Unlocking AI Capabilities

Advanced Applications through High-Performance Computing

MODAL | 19. 04. 2024

Stefano Izzo, Sara Amitrano, Martina Savoia, Edoardo Prezioso, Fabio Giampaolo and Francesco Piccialli

Department of Mathematics and Applications "Renato Caccioppoli", University of Naples Federico II, Italy
***M**athematical m**O**deling and **D**ata **A**na**L**ysis (M.O.D.A.L) research group*

Deep Learning-based HPC applications

Eco-FL

Energy-Efficient Client Selection in a Federated Learning scenario.

AI Eyes in the Sky

Acquiring satellite images and detecting illegal landfills using Deep Learning models.

YOLO and HPC

Training object-detection models through HPC.

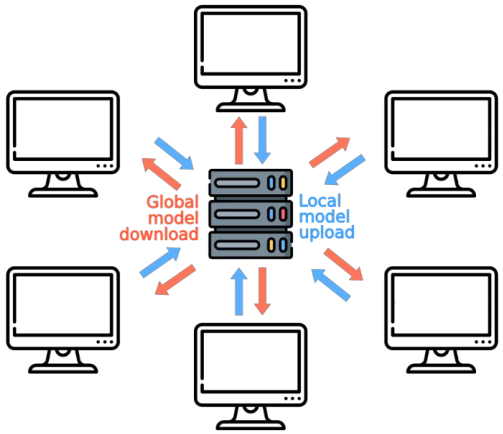
Focal mechanism

Simulation of numerous different earthquake configurations in a dense grid of hypocenters.

Eco-FL: Enhancing Federated Learning Sustainability in Edge Computing through Energy-Efficient Client Selection

Application in the field of energy sustainability

Federated Learning (FL)



FL enables the circumvention of problems linked to **data distribution**, safeguarding **privacy**.

If data does not follow the same probability distribution and can be correlated or have different characteristics from one client to another, it is referred to as **Non-IID** (Non-Independently and Identically Distributed).

FL moves models where data resides, instead of centralizing the data for model learning.

- **Server** starts the FL process by initializing a global model, which is distributed to all participating clients.
- **Clients** train the model locally on their data and sends their model weights.

1st round

- **Server** collects and aggregates clients updates to create an improved global model and evaluates the performance. Then it sends the updated model to the clients.
- **Clients** train the model locally on their data and sends their model weights.

2nd round

and so on …

- **Server** aggregates clients updates, evaluates the performance and stops the FL process.

Client selection optimization

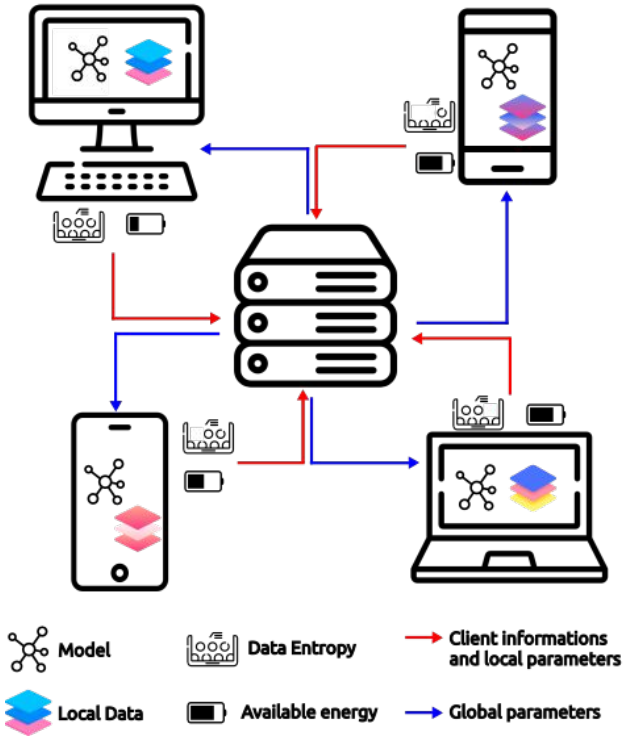
Client selection plays a crucial role in FL performance, affecting both **accuracy** and **energy consumption**. The proposed methodology aims to **optimize** client selection through an optimization problem focusing on two key aspects: **residual energy** and **data entropy**.

- **Residual energy**: Clients are evaluated based on their remaining **energy reserves** to select devices with the **highest charge**. Those with excessively low energy levels are excluded to prevent disruptions due to energy depletion during FL, ensuring that selected clients can sustain model training requirements.
- **Data Entropy**: Each client holds different types and amounts of data, contributing to the FL model. Assessment is based on the entropy of local datasets, **prioritizing** clients with **diverse** and **information-rich** data. This ensures the global model adapts well to varied data distributions.

Flower and HPC

To manage the FL process and establish a collaborative environment between a server and multiple clients, the **Flower Framework**[1] was utilized. **Flower** is known for its ability to extend FL implementations to mobile and wireless clients, accommodating a range of computational, memory, and network resources and its adaptability in incorporating emerging algorithms, training strategies, and communication protocols.

IBISCO has been useful for performing FL for 1500 rounds, with a high number of clients (**100**) each running on a different GPU and managed by a separate server on a different GPU.



1. Beutel, D.J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Li, K.H., Parcollet, T., de Gusmão, P.P.B., et al., 2022. Flower: A friendly federated learning framework

The case study

Datasets

- CIFAR-10
- CIFAR-100
- CINIC-10

Benchmarks

- FedEntropy
- FedAvg
- FedProx

FL parameters

- 100 clients
- 1500 rounds
- 5 local epochs

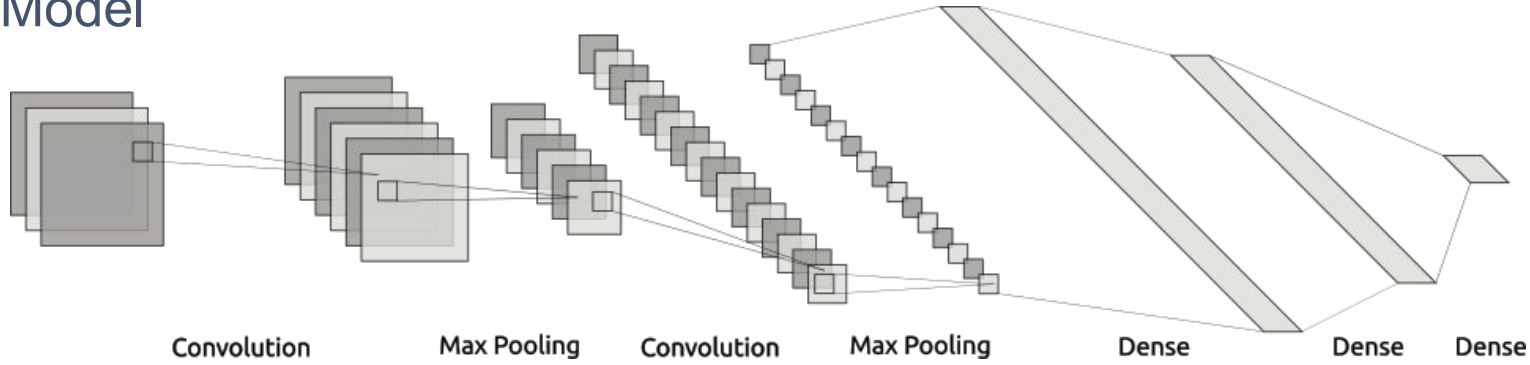
Task

Classify images into one class

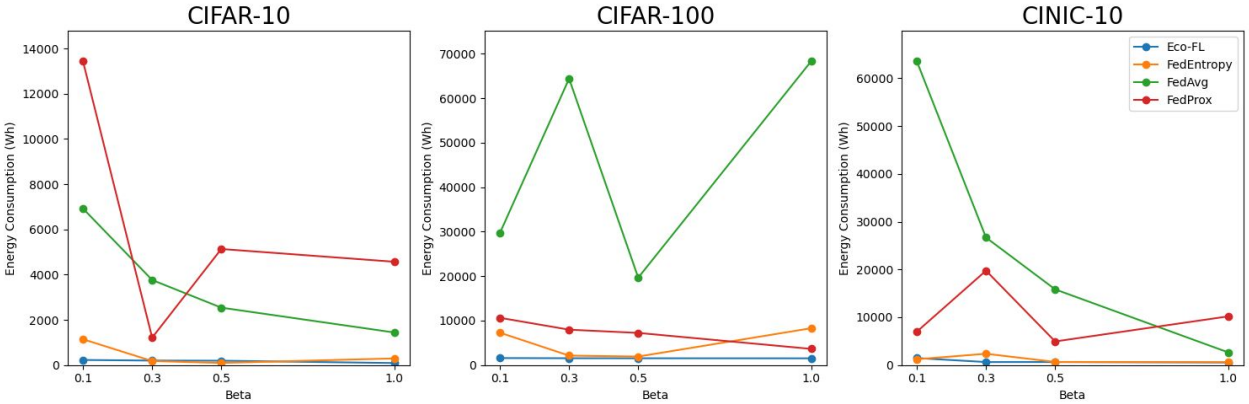
Scenarios

- Homogeneous: available energy = 15 Wh
- Heterogeneous: available energy from a Gaussian probability distribution centered in 15 Wh with a standard deviation of 2Wh

Model



Results



Homogeneous case: Comparison of energy consumption between Eco-FL and the three benchmark methods.

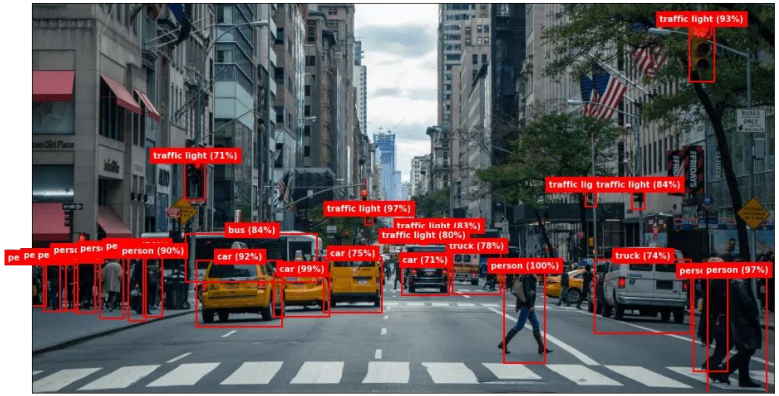
Dataset	β	Homogeneous	Heterogeneous
CIFAR-10	0.1	227	125
	0.3	200	109
	0.5	194	120
	1.0	91	123
CIFAR-100	0.1	1583	1589
	0.3	1546	1560
	0.5	1536	1579
	1.0	1524	1528
CINIC-10	0.1	1471	938
	0.3	637	394
	0.5	647	323
	1.0	550	254

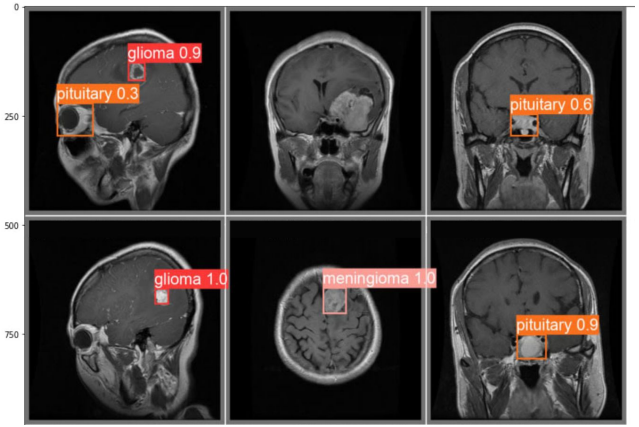
Comparison of energy consumption of Eco-FL between homogeneous and heterogeneous case.

YOLO: Real-Time Object Detection

Object Detection

Object detection is a computer vision method that aims to identify and classify instances of objects of certain classes within images, providing bounding box information for each object.
Object detection algorithms find extensive applications across domains such as autonomous vehicle navigation, intelligent video surveillance, medical detections and more.

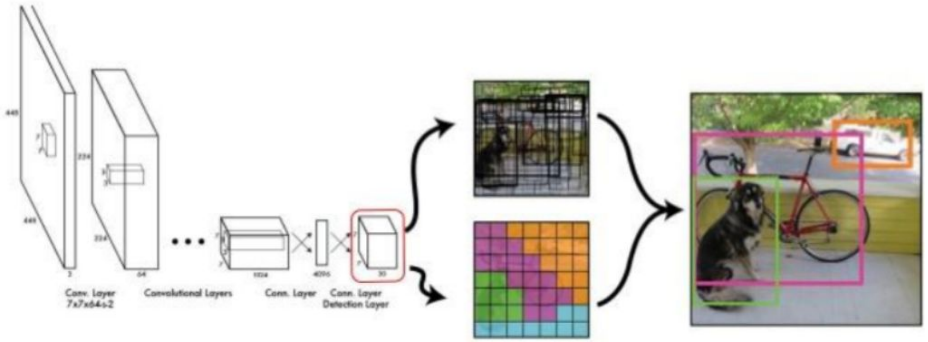




YOLO- You Only Look Once

The **YOLO** algorithm, developed by *Joseph Redmon et al. in 2015*, is one of the most representative works in the field of object detection. It is a **single-stage object detector** that uses a convolutional neural network (CNN) to predict the **bounding boxes** and **class probabilities** of objects in input images.

- The input image is divided into an *S x S* grid cells of equal shape.
- Each cell predicts *B* bounding boxes, confidence scores for those boxes and *C* class probabilities.
- *Non-Maximum Suppression* method is applied to remove duplicate predictions by keeping only the bounding box with the highest confidence score among overlapping boxes for the same object.



YOLO revolutionizes object detection with its real-time performance, high accuracy and versatility, but its effectiveness is deeply tied to two critical factors: **data** and **model complexity**.

- YOLO employs a sophisticated convolutional neural network architecture, which entails millions of parameters.

Model size	Input size	Parameters (Millions)
YOLOv8n	640	3.2
YOLOv8s	640	11.2
YOLOv8m	640	25.9
YOLOv8l	640	43.7
YOLOv8x	640	68.2

Table: Number of trainable parameters varies across different sizes of utilized YOLO implementation version.

- Training the YOLO model to accurately detect objects requires exposure to a diverse range of visual scenarios.
- Access to large dataset can enables fine-tuning for specialized applications.
- More data allows for more effective optimization of the model's parameters during training, which can lead to improved accuracy, robustness, and speed of inference.

HPC Results

Goal: enhance the performance of the YOLO model on live webcam streams by fine-tuning it with custom datasets.

Dataset

- Train set: 994545 images
- Validation set: 31876 images

Gathered from COCO, Open Images v7, Pascal VOC and Roboflow datasets.

Training parameters

- Model size: YOLOv8l, YOLOv8x
- 1000 epochs
- 26 classes

Training time

Model size	1 GPU	4 GPUs	24 GPUs	32 GPUs	36 GPUs
YOLOv8l	~ 131 min	~ 100 min	~ 37 min	~ 19 min	~ 18 min
YOLOv8x	~ 195 min	~ 150 min	~ 45 min	~ 30 min	~ 20 min

Table: mean training time for epoch

AI Eyes in the Sky

Satellite Surveillance for Landfill Detection

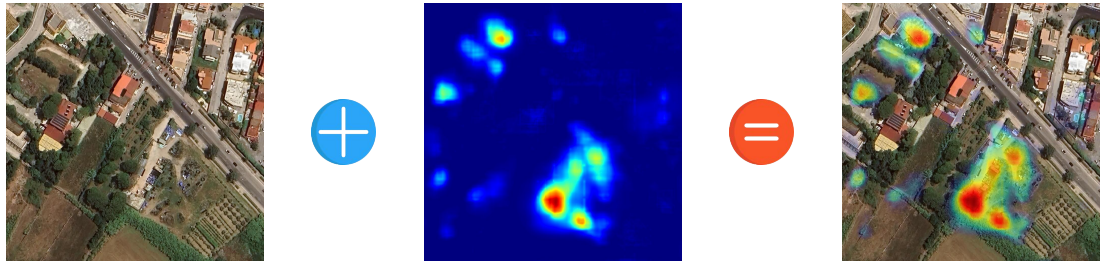
Waste detection through CAMs

Class-activation maps (CAMs) are a technique used to visualize and interpret the decision-making process of convolutional neural networks (CNNs) in artificial vision tasks.

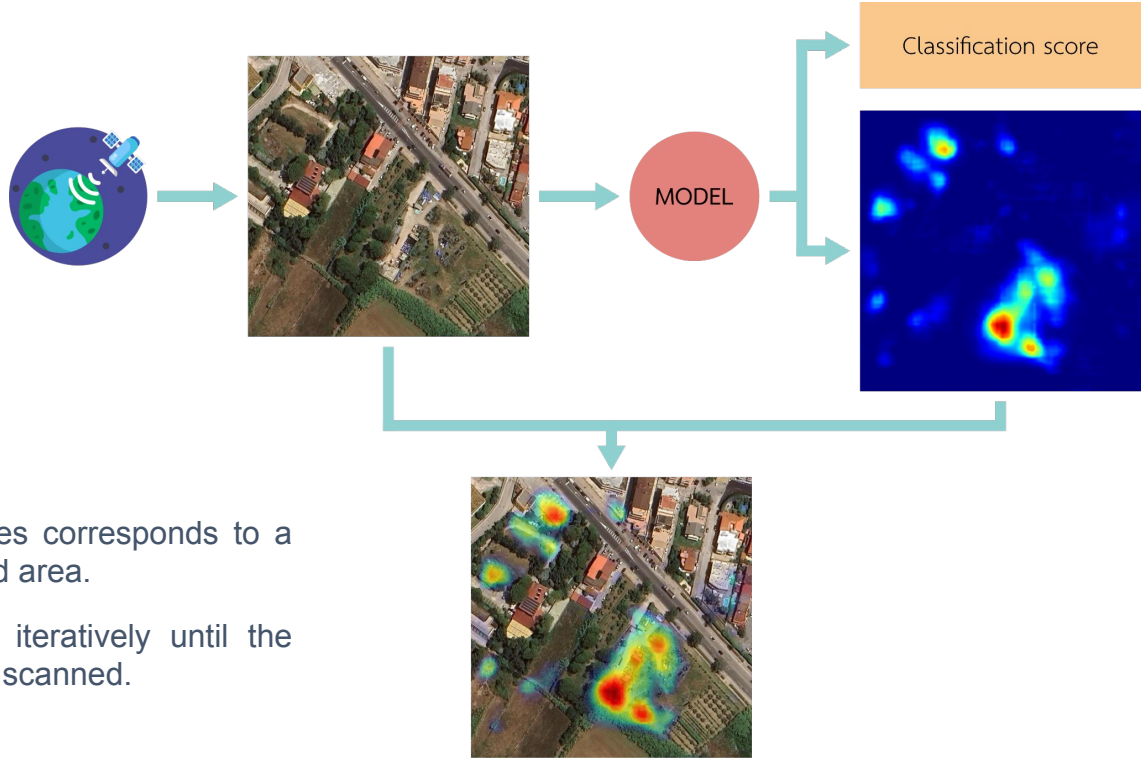
CAMs help to understand which parts of an image *were relevant to the neural network* during the classification of an image. This is particularly useful for making CNNs more **interpretable** and for verifying if they are considering the right visual features for making a decision.

The implementation of CAMs involves inserting a **Global Average Pooling (GAP)** layer after the last convolutional layer of a CNN, which calculates the average of each feature map produced by the convolutional layer.

Therefore, when there is an image and a specific class, the CAM shows the **regions** of the image that **most contributed** to the class decision. This is achieved by overlaying the activation map on the original image, highlighting the important areas.



Waste’s Framework

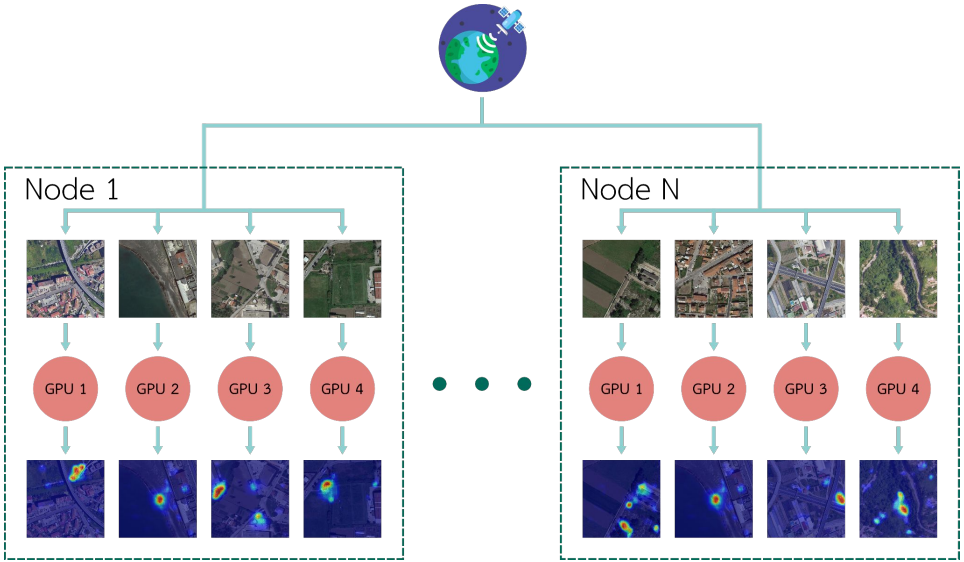


Each of the input images corresponds to a small part of the selected area.

The process continues iteratively until the entire zone of interest is scanned.

HPC Implementation

An area can include a vast number of tiles, which would make the overall analysis extremely prolonged. However, the process is **inherently parallel**, as there is no mutual dependency between the images and their respective results. Therefore, it is feasible to **divide** the area into **subsections**, which can be processed in parallel by **multiple nodes**.



HPC Results

Procedure Implementation: The procedure was implemented by selecting the entire area of the **Caserta area (NA)**. By using 4 tiles, we were able to generate a total of **80,080 satellite images**.

Results Analysis: The table below illustrates the processing times required, highlighting variations in relation to the number of **nodes** used and the number of **GPUs** utilized.

Nodes	GPUs per Nodes	Total time (h)	Time per Image (s)
1	1	79.28	3.56
1	4	42.13	1.89
4	1	21.11	3.59
4	4	8.54	1.44

Focal mechanism and HPC for AI

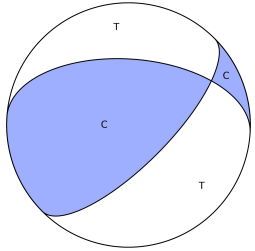
Synthetic generation of seismological data

Focal mechanism

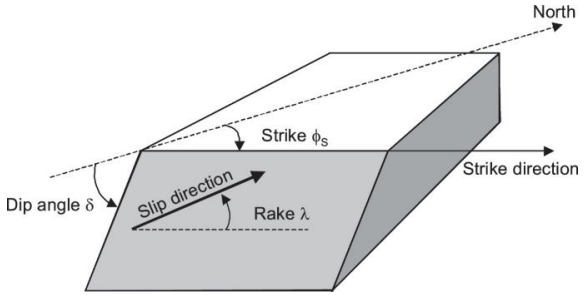
Focal mechanisms, often visualized through *beachball diagrams*, serve as a tool to interpret how an earthquake occurs at its source. These diagrams represent different directions in which the Earth can break during an earthquake, showing us the 'story' of the quake—where the stress came from and how it was released.

Such data can be expressed as three angular quantities:
strike $\in [0, 360[$
dip $\in [0, 90]$
rake $\in [-180, 180[$

These diagrams are not just crucial for predicting earthquake behavior but also for understanding fundamental geological processes. By studying these patterns, researchers can gain insights into the underground stress fields and how they influence seismic activities.



A focal mechanism, divided in two regions: compressional (C) and tensorial (T). It is generated with the following values:
strike = 45, dip = 60, rake = 60



Graphical definition of strike, dip and rake.

Data Generation

Given an earthquake, its focal mechanism is estimated through the **first-motion P-wave polarities** ($p \in \{-1,0,1\}$) obtained from the seismic stations located nearby the earthquake, but the estimated value is affected by large uncertainties (given by imprecise instrument, non-detailed geological modelling). It is fundamental to reduce such *uncertainty*.

In this work we generated a dataset $\mathbb{D}$, based on the theoretical propagation model (*Aki and Richards (1980)*), with the aim to understand such uncertainty. Each row of the dataset is composed as follows:

- **earthquake hypocenter** (*x, y, z*),
- **focal mechanism** (*strike, dip, rake*),
- **angular propagation data for each station** (*azimuth, takeoff*),
- theoretical **P wave radiation amplitude** for each station,
- **first-motion P-wave polarity** for each station.

The generation procedure takes in input **x, y, z, strike, dip** and **rake** and returns in output **azimuth, takeoff, amplitude, polarity.**

Biggest issue: considering all the combinations of focal mechanisms, earthquake epicenters and randomizations for robust estimations, we need to generate very large data (~trillions or rows).

Biggest advantage: such generation can be parallelized, hence HPC can help this.

HPC solution

We deployed two levels of parallelism: **Node parallelism** and **CPU parallelism**

Node parallelism: each HPC node receives a batch of focal mechanisms and all the hypocenters of the earthquake in input and saves *N* dataset files, with *N* the number of hypocenters.

CPU parallelism: a parallel for loop is run for each hypocenter, each core receives the batch of focal mechanisms for the node and a single hypocenter of the earthquake and generates a dataset file.

Our setup:
Hypocenter grid: ~13k points (1 km interval)
Focal mechanism grid: ~94k points (5 degrees interval)

Hardware:
Number of jobs: 32
Number of cpu cores used per each node: 48

Results:
Overall generation time with ibisco HPC: ~2 days

Estimated time without HPC (Intel Core i9-10980XE):

- *single core*: ~1 year
- *parallel with all physical cores (18)*: ~1 month with linear speedup

Conclusion:
With HPC, it is possible to generate even more granular data in a scalable manner.

Thoughts and Considerations

In conclusion, the applications proposed here highlight the indispensable role of High-Performance Computing in tackling complex scenarios. HPC has been pivotal for:

- **Simulating Federated Learning Scenarios**: Enhancing collaborative machine learning without centralized data, ensuring privacy, and reducing communication overhead.
- **Accelerating Deep Learning Models**: Leveraging parallel computing to efficiently train complex models on large datasets, significantly reducing training times.
- **Resource Redistribution for Satellite Image Analysis**: Effectively allocating and parallelizing resources to analyze extensive satellite imagery data, leading to faster results.
- **Earthquake Configuration Simulation**: Utilizing HPC to simulate a wide array of earthquake scenarios, aiding in predictive analysis and disaster preparedness.

These applications demonstrate the critical role of HPC in advancing the field of Artificial Intelligence. They foster an environment where innovation thrives, leading to breakthroughs that are not achievable with conventional computing resources.

Despite significant progress being made in these applications with naive parallelization approaches and the usage of HPC systems, there is still vast potential for optimization. By refining our strategies and fully leveraging the available computational power, we can achieve greater efficiencies and enhance the productivity of AI applications.

Thanks for your attention