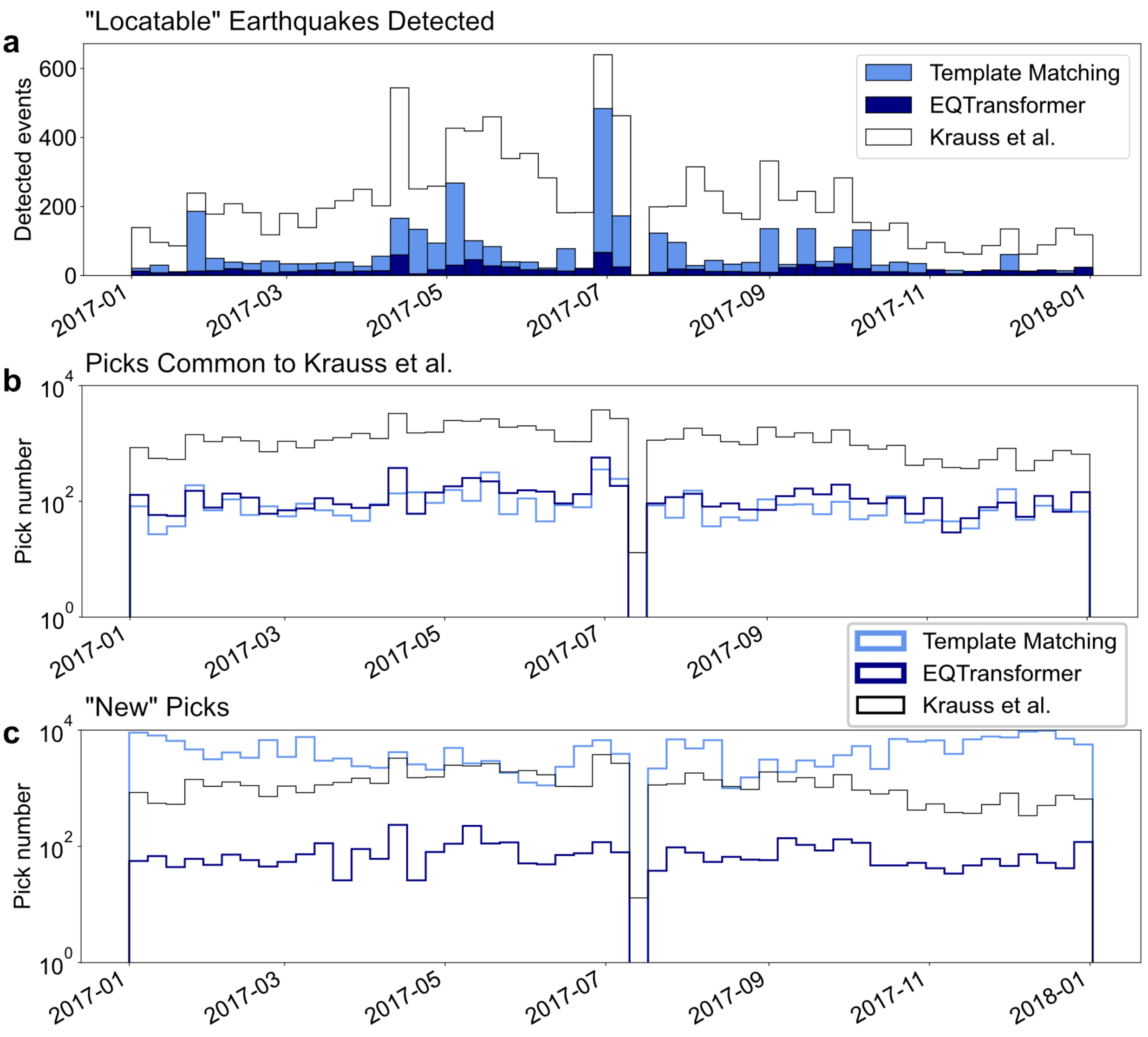


Figure 5 Results of earthquake detection on the NV network for the year of 2017. **(a)** Histogram of earthquakes detected using either TM or EQT in comparison to the catalog of Krauss et al. (2023b), with bin widths of 1 week. The plotted TM events all have absolute cross correlation sums of greater than 3.2, and the plotted EQT events all have at least 6 picks included in the associated event. **(b)** Histogram of how many overall picks were common to the Krauss et al. (2023b) catalog for either TM or EQT at a threshold of 0.5 s pick difference, with bin widths of 1 week. Note that the y-axis is shown in log-scale to improve visual comparison. The data gap in 2017-08 was due to a network outage.

ocean bottom seismometer network that sits within the hydrothermal vent fields of the Endeavour segment on the Juan de Fuca ridge (Heesemann et al., 2014). This area typically experiences at least 10 shallow small earthquakes (Mw < 2.5) per day and frequent swarms. Most recorded earthquakes are located within 40 km of the network. For ground truth comparison of the results of TM and EQT detection, we use the catalog of Krauss et al. (2023b), which was created using traditional STA/LTA methods and has a magnitude of completeness of Mw ~0.5 for earthquakes within the network.

For the TM detection, we use a set of 53 templates that were chosen as representative of the most frequently-occurring families of earthquakes with high waveform similarity during 2017 following Krauss and Wilcock (2022). This is far fewer than the total number of earthquakes (> 11,000) in the Krauss et al. (2023b) catalog for 2017 (Figure 5a), such that we do not expect the TM detection to be able to fully reconstruct the entire catalog. We present the TM detections that have a summed absolute cross correlation across the eight template channels greater than 3.2 (Figure 5), equivalent to a median absolute deviation threshold near 8. For EQT detection, we use the EQT model pretrained on STEAD and required both P- and S-wave thresholds of 0.1, similar to (Jiang et al., 2022; Scotto di Uccio et al., 2023). For EQT, we include earthquakes that contain at least 6 picks after association through GaMMa (Figure 5).

In contrast to the 11,000 located earthquakes in the Krauss et al. (2023b) catalog, the TM and EQT workflows find 3,543 and 924 earthquakes, respectively (Figure 5a). Figure 5 compares the total number of picks made by EQT and TM directly to the Krauss et al. (2023b) catalog, separately classifying those that are common to the catalog (Figure 5b) and those that are "new" picks (Figure 5c). We classify a TM or EQT pick as common to the Krauss et al. (2023b) catalog if it has the same station, phase, and occurs within 0.5 s of a pick in the Krauss et al. (2023b) catalog.

Notably, TM and EQT find comparable numbers of picks that are common to the Krauss et al. (2023b) catalog (Figure 5b), even though TM decisively finds more earthquakes overall (Figure 5a). The picks found by the TM method are mostly “new” picks: only 2% of the TM picks were also in the Krauss et al. (2023b) catalog. This suggests that the picks found by TM are almost entirely just smaller or noisier examples of the template events that went undetected by traditional methods. In contrast, 62% of the picks found by the EQT branch are also in the Krauss et al. (2023b) catalog. Therefore, although EQT finds overall less earthquakes than TM, it likely captures a more complete representation of waveform diversity in the dataset.

These results support the complementary use of template matching and pre-trained machine learning models in constructing earthquake catalogs. Template matching is typically used as a post-processing step to expand catalog completeness after an initial catalog is constructed with machine learning (Shi et al., 2022; Zhang et al., 2022; Zhou et al., 2021; Scotto di Uccio et al., 2023). Our example results, which show that EQT captures a wider range of waveforms while TM detects smaller versions of similar waveforms, support the use of EQT to create a starting catalog and then TM to expand the EQT catalog.

Discussion

We have documented that running seismic workflows on commercial cloud providers can be relatively inexpensive. In our case, costs are < $5 to process one year of continuous data for a small network (Figure 4). This knowledge could significantly expand the ability of the seismic community to perform research at scale: low-cost cloud resources offer a valuable alternative to researchers who do not have institutional access to HPC resources. The pay-as-you-go nature of commercial cloud resources also means that researchers can experiment with different types and sizes of computing resources before committing to the large start-up costs of building a local cluster. This can also mean that fewer machines are plugged in and unused.

However, it is important to reemphasize that working in the cloud can be difficult and unintuitive. Our functioning cloud resource system was only accomplished after months of effort and occasional consultation with data scientists and cloud experts. Much of the workflow we report here relied on skills developed during a week-long “hackathon” with one-on-one help and devoted resources. Our experience suggests that researchers will need access to cloud expertise through their academic institutions to realize the true potential of cloud computing, although we hope the documentation provided here can alleviate some of that need.

Another informative result of our cyberinfrastructure investigation is that the deployment of pre-trained machine-learning workflows on CPU-only set-ups can be as fast as and much easier than a GPU implementation, for the same or lower cost (Figure 4). We attribute this to the large size of the CUDA package in the container and the high speed of prediction using these models on CPU. Traditionally, it is assumed that GPU are better for machine learning workflows, but we suggest that this statement is more strongly true for training these models. In contrast, Yu et al. (2023) report that GPU use for prediction, not training, offers a significant speed up in comparison to CPU-only computation. However, we point out that their timing benchmarks are made using a production-level implementation that directly applies pre-trained machine learning models to discrete non-continuous waveforms. Alternatively, our method uses SeisBench to pre-process waveforms and manage the overlapping of input continuous data prior to application of the machine learning model. Starting from continuous data is very common for many research workflows, and in such cases we have shown that using CPUs can be more cost-effective.

To strengthen the transfer of local workflows to the cloud beyond what we have demonstrated here, researchers could create storage and compute instances through code, referred to as “infrastructure as code”, rather than through desktop portals (e.g., Morris, 2020). We found that this was not feasible with the current permissions settings required with our chosen types of resources, Azure Blob storage and Batch Pools, but we encourage researchers intending to set up large cloud systems that will run long-term to pursue non-portal workflows. Alternatively, point-and-click methods through desktop portals as we have shown here can be an easily understandable way for beginners to get started with cloud resources. It would also be possible for researchers to emulate this workflow on other cloud platforms such as Amazon Web Services or Google Cloud Platform; the codes to interact with the cloud resources (Azure Blob storage, Azure Batch) would need to be adapted to the interface specific to the cloud provider, but no significant changes would need to be made to the code base itself.

Conclusion

The workflow we present provides a basis for individual researchers to adapt their local seismic processing work to the commercial cloud. We have documented how to containerize code repositories using Docker, how to construct and access storage in the Azure cloud, and how to combine these resources to construct and access computing resources in the Azure cloud. For researchers unfamiliar with parallelization techniques in general, we also provide examples for the parallelization of seismic workflows on local machines. The learning curve associated with cloud set-up is steep. But, the results of our scaling tests show that seismic processing in the cloud is both cheap and fast. Since the low cost of cloud computing makes large-scale processing more accessible to the seismic community, the migration of local workflows to the cloud is a worthy endeavor. We hope this work can serve as a useful starting point for other researchers.

Acknowledgements

We thank Naomi Alterman, Jesse Biehl, and Charles Rice for their invaluable knowledge of and assistance with the Azure cloud platform. We also thank two anonymous reviewers for their thoughtful comments which greatly improved the manuscript. Funding was provided by the NSF SCOPED award (EAR-2103701), NSF CyberInfrastructure Award (EAR-2117834), by the University of Washington eScience Institute's inaugural Cloud ReproHackweek, and by a National Defense Science and Engineering Graduate Fellowship to ZK.

Data and code availability

The facilities of IRIS Data Services, and specifically the IRIS Data Management Center, were used for access to waveforms used in this study. IRIS Data Services are funded through the Seismological Facilities for the Advancement of Geoscience (SAGE) Award of the National Science Foundation under Cooperative Support Agreement EAR-1851048. The earthquake catalog for the Endeavour segment that is presented for comparison in this study is available through the Marine Geoscience Data System via 10.26022/IEDA/330498 (Krauss and Wilcock, 2021). The codes referenced in this study, including the Jupyter Notebook tutorials named Notebook S1-S4, are available at https://github.com/Denolle-Lab/seismicloud, stored on Zenodo (Krauss et al., 2023a). Tutorials S1-S3, which are in PDF format, are available as supplementary materials and also in the seismicloud GitHub repository.

References

Arrowsmith, S., Trugman, D., MacCarthy, J., Bergen, K., Lumley, D., and Magnani, M. Big Data Seismology. *Reviews of Geophysics*, 60(2):2021 000769, 2022. doi: 10.1029/2021RG000769.

Barker, M., Chue Hong, N., Katz, D., Lamprecht, A.-L., Martinez-Ortiz, C., Psomopoulos, F., Harrow, J., Castro, L., Gruenpeter, M., Martinez, P., and Honeyman, T. Introducing the FAIR Principles for research software. *Scientific Data*, 9(1):1, 2022. doi: 10.1038/s41597-022-01710-x.

Beaucé, E., Frank, W., and Romanenko, A. Fast Matched Filter (FMF): An Efficient Seismic Matched-Filter Search for Both CPU and GPU Architectures. *Seismological Research Letters*, 89(1): 165–172, 2017. doi: 10.1785/0220170181.

Beyreuther, M., Barsch, R., Krischer, L., Megies, T., Behr, Y., and Wassermann, J. ObsPy: A Python Toolbox for Seismology. *Seismological Research Letters*, 81(3):530–533, 2010. doi: 10.1785/gssrl.81.3.530.

Chamberlain, C., Hopp, C., Boese, C., Warren-Smith, E., Chambers, D., Chu, S., Michailos, K., and Townend, J. EQcorrscan: Repeating and Near-Repeating Earthquake Detection and Analysis in Python. *Seismological Research Letters*, 89(1):173–181, 2017. doi: 10.1785/0220170151.

Clements, T. and Denolle, M. The Seismic Signature of California's Earthquakes, Droughts, and Floods. *Journal of Geophysical Research: Solid Earth*, 128(1):2022 025553, 2023. doi: 10.1029/2022JB025553.

Gabriel, E., Fagg, G., Bosilca, G., Angskun, T., Dongarra, J., Squyres, J., Sahay, V., Kambadur, P., Barrett, B., and Lumsdaine, A. Open MPI: Goals, concept, and design of a next generation MPI implementation. *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting Budapest, Hungary, September 19-22, 2004*, Proceedings 11:97–104, 2004. doi: 10.1007/978-3-540-30218-6$_1$9.

Gibbons, S. and Ringdal, F. The detection of low magnitude seismic events using array-based waveform correlation. *Geophysical Journal International*, 165(1):149–166, 2006. doi: 10.1111/j.1365-246X.2006.02865.x.

Heesemann, M., Insua, T., Scherwath, M., Juniper, K., and Moran, K. Ocean Networks Canada: From geohazards research laboratories to smart ocean systems. *Oceanography*, 27(2):151–153, 2014. doi: 10.5670/oceanog.2014.50.

Ishii, M., Shearer, P., Houston, H., and Vidale, J. Extent, duration and speed of the 2004 Sumatra–Andaman earthquake imaged by the Hi-Net array. *Nature*, 435(7044):7044, 2005. doi: 10.1038/nature03675.

Jiang, C., Zhang, P., White, M., Pickle, R., and Miller, M. A Detailed Earthquake Catalog for Banda Arc–Australian Plate Collision Zone Using Machine-Learning Phase Picker and an Automated Workflow. *The Seismic Record*, 2(1):1–10, 2022. doi: 10.1785/0320210041.

Krauss, Z. and Wilcock, W. Microseismicity earthquake catalog, Endeavour Segment, Juan de Fuca ridge, 1995–2021 [Dataset. *IEDA*, 2021. doi: 10.26022/IEDA/330498.

Krauss, Z. and Wilcock, W. Investigating microearthquake multiplets using ocean bottom seismometers in a mid-ocean ridge hydrothermal field, 2022.

Krauss, Z., Ni, Y., and Henderson, S. v2.0 Denolle-Lab/seismicloud: Notebooks, Tutorials and Code for "Seismology in the cloud: guidance for the individual researcher", 2023a. doi: 10.5281/zenodo.7948849.

Krauss, Z., Wilcock, W., Heesemann, M., Schlesinger, A., Kukovica, J., and Farrugia, J. A Long-Term Earthquake Catalog for the Endeavour Segment: Constraints on the Extensional Cycle and Evidence for Hydrothermal Venting Supported by Propagating Rifts. *Journal of Geophysical Research: Solid Earth*, 123:2022 025662, 2023b. doi: 10.1029/2022JB025662.

Lindsey, N., Martin, E., Dreger, D., Freifeld, B., Cole, S., James, S., Biondi, B., and Ajo-Franklin, J. Fiber-optic network observations of earthquake wavefields. *Geophysical Research Letters*, 44(23):11–792, 2017. doi: 10.1002/2017GL075722.

MacCarthy, J., Marcillo, O., and Trabant, C. Seismology in the cloud: A new streaming workflow. *Seismological Research Letters*, 91(3):1804–1812, 2020. doi: 10.1785/0220190357.

Michelini, A., Cianetti, S., Gaviano, S., Giunchi, C., Jozinović, D., and Lauciani, V. INSTANCE – the Italian seismic dataset for machine learning. *Earth System Science Data*, 13(12):5509–5544, 2021. doi: 10.5194/essd-13-5509-2021.

Morris, K. *Infrastructure as Code*. O'Reilly Media, Inc, 2020.

Mousavi, S., Sheng, Y., Zhu, W., and Beroza, G. STanford EArthquake Dataset (STEAD): A global data set of seismic signals for AI. *IEEE Access*, 7:179464–179476, 2019. doi: 10.1109/ACCESS.2019.2947848.

Mousavi, S., Ellsworth, W., Zhu, W., Chuang, L., and Beroza, G. Earthquake transformer—An attentive deep-learning model for simultaneous earthquake detection and phase picking. *Nature Communications*, 11(1):3952, 2020. doi: 10.1038/s41467-020-17591-w.

Münchmeyer, J., Woollam, J., Rietbrock, A., Tilmann, F., Lange, D., Bornstein, T., Diehl, T., Giunchi, C., Haslinger, F., and Jozinović, D. Which picker fits my data? A quantitative evaluation of deep learning based seismic pickers. *Journal of Geo-*

physical Research: Solid Earth, 127(1):2021 023499, 2022. doi: 10.1029/2021JB023499.

Ni, Y., Hutko, A., Skene, F., Denolle, M., Malone, S., Bodin, P., Hartog, R., and Wright, A. Curated Pacific Northwest AI-ready Seismic Dataset. *Seismica*, 2(1):1, 2023. doi: 10.26443/seismica.v2i1.368.

Norman, M., Kellen, V., Smallen, S., DeMeulle, B., Strande, S., Lazowska, E., Alterman, N., Fatland, R., Stone, S., Tan, A., Yelick, K., Dusen, E., and Mitchell, J. CloudBank: Managed Services to Simplify Cloud Access for Computer Science Research and Education. *Practice and Experience in Advanced Research Computing*, page 1–4, 2021. doi: 10.1145/3437359.3465586.

Obara, K. Nonvolcanic deep tremor associated with subduction in southwest Japan. *Science*, 296(5573):1679–1681, 2002. doi: 10.1126/science.1070378.

Park, Y., Beroza, G., and Ellsworth, W. A Mitigation Strategy for the Prediction Inconsistency of Neural Phase Pickers. *Seismological Research Letters*, 94(3):1603–1612, 2023. doi: 10.1785/0220230003.

Perol, T., Gharbi, M., and Denolle, M. Convolutional neural network for earthquake detection and location. *Science Advances*, 4(2): 1700578, 2018. doi: 10.1126/sciadv.1700578.

Quinteros, J., Carter, J., Schaeffer, J., Trabant, C., and Pedersen, H. Exploring Approaches for Large Data in Seismology: User and Data Repository Perspectives. *Seismological Research Letters*, 92(3):1531–1540, 2021. doi: 10.1785/0220200390.

Rogers, G. and Dragert, H. Episodic tremor and slip on the Cascadia subduction zone: The chatter of silent slip. *Science*, 300(5627): 1942–1943, 2003. doi: 10.1126/science.1084783.

Ross, Z., Trugman, D., Hauksson, E., and Shearer, P. Searching for hidden earthquakes in Southern California. *Science*, 364(6442): 767–771, 2019. doi: 10.1126/science.aaw6888.

Schorlemmer, D., Euchner, F., Kästli, P., Saul, J., and Group, Q. QuakeML: Status of the XML-based seismological data exchange format. *Annals of Geophysics*, 54(1), 2011. doi: 10.4401/ag.

Schovanec, H., Haynie, K., and Hearne, M. Development of Cloud Computing Infrastructure for the Automated Creation and Delivery of Near-Real-Time Earthquake Impact Products. *AGU Fall Meeting Abstracts*, page 33 –02, 2021.

Scotto di Uccio, F., Scala, A., Festa, G., Picozzi, M., and Beroza, G. Comparing and integrating artificial intelligence and similarity search detection techniques: Application to seismic sequences in Southern Italy. *Geophysical Journal International*, 233(2): 861–874, 2023. doi: 10.1093/gji/ggac487.

Shapiro, N., Campillo, M., Stehly, L., and Ritzwoller, M. High-resolution surface-wave tomography from ambient seismic noise. *Science*, 307(5715):1615–1618, 2005. doi: 10.1126/science.1108339.

Shelly, D., Beroza, G., and Ide, S. Non-volcanic tremor and low-frequency earthquake swarms. *Nature*, 446(7133):305–307, 2007. doi: 10.1038/nature05666.

Shi, P., Grigoli, F., Lanza, F., Beroza, G., Scarabello, L., and Wiemer, S. MALMI: An Automated Earthquake Detection and Location Workflow Based on Machine Learning and Waveform Migration. *Seismological Research Letters*, 93(5):2467–2483, 2022. doi: 10.1785/0220220071.

Tan, Y., Waldhauser, F., Ellsworth, W., Zhang, M., Zhu, W., Michele, M., Chiaraluce, L., Beroza, G., and Segou, M. Machine-Learning-Based High-Resolution Earthquake Catalog Reveals How Complex Fault Structures Were Activated during the 2016–2017 Central Italy Sequence. *The Seismic Record*, 1(1):11–19, 2021. doi: 10.1785/0320210001.

Trabant, C., Hutko, A., Bahavar, M., Karstens, R., Ahern, T., and Aster, R. Data Products at the IRIS DMC: Stepping Stones for Research and Other Applications. *Seismological Research Letters*, 83(5):846–854, 2012. doi: 10.1785/0220120032.

Turin, G. An introduction to matched filters. *IRE Transactions on Information Theory*, 6(3):311–329, 1960. doi: 10.1109/TIT.1960.1057571.

Walter, J., Ogwari, P., Thiel, A., Ferrer, F., and Woelfel, I. easyQuake: Putting Machine Learning to Work for Your Regional Seismic Network or Local Earthquake Study. *Seismological Research Letters*, 92(1):555–563, 2021. doi: 10.1785/0220200226.

Wilkinson, M., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Silva Santos, L., Bourne, P., Bouwman, J., Brookes, A., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C., Finkers, R., and Mons, B. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1):1, 2016. doi: 10.1038/sdata.2016.18.

Woollam, J., Münchmeyer, J., Tilmann, F., Rietbrock, A., Lange, D., Bornstein, T., Diehl, T., Giunchi, C., Haslinger, F., Jozinović, D., Michelini, A., Saul, J., and Soto, H. SeisBench—A Toolbox for Machine Learning in Seismology. *Seismological Research Letters*, 93(3):1695–1709, 2022. doi: 10.1785/0220210324.

Yoo, A., Jette, M., and Grondona, M. SLURM: Simple Linux Utility for Resource Management. In Feitelson, D., Rudolph, L., and Schwiegelshohn, U., editors, *Job Scheduling Strategies for Parallel Processing*, page 44–60. Springer, 2003. doi: 10.1007/10968987$_3$.

Yoon, C., O'Reilly, O., Bergen, K., and Beroza, G. Earthquake detection through computationally efficient similarity search. *Science Advances*, 1(11):1501057, 2015. doi: 10.1126/sciadv.1501057.

Yu, E., Bhaskaran, A., Chen, S., Ross, Z., Hauksson, E., and Clayton, R. Southern California Earthquake Data Now Available in the AWS Cloud. *Seismological Research Letters*, 92(5):3238–3247, 2021. doi: 10.1785/0220210039.

Yu, Z., Wang, W., and Chen, Y. Benchmark on the accuracy and efficiency of several neural network based phase pickers using datasets from China Seismic Network. *Earthquake Science*, 36 (2):113–131, 2023. doi: 10.1016/j.eqs.2022.10.001.

Zhang, M., Liu, M., Feng, T., Wang, R., and Zhu, W. LOC-FLOW: An End-to-End Machine Learning-Based High-Precision Earthquake Location Workflow. *Seismological Research Letters*, 93(5): 2426–2438, 2022. doi: 10.1785/0220220019.

Zhou, Y., Ghosh, A., Fang, L., Yue, H., Zhou, S., and Su, Y. A high-resolution seismic catalog for the 2021 MS6.4/MW6.1 Yangbi earthquake sequence. *Earthquake Science*, 34(5):390–398, 2021. doi: 10.29382/eqs-2021-0031.

Zhu, W., McBrearty, I., Mousavi, S., Ellsworth, W., and Beroza, G. Earthquake Phase Association Using a Bayesian Gaussian Mixture Model. *Journal of Geophysical Research: Solid Earth*, 127 (5):2021 023249, 2022. doi: 10.1029/2021JB023249.

Zhu, W., Hou, A., Yang, R., Datta, A., Mousavi, S., Ellsworth, W., and Beroza, G. QuakeFlow: A scalable machine-learning-based earthquake monitoring workflow with cloud computing. *Geophysical Journal International*, 232(1):684–693, 2023. doi: 10.1093/gji/ggac355.