EPJ Web of Conferences **295**, 04007 (2024) https://doi.org/10.1051/epjconf/202429504007
CHEP 2023

ATLAS Data Analysis using a Parallel Workflow on Distributed Cloud-based Services with GPUs

Jay Sandesara[1,*], *Rafael* Coelho Lopes de Sa[1], *Verena* Martinez Outschoorn[1], *Fernando* Barreiro Megino[2], *Johannes* Elmsheuser[3], and *Alexei* Klimentov[3] on behalf of the ATLAS Computing Activity

[1]University of Massachusetts Amherst, Amherst, MA, USA
[2]University of Texas at Arlington, Arlington, TX, USA
[3]Brookhaven National Laboratory, Upton, NY, USA

Abstract. A new type of parallel workflow is developed for the ATLAS experiment at the Large Hadron Collider, that makes use of distributed computing combined with a cloud-based infrastructure. This has been developed for a specific type of analysis using ATLAS data, one popularly referred to as Simulation-Based Inference (SBI). The JAX library is used for the parts of the workflow to compute gradients as well as accelerate program execution using just-in-time compilation, which becomes essential in a full SBI analysis and can also offer significant speed-ups in more traditional types of analysis.

1 Introduction to Simulation-Based Inference

In high energy physics experiments like ATLAS [1] at the Large Hadron Collider, the advancement in Monte Carlo simulators over the years has enabled the accurate modelling of highly dense and complex physical interactions taking place inside particle detectors. Using such simulators, it is possible to sample an aribtrarily large number of events corresponding to a physics hypothesis.

In order to do statistical inference given an observed set of data, it is not enough to sample events using the Monte Carlo simulations but also to be able to quantify their agreement with a hypothesis of interest. The way to do this is to estimate the so-called likelihood function $L(\theta|x)$ associated with a set of observations x, where θ is the parameter corresponding to the hypothesis under study. We can think of the simulator as a computer program that takes as input a vector of parameters θ and produces a data vector $x \sim p(x|\theta)$ as output, sampled from some probability density $p(x|\theta)$ of the given hypothesis model, as shown in Figure 1. This Monte Carlo sampling is done over a series of thousands of internal states or latent variables that describe the complex interactions of particles inside the detector. This integration over thousands of variables is impractical to perform analytically making it challenging to reverse the direction from the simulated or observed set of events x to calculate $p(x|\theta)$, which is needed to build the likelihood $L(\theta|x)$ for statistical inference.

*e-mail: jay.ajitbhai.sandesara@cern.ch

Figure 1: Reconstructed events x as observed in a detector can be simulated easily from an underlying hypothesis θ. But it is difficult to reproduce the exact probability density $p(x|\theta)$ given the reconstructed events x.

Simulation-Based Inference (SBI) refers to an emerging new set of techniques that can be used to perform statistical inference in situations where the likelihood function is analytically intractable [2]. In the context of the ATLAS analysis, we have adopted one such SBI technique that employs deep neural networks (NNs), trained using simulated data, to predict event-by-event estimates of the likelihood ratio between two hypotheses as a function of the full multi-dimensional final state. The likelihood ratios are then utilized to perform statistical inference using the profile likelihood method. A high-level illustration of this SBI analysis can be found in Figure 2.

This approach of directly estimating the likelihood ratios as a function of the full multi-dimensional input x can be significantly more powerful than the traditional approach of estimating Poisson likelihoods as function of binned low-dimensional summary observable $f(x)$ to perform statistical inference, as shown using toy experiments in Ref. [2].

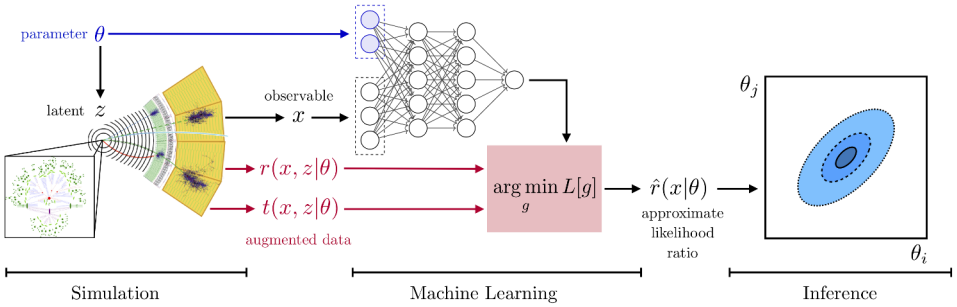
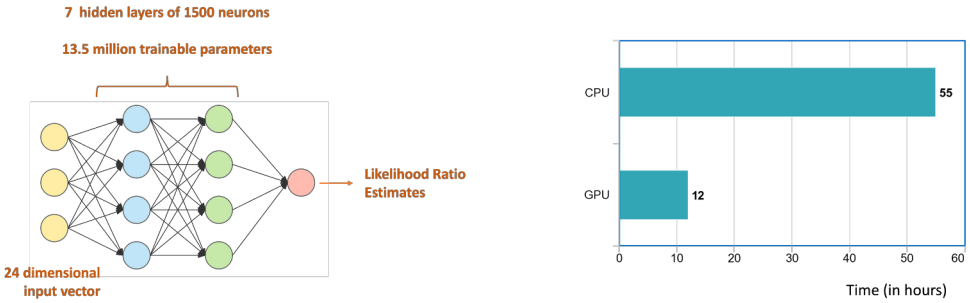


Figure 2: An overview of SBI analysis. Deep NNs are used to predict estimates of likelihood ratios corresponding to a given hypothesis parameter θ using fully simulated events x. The likelihood ratios are then used for direct inference on the parameter of interest, without the need for binning the output and doing a Poisson fit. Figure taken from Ref. [3].

2 Computational Challenges

The prediction of high quality estimates of the likelihood ratios using real experimental data requires an ensemble of deep and wide NNs. Figure 3(a) shows an example of the NN architecture used in the full ATLAS analysis to model likelihood ratios with minimal bias and variance.

Training these deep and wide NNs using CPUs can be inefficient. As shown in Figure 3(b), GPUs offer a significant speed-up compared to CPU for training the same NN using O(1M) simulated events. In order to get an estimate of the statistical uncertainty and to increase the precision of the predictions, a large ensemble of the deep NNs like the one shown in Figure 3(a) are trained. Additional NNs are trained to model the likelihood ratios as a function of the O(100) nuisance parameters needed in a typical experimental measurement. The total number of NNs needed to estimate the fully parameterized likelihood ratios with sufficient accuracy and precision is on the order of many thousands, making large scale GPU infrastructure essential for an SBI analysis in a real experimental setting.



(a) (b)

Figure 3: (a) NN architecture of one of the many thousands of NNs used in the analysis, each with millions of trainable parameters. (b) Training time comparison, between a CPU and an NVIDIA T4 GPU, for one of the NNs used in the analysis. Training on GPU offers almost 5× the speed-up compared to the impractically large training time on CPU, with the same O(1M) simulated events used for each training. This time difference becomes especially relevant when optimizing an ensemble of thousands of NNs.

3 Use of cloud-based distributed computing

The distributed computing system at ATLAS is built around two main components: the workflow management system PanDA [4] and the data management system Rucio [5]. Together, PanDA and Rucio have been developed to manage the distribution of jobs across a global network of computational centers, known as the Worldwide LHC Computing Grid (WLCG) [6]. Within this computing grid, the sites processing ATLAS data do not have access to a large number of GPUs as they are typically used for more traditional tasks that do not require such infrastructure.

In recent years, cloud-based computing has advanced rapidly, allowing users to potentially gain access to a range of powerful computing hardware, with considerable flexibility in terms of resource allocation based on the user needs. Notably, these services have the potential to allow cutting-edge R&D projects to be performed without the need for an upfront investment in a dedicated hardware setup. Driven by these motivations, the ATLAS experiment has pursued several initiatives to integrate cloud-based services with its existing distributed computing framework. Consequently, PanDA and Rucio have now been seamlessly integrated with the commercial cloud services [7], enabling ATLAS users to submit jobs to the Kubernetes clusters [8] offered by leading commercial cloud companies like Google and Amazon. As shown in Figure 4, the Harvester [9] module of PanDA interfaces with Kubernetes clusters to assign jobs efficiently to the available resources. For a more detailed outline on the technical implementation, see Ref. [10].

The integration of cloud services with PanDA offers the user convenient access to a range of powerful and scalable resources on the cloud. In the specific instance of NN training for the SBI analysis, 200 nodes were leveraged on the Google Cloud Platform, each equipped with 4 cores of CPU and an NVIDIA T4 GPU. This has enabled the SBI analysis to be performed with a level of precision that would be otherwise impossible without a costly infrastructure setup.

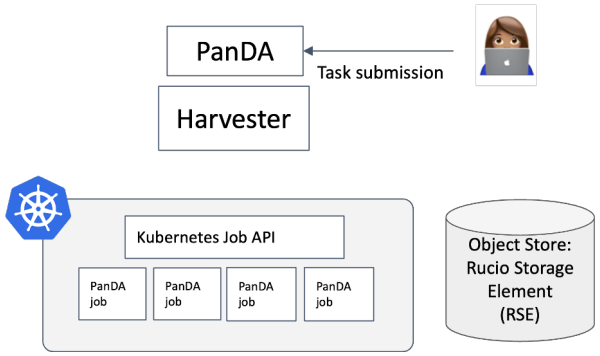


Figure 4: PanDA uses the Harvester module to submit jobs via the Kubernetes API to available nodes with GPUs on the cloud. In the cloud environment, the Rucio Storage Element is implemented as an Object Store bucket [11], enabling essential functionalities such as download and upload of data from the jobs [10].

4 NN Training Workflow

A streamlined workflow has been designed to exploit the cloud-based distributed computing infrastructure, wherein the training of each NN in the ensemble is submitted to a single GPU node via PanDA, enabling simultaneous training across multiple nodes. Rucio Storage Element (RSE) is used to store the inputs and outputs for these submitted jobs, while portable container images from Docker [12] are used to run the NN trainings on the remote node, using Tensorflow [13] as shown in Figure 5. Following the training, the results are downloaded from the RSE to a local machine, and are analyzed as appropriate using another diagnostic script. Figure 6 shows a diagram of the NNs being trained in parallel.

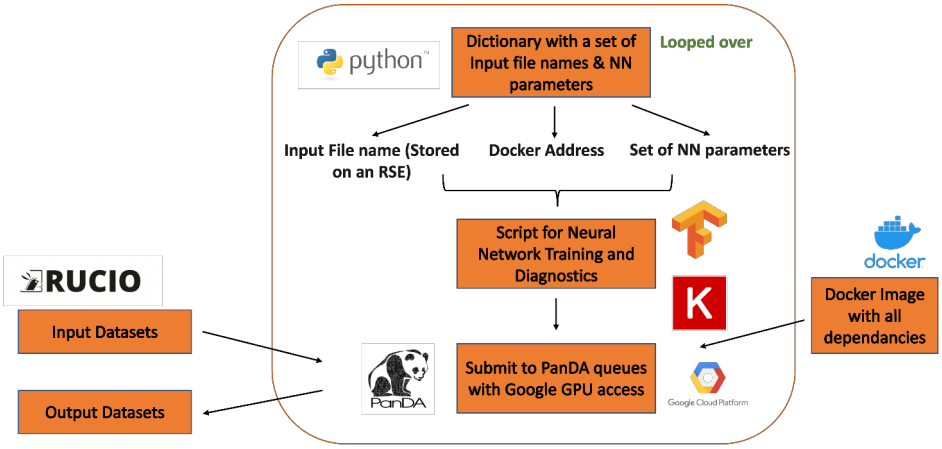


Figure 5: Workflow used to submit jobs to the PanDA queue that has access to GPU nodes on the Google Cloud Platform for the training of several NNs in parallel, one per node. A Docker container is used to run the training on the remote node using the Tensorflow library, and Rucio Storage Element is used to store the inputs and outputs in the training.

We made use of this infrastructure to perform hyper-parameter (HP) optimizations for single NNs and ensemble of NNs. For single NN optimizations, O(1k) NNs are submitted at

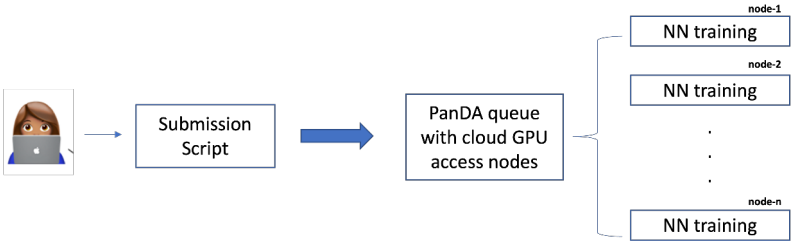


Figure 6: Overview of the submission for parallel training of NNs on the cloud. PanDA interacts with the Kubernetes API for allocation of available GPU nodes on Google Cloud as shown in Figure 4.

a time with a grid of HPs, one set per training to be executed in parallel using the workflow described above. In approximately a day, about a thousand fully trained NNs are available to be diagnosed for finding the optimal set of HPs. Once this set of HPs is found, a submission of an ensemble of NNs was done followed by the diagnostics and optimization for the combined NN predictions.

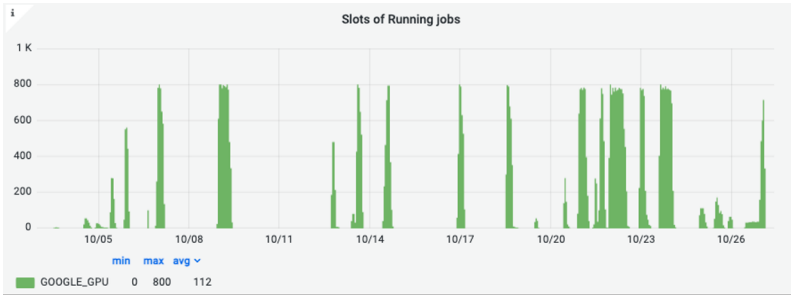


Figure 7: Summary of cloud resources used for the SBI analysis R&D, in October 2022. The vertical axis shows the number of CPU cores in use at given time. Total of 200 nodes were made available for the analysis, each with four cores of CPU and an NVIDIA T4 GPU.

Figure 7 shows the graph of real CPU usage on Google cloud nodes for the HP optimizations in the ATLAS SBI analysis during the month of October 2022. A maximum capacity of 200 nodes was made available, each equipped with 4 cores of CPU and an NVIDIA T4 GPU, to accommodate analysis requirements. The maximum number of available nodes is not constrained in principle and can be adjusted based on the analysis needs. From Figure7, it is evident that the infrastructure had the capacity to support the concurrent execution of a maximum of 200 training jobs, corresponding to all 800 cores of CPU in use. Notably, the availability of resources and their quantity depend on the number of active jobs, as can be seen by the varying sizes of the several peaks in CPU usage. These nodes on the cloud are allocated on an on-demand basis, ensuring no additional infrastructure costs during periods without submitted jobs.

5 Profile Likelihood Workflow

Once the likelihood ratios have been calculated using the NN ensemble, the final step of the analysis is to compute the best-fit values and uncertainties of model parameters using the profile likelihood test statistic $T(\theta)$, where θ is the set of parameters. With the SBI analysis,

the computation time to perform parameter fitting using $T(\theta)$ with the typical O(100) nuisance parameters (NPs) of the ATLAS experiment is significantly higher than that of a binned fit. Unlike a binned fit with parameterized likelihood ratios for number of bins ranging from 10 to 20, in the SBI analysis there are event-by-event parameterized likelihood ratio predictions for events ranging from 1M to 10M in number. Subsequently, the new analysis makes use of auto-differentiation and just-in-time (JIT) compilation using the JAX library [14], which offers the potential to reduce computation time by several orders of magnitude. A comparison of the time to find the best fit values of parameters using the Migrad algorithm of iminuit [15, 16] library is shown in Figure 8.

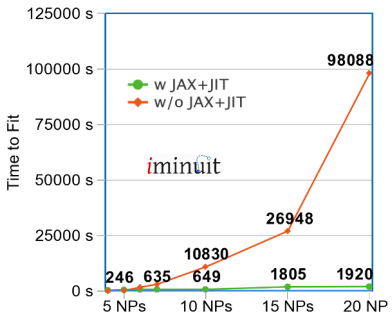


Figure 8: Comparison of the time, in seconds, to perform a profile likelihood fit using the SBI test statistic written as a JIT-compiled JAX function and as a default python function. The fit is performed using event-by-event systematic variations for an Asimov dataset with O(10M) events. While the computation time remains similar between the python function and the JIT-compiled JAX function for a small number of NPs, the latter scales significantly better as the number of NPs in the fit increases.

The new analysis makes use of the auto-differentiation tool offered by JAX to infer both the uncertainties on the NPs and the propagation of these uncertainties to the best fit value of the parameter of interest, by calculating the exact second derivative Hessian matrix of the test statistic function $T(\theta)$. Calculating the Hessian matrix in the SBI analysis is unfeasible, since storing weights during the forward and/or reverse pass calculation of the second derivative $\nabla^2 T : \mathbb{R}^{\mathrm{O}(100)} \to \mathbb{R}^{\mathrm{O}(100)\times\mathrm{O}(100)}$ requires enormous memory resources owing to the event-by-event nature of the analysis and the O(10M) events in a typical application. There exists a memory-efficient solution, which can then be executed in a parallel workflow using the distributed cloud-based computing infrastructure.

To make the computation more memory-efficient, instead of directly materializing the full Hessian matrix, we calculate the Hessian vector product (HVP) making use of the following identity:

$$\nabla^2 T(x)\, v = \nabla \left[\nabla T\,(x) \cdot v\right] \tag{1}$$

The first step is to calculate the Jacobian $\nabla T : \mathbb{R}^{\mathrm{O}(100)} \to \mathbb{R}^{\mathrm{O}(100)}$, and take its dot product with some vector v. This step transforms the Jacobian into a scalar-valued function $\nabla T \cdot v : \mathbb{R}^{\mathrm{O}(100)} \to \mathbb{R}$. This reduces the required memory resources for the subsequent computation of the gradient of this scalar valued function $\nabla \left[\nabla T \cdot v\right] : \mathbb{R}^{\mathrm{O}(100)} \to \mathbb{R}^{\mathrm{O}(100)}$ giving us $\nabla^2 T(x)\, v$.

Repeating the computations of $\nabla^2 T(x)\, v$ with the vector v chosen to be one of the orthogonal unit vectors each time, we extract the Hessian matrix $\nabla^2 T(x)$ one row at a time. By only computing the gradients of scalar valued functions, we significantly cut down on the memory requirements. For SBI analysis, this new solution makes the calculation of the Hessian matrix possible to perform in practice. However, the memory requirement can still add up to several hundred gigabytes, necessitating the use of specialized hardware.

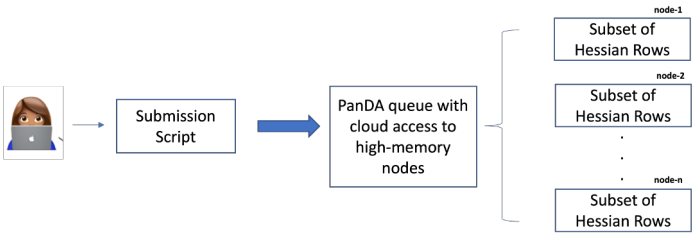


Figure 9: Overview of the submission for parallel computation of Hessian matrix rows on the cloud. PanDA interacts with the Kubernetes API for allocation of available high memory nodes on Google Cloud as shown in Figure 4.

Cloud-based distributed computing, owing to its power and elasticity, once again provides a solution to this computationally challenging scenario. Several high memory nodes, each with a terabyte of memory, were used on the Google Cloud Platform for the Hessian matrix computations. Leveraging the row-by-row computation of the Hessian matrix, the task of estimating the full Hessian matrix can be divided into several sub-tasks each estimating a subset of the Hessian rows. Each of these sub-tasks are then submitted to the high memory node on the cloud using PanDA, and the calculation of the subsets of the full Hessian matrix is executed in parallel as shown in Figure 9.

6 Conclusions and Outlook

Scalable, on-demand GPU and high-memory CPU infrastructure made available using the Google Cloud Platform has made a full experimental analysis with Simulation-Based Inference practically possible in the ATLAS experiment. The full workflow is depicted in Figure 10.

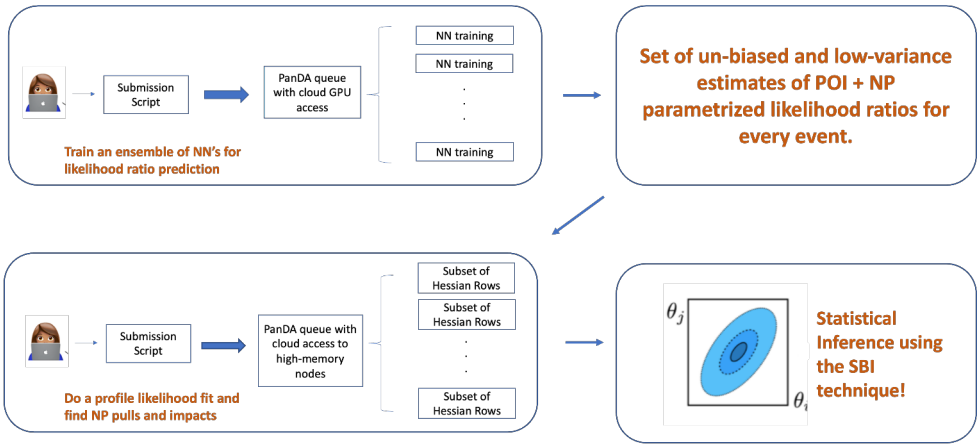


Figure 10: Overview of the full Simulation-Based Inference analysis workflow and how the cloud-based distributed computing infrastructure was leveraged in the different steps.

By leveraging the power and flexibility of cloud infrastructure, the likelihood ratios in the SBI analysis of ATLAS experiment data are calculated through an aggregate of over a billion NN parameters, distributed across several thousand deep NNs. This marks a new frontier in LHC experimental data analysis and opens new doors for the use and development of machine

learning techniques in high energy physics. A number of physics analysis have the potential to significantly benefit from using SBI techniques and the computing workflow as presented in this proceeding can be easily adapted to achieve the same. The parameter fitting workflow of the SBI analysis, described in Sec. 5, leverages JAX and JIT compilation for accelerated computation of profile likelihood fit, offering substantial potential for reducing computational time in both SBI and traditional histogram-based analyses.

Broadly speaking, applications needing extensive GPU and/or advanced CPU setups can take advantage of the readily available and scalable nature of cloud infrastructure. Although High-Performance Computing (HPC) systems are incorporated into the ATLAS computing grid, providing a feasible option, the cloud presents a wider range of available hardware, which can greatly reduce the initiation period for scientific research with new resources. Further exploration of such applications utilizing cloud-based infrastructure in the ATLAS experiment is discussed in Ref. [10].

References

[1] The ATLAS Collaboration, JINST **3** S08003 (2008)

[2] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, Phys. Rev. D **98** 052004 (2018)

[3] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, Phys. Rev. L **121** 111801 (2018)

[4] F. Barreiro Megino et al., J. Phys. Conf. Ser **898** 052002 (2017)

[5] M. Barisits et al., Comput. Softw. Big Sci. **3** 11 (2019)

[6] *WLCG homepage*. URL: https://wlcg.web.cern.ch/ (visited on 07/14/2023)

[7] F. Barreiro Megino et al., EPJ Web Conf. **251** 02005 (2021)

[8] *Kubernetes Documentation*. URL: https://kubernetes.io/docs/home/ (visited on 07/14/2023)

[9] T. Maeno et al., EPJ Web Conf. **214** 03030 (2019)

[10] F. H. Barreiro Megino et al. *Accelerating science: the usage of commercial clouds in ATLAS Distributed Computing*, these proceedings. (2023)

[11] M. Barisits et al. *Extending Rucio with modern cloud storage support: Experiences from ATLAS, SKA and ESCAPE*, these proceedings. (2023)

[12] D. Merkel, Linux J. **2014** 2 (2014)

[13] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. (Visited on 07/14/2023)

[14] J. Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax (visited on 07/14/2023)

[15] H. Dembinski et al. scikit-hep/iminuit. Dec. 2020. URL: https://doi.org/10.5281/zenodo.3949207 (visited on 07/14/2023)

[16] F. James and M. Roos, Computer Physics Communications **10** 343–367 (1975)