GPU coprocessors as a service for deep learning inference in high energy physics

GPU coprocessors as a service for deep learning inference in high energy physics

IOP Publishing Mach. Learn.: Sci. Technol. 2 (2021) 035005 https://doi.org/10.1088/2632-2153/abec21

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
21 July 2020

REVISED
29 January 2021

ACCEPTED FOR PUBLICATION
4 March 2021

PUBLISHED
23 April 2021

PAPER

GPU coprocessors as a service for deep learning inference in high energy physics

Jeffrey Krupa[1], Kelvin Lin[2], Maria Acosta Flechas[3], Jack Dinsmore[1], Javier Duarte[4], Philip Harris[1], Scott Hauck[2], Burt Holzman[3], Shih-Chieh Hsu[2], Thomas Klijnsma[3], Mia Liu[3], Kevin Pedro[3], Dylan Rankin[1], Natchanon Suaysom[2], Matt Trahms[2] and Nhan Tran[3,5]

[1] Massachusetts Institute of Technology, Cambridge, MA 02139, United States of America
[2] University of Washington, Seattle, WA 98195, United States of America
[3] Fermi National Accelerator Laboratory, Batavia, IL 60510, United States of America
[4] University of California San Diego, La Jolla, CA 92093, United States of America
[5] Northwestern University, Evanston, IL 60208, United States of America

E-mail: pcharris@mit.edu

Keywords: LHC, HEP, GPU, GPUaaS, deep learning, coprocessor, HPC

Abstract

In the next decade, the demands for computing in large scientific experiments are expected to grow tremendously. During the same time period, CPU performance increases will be limited. At the CERN Large Hadron Collider (LHC), these two issues will confront one another as the collider is upgraded for high luminosity running. Alternative processors such as graphics processing units (GPUs) can resolve this confrontation provided that algorithms can be sufficiently accelerated. In many cases, algorithmic speedups are found to be largest through the adoption of deep learning algorithms. We present a comprehensive exploration of the use of GPU-based hardware acceleration for deep learning inference within the data reconstruction workflow of high energy physics. We present several realistic examples and discuss a strategy for the seamless integration of coprocessors so that the LHC can maintain, if not exceed, its current performance throughout its running.

1. Introduction

The detectors at the CERN Large Hadron Collider (LHC) [1] have enormous data rates, with a current aggregate data rate of 100 Tb s^{-1} and plans to exceed over 1 Pb s^{-1}. The challenge of processing this data continues to be one of the most critical elements in the execution of the LHC physics program [2–6]. A three-tiered approach is utilized to process LHC data, where at each tier, the data rate is reduced by roughly two orders of magnitude, resulting in a manageable final data rate of 10 Gb s^{-1}. Due to the high initial rate and restrictions coming from the high radiation collision environment, the first tier of computing consists of specialized hardware that utilizes field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs). The second tier, the high-level trigger (HLT), consists of a CPU-based computing cluster on-site at the LHC. The first two tiers are described as 'online' computing, because they occur in real time, as each LHC collision is measured by the detector. The third tier, performing complete event processing, consists of a globally distributed CPU-based computing grid. This third tier is described as 'offline' computing, because it occurs after the initial collision data has already been written to disk. Both the online and offline computing tiers run a similar set of algorithms, but the HLT employs certain approximations in order to satisfy the online latency budget.

The first decade of LHC running has led to an extensive set of scientific results. These results include the discovery of the Higgs boson [7–9] and, more recently, strong constraints on the nature of dark matter [10–12]. To contend with these strong dark matter constraints, physicists have been forced to re-think their approach to searching for dark matter and, generically, new physics models. This has led to the development of light dark matter models [13]. These models often predict signatures that could be produced at the LHC but would be discarded in the early tiers of data reduction. To enable the search for these particles,

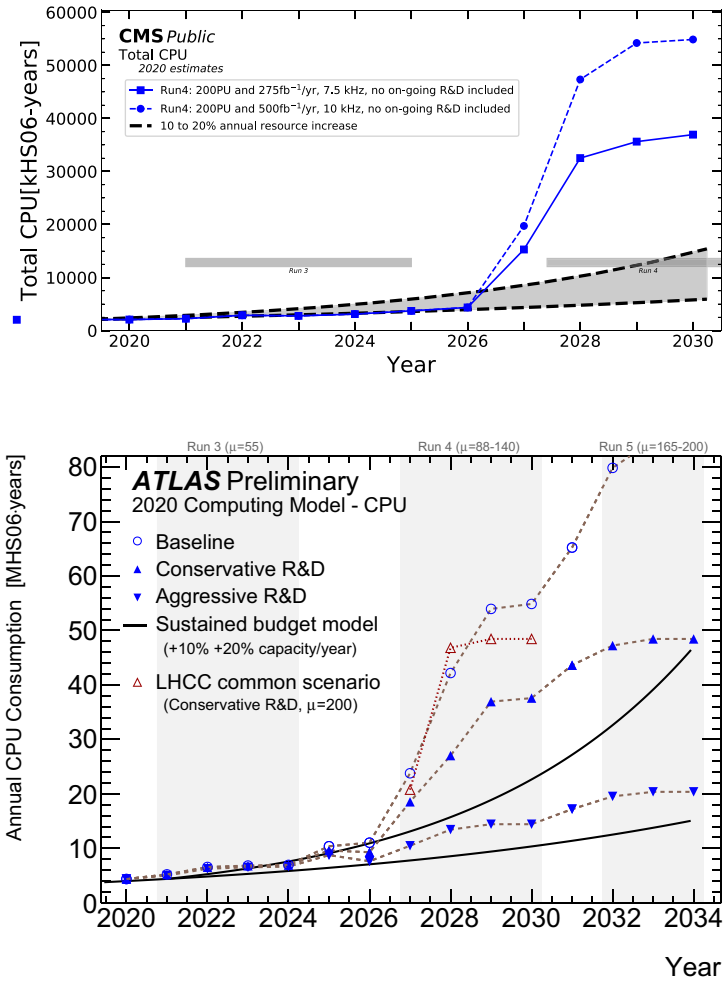

Figure 1. Computing needs projections for the CMS (upper) and ATLAS (lower) experiments at the LHC [14–16]. For ATLAS, the computing needs estimate is shown under the baseline computing model and under different R&D scenarios, while for CMS, the estimate is shown without including ongoing R&D in different collider conditions. The black solid lines show the range of expected available CPU, assuming CPU performance increases of 10%–20% per year. MHS06-years (kHS06-years) stands for 10^6 (10^3) HEPSPEC06 per year, a standard CPU performance metric for high energy physics (HEP) [17]. The different LHC operating runs are also indicated. A large upgrade to the collider and all LHC detectors will occur between Run 3 and Run 4, starting in 2026.

it is imperative to increase the probability that these particle signatures are not discarded by the data reduction. This can be done by improving the quality of LHC data reconstruction at all tiers of processing. Additionally, over the next decade, the LHC will progressively increase the beam intensity, resulting in more data recorded by the detectors [4]. As a consequence, the demands for computing will increase proportionally to sustain the current level of physics output. Figure 1 shows the expected computing needs over the next decade. These expectations arise from modeling by the two general-purpose particle detectors at the LHC: the Compact Muon Solenoid (CMS) and ATLAS experiments. To contend with the high-luminosity upgrade of the LHC (HL-LHC), a large increase in computing capacity is needed starting from 2026. These demands outpace the expected growth of CPU performance. As a consequence, the LHC needs a computing solution at least to sustain the current computing performance and, potentially, to exceed it.

With the end of Dennard scaling [18] in the late 2000s, processor technology has undergone several changes [19]. These changes have included the adoption of multicore processors and the rise of alternative processing architectures, or coprocessors, such as graphics processing units (GPUs), FPGAs, and ASICs. With the rise of deep learning (DL), these alternatives have become increasingly appealing due to the inherent parallelism in both DL algorithms and in these coprocessors. The gains from using coprocessors can be substantial, with improvements in inference latency for large algorithms exceeding multiple orders of magnitude [20]. Given the scale of developments related to DL, future growth in processor technology is increasingly leaning towards heterogeneous systems in which combinations of CPUs, GPUs, FPGAs, and ASICs are all deployed, with each designed to solve specific tasks. However, HEP experiments have thus far undertaken only limited use of alternative processors within the HLT and offline computing grids, despite common use of machine learning (ML). HEP experiments have historically relied on ML as a way to improve

the overall quality of the data and to separate small signals from enormous backgrounds, such as the discovery of the Higgs boson [21]. DL approaches have enhanced both the performance and flexibility of ML techniques. In light of this, the LHC experiments have been quick to adopt DL techniques to improve the quality of data analysis especially during Run 2 (2015–2018) [21–24]. This includes core components such as low-level detector energy reconstruction [25], electron and photon reconstruction [26], and quark and gluon identification [27–29]. The increasing deployment of these algorithms is starting to comprise a significant portion of the overall computing budget: the full event reconstruction takes tens of seconds per event on modern CPUs [30], while large DL algorithms may require seconds per inference. The goal of this study is to enable the use of these algorithms in online and offline data processing tiers, in the context of the LHC experiments' increasing data rates. Our approach does this by offloading the computational burden of these algorithms to GPUs while making minimal changes to existing CPU-based workflows.

To achieve this, we move existing work a step further by exploiting the 'as-a-service' paradigm, in which user access to applications running on remote cloud infrastructure is provided through a thin client interface [31]. In this paper, we design a prototypical framework for LHC computing as a service. We apply DL algorithms to replace domain-specific algorithms to solve a variety of physics problems through DL inference. We then transfer the algorithm to a coprocessor on an independent (local or remote) server and re-configure the existing CPU nodes to communicate with this server through asynchronous and non-blocking inference requests. With the inference task offloaded as a request to the server, the CPU is free to perform the rest of the necessary computing within the event.

Deploying GPUs as a service (GPUaaS) is a natural way to incorporate alternative coprocessors that has several advantages over a direct-connection approach. In particular, deploying GPUaaS increases hardware cost-effectiveness by reducing the number of GPUs required to achieve the same throughput. This is possible because each GPU can service many more CPUs than a direct-connection paradigm would allow. It is nondisruptive to the existing LHC computing model by offloading the specific algorithms with minimal client-side re-configuration (see section 3). It facilitates seamless integration and scalability of heterogeneous coprocessors (such as GPUs and FPGAs), as suited for optimal algorithmic performance. Finally, by exploiting existing open-source, widely-adopted frameworks that have been optimized for fast GPU-based DL, this approach can be adapted quickly to different tasks at the LHC and beyond.

In this paper, we present several examples of integrating GPUaaS into LHC workflows. We consider three DL-based algorithms that span a variety of LHC computing applications. We integrate these algorithms into both online and offline LHC workflows with GPUaaS and we benchmark them to evaluate the impact of GPUaaS on the operation of the HLT and the offline computing grid. With the goal of optimizing throughput in the high-rate LHC computing environment, we focus on accelerating model inference on coprocessors, as opposed to training. Based on our results, we propose a model for incorporating GPUs and other coprocessors into LHC computing workflows.

The remainder of this paper is organized as follows. In section 2, we briefly review related work. In section 3, we provide an overview of the current LHC computing model, the as-a-service computing model, and we derive metrics that quantify the cost-effectiveness of coprocessors in LHC workflows. Section 4 describes the three ML-based algorithms to be deployed. We describe our configuration of the servers in Google Cloud and LHC data centers and evaluate the limitations of each site in section 5. We also measure several performance-related quantities relevant for full-scale LHC reconstruction as a service, including hardware throughput, network bottlenecks, and scaling with number of GPUs. In section 6, we determine the hardware and networking requirements for maximizing the throughput of these algorithms at scale for LHC computing as a service. Finally, we conclude in section 7.

2. Related work

Researchers across HEP have investigated the use of GPUs in event reconstruction in a variety of ways. At the LHC, the focus has been on implementations for the HLT where faster computing times lead directly to increased throughput. For offline computing at the LHC, GPUs have not been pursued since previous usage of GPUs, through a direct connection to the CPU, would require a larger redesign of the LHC computing grid. This work avoids the need for directly connected GPUs by employing GPUaaS, which provides a method to allow GPUs to be run remotely. GPUaaS enables GPU use in both the HLT and offline workflows without a large redesign of the existing LHC computing grid. Additionally, we utilize DL algorithms, which allow for the use of existing GPU compiler frameworks to quickly obtain optimized code.

To perform data analysis on reconstructed objects, DL algorithms are extensively used in HEP. In this context, training of DL algorithms is almost exclusively performed on GPUs. Frameworks such as GooFit [32], pyhf [33], and hepaccelerate [34] exploit GPU acceleration for HEP maximum likelihood fits and data processing applications.

Within the context of online computing, GPUs were first integrated into the 180 compute nodes of the HLT workflow of the ALICE experiment at the LHC to perform charged particle reconstruction. They were part of the ALICE operations during 2010–2013 and 2015–2018 [35]. More recently, GPUs are being considered for the HLT of the LHCb experiment [36, 37]. Within the CMS experiment, GPUs have been explored for reconstruction of tracks with the pixel detector [38] and charged particle reconstruction, through the use of cellular automata [39], and through the use of an accelerated pattern recognition algorithm [40]. Furthermore, algorithms for GPUs and FPGAs have been developed for real time processing of ring imaging Cherenkov detectors for the LHCb HLT [41, 42]. Beyond the LHC experiments, GPU algorithms have been developed for the trigger readout of the Mu3e experiment [43], and the dark matter experiment NA62 [44]. These algorithms are planned to run in the next round of data taking for each experiment, starting in 2021. In all instances, GPUs have been considered in the context of direct connection to CPUs via PCI Express.

Within the context of offline computing, GPU use in HEP has remained limited. In neutrino physics, GPUs have been used for simulation of the propagation of Cherenkov light signatures for the IceCube experiment [45]. The experiment recently performed a large-scale test of a GPU-only simulation of neutrino signatures, using over 50 000 GPU cores for a period of 20 min [45, 46]. In this scenario, a large number of cores are used to accelerate a certain component of IceCube simulation.

The offline and online reconstruction software for large LHC experiments consists of several million lines of CPU code. Rewriting this code to run on GPUs, for example using CUDA, would be prohibitively costly. In some cases, such as non-parallelizable or transfer-limited operations, it would likely lead to substantially worse timing performance. In this paper, we present, for the first time, an alternative model whereby only algorithms with substantial speedups are ported to GPUs, with each GPU serving many CPU nodes. We demonstrate that GPUaaS can be integrated within full LHC workflows and can produce significant overall algorithmic speed improvements. A similar model for the utilization of CPUs and FPGAs within the LHC workflow was presented in [20] using the Services for Optimized Network Inference on Coprocessors (SONIC) framework [47]. The study exploited the Microsoft Brainwave service [48] and demonstrated a decrease in deep neural network (DNN) inference time by nearly 3 orders of magnitude when using an FPGA compared to a CPU. This paper extends SONIC to support GPUaaS, demonstrating a viable model for fast and nondisruptive integration of GPUs into the LHC workflow. Outside of the SONIC work described above and recent upgrades in the ALICE software stack [49], computing as a service has not previously been pursued in HEP.

3. As-a-service computing for LHC physics

The current LHC computing model is shown in figure 2. In typical LHC event reconstruction, data is processed sequentially event-by-event, possibly on multiple threads on the CPU. However, if certain algorithms are significantly accelerated by the use of coprocessors (as shown in [20]), a modified scenario with coprocessing as a service can be considered. In this model, a single coprocessor can serve hundreds of CPU processing elements. The CPUs are executing numerous different algorithms of the full event reconstruction, whereas the inference server is executing a single algorithm very efficiently. To benefit from this type of computing model, there must be a sufficiently large acceleration such that the overhead of offloading this processing onto a separate server does not further increase the reconstruction latency. To explain when this is the case, we first review the reconstruction model at the LHC and then discuss how as-a-service computing can be implemented within the LHC reconstruction workflow.

3.1. LHC reconstruction

Detectors at the LHC are general-purpose devices with millions of channels, each of which records information from particles passing through or decaying within it. Event reconstruction involves combining these individual signals from different detector channels as optimally as possible to form the set of observable particles, including their energy, momentum, and type, for each event. This collection of particles is then used to infer the underlying physics process. For example, an event containing a Higgs boson decaying to a bottom quark-antiquark pair can lead to roughly 100 particles and we can use the aggregate properties of these particles to infer the presence of a Higgs boson. The variety of particles with different signatures in each detector (physics objects) that may be present in any given collision leads to a large number of different reconstruction algorithms that must be run on each event as each physics object typically has its own reconstruction algorithm. This, in turn, leads to a large codebase that is written entirely for CPUs.

Parallelization of the reconstruction algorithms that create particles is possible by splitting the reconstruction into separate geometric regions and reconstructing the individual particles within that

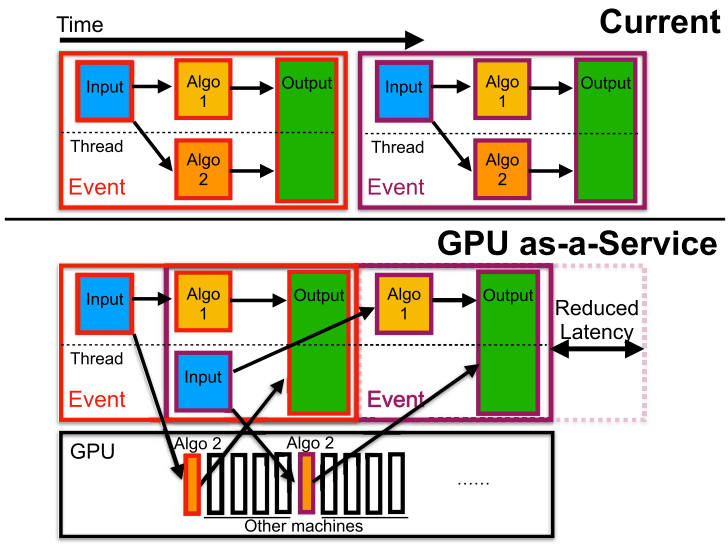

Figure 2. Diagram comparing the traditional LHC production model on CPUs (upper) with the GPUaaS approach (lower). Each block represents a module within the reconstruction framework. For the GPUaaS approach, algorithm 2 is run on the GPU, which allows the processing of the second event (outlined in purple) to run concurrently with the first event (outlined in red).

region. Further parallelization is possible through the separate reconstruction within the individual detectors before they are aggregated into particles. The current reconstruction aims to exploit possible parallelization by compartmentalizing separate reconstruction algorithms into modules that can be run in parallel. No single algorithm dominates the overall computing time, but some fundamental tasks, such as tracking and clustering detector hits to form particles, are the most computationally intensive. The potential for parallelization has only partly been realized through standard CPU optimizations, such as auto-vectorization. In this work, large-scale batching of the reconstruction to allow for algorithm-level parallelism is achieved through the use of DL algorithms on the GPU.

3.2. As-a-service computing

To apply DL under the as-a-service paradigm, we choose an algorithm that has a significant speedup when using a GPU. We then take this algorithm and set up a GPU inference server using the NVIDIA Triton Inference Server [50]. This package uses a custom gRPC-based communication protocol [51], and it supports load-balancing between multiple GPUs hosted together in a single server. By default, the server queues and processes single-event inference requests with a batch size defined by the client. In some cases, the throughput can be increased by aggregating requests from multiple single-event calls, as discussed in section 5.3.1. Inference requests can be made for models from various ML frameworks, and multiple models can be loaded on the same server. Software frameworks in HEP are typically written in C++. The software framework for the CMS experiment, CMSSW [52], uses task-based multithreading enabled by tbb [53]. This facilitates asynchronous, non-blocking calls to external resources using a feature called ExternalWork [54]. This is the most efficient way to utilize coprocessors as a service because the CPU running the experiment software can perform other tasks while the service call finishes. For this paper, we have taken advantage of these features by extending the CMSSW version of the SONIC software to perform remote gRPC calls to GPUs via the Triton Inference Server. In the SONIC approach, only the client code needs to be provided in the software framework. This minimizes the maintenance burden, as the client code has just two responsibilities: converting between the experiment and server data formats, and making the call to the server via the chosen communication protocol. All the details of the model architecture, any optimizations, and even the choice of coprocessors can be decided on the server without any change in the client. This setup enables the modified reconstruction workflow depicted in figure 2.

By extending the SONIC framework to handle the gRPC calls utilized by the NVIDIA Triton Inference Server, the new client code uses a standard interface such that the user-developed software to convert between experiment and server data formats remains independent. Beyond the specification of the remote server protocol and location within a global configuration file, the user code remains completely intact, and switching between the remote FPGA calls, remote GPU calls, and other local calls are done seamlessly through a configuration file.

3.3. Metrics for optimization

To determine the cost-effectiveness of deploying a given algorithm as a service, we compose a simplified heuristic. We assume a computing model similar to those used by LHC experiments, which schedules modules to run during the service request, as in figure 2. We introduce the GPU-to-CPU replacement ratio $F_{\mathrm{GPU}}^{\mathrm{eq}}$ to maintain the same throughput:

$$F_{\mathrm{GPU}}^{\mathrm{eq}} = \frac{X - S - L}{Y}, \tag{1}$$

where X is the algorithm processing time on the CPU, S is the overhead time due to input/output packaging in SONIC, and $L = f(Y + T)$ is the rescheduling time as a function of the processing time on the GPU Y (which depends on the algorithm, hardware, and batch size), and the packet transfer time T. For instance, a value of $F_{\mathrm{GPU}}^{\mathrm{eq}} = 32$ implies that one GPU can replace 32 CPUs at no cost in the overall event throughput. While the number of CPUs required to achieve the same throughput decreases due to algorithmic speedups on GPUs, a baseline CPU farm will always be required to perform core event processing and make calls to the GPU. A pre-existing CPU farm can serve this baseline requirement. The optimal value of $F_{\mathrm{GPU}}^{\mathrm{eq}}$ depends on the demands of the system design, as well as the algorithm- and software-dependent values for both S and L. Time spent in data transfer or queuing on the server plays a small role in total throughput because of the asynchronous, non-blocking call employed in SONIC.

We use $F_{\mathrm{GPU}}^{\mathrm{eq}}$ as a guide to contextualize our results for GPU acceleration for each of the different scenarios studied. It is derived provided that no substantial bottlenecks are present in the software infrastructure, and further studies will refine this model. In the following sections, we explore the GPU speedups utilizing the SONIC framework for algorithms with various $F_{\mathrm{GPU}}^{\mathrm{eq}}$ values. We discuss the discovered bottlenecks and present a path towards a realistic implementation of GPUaaS at the LHC. An interesting potential extension of these studies would be to systematically investigate the relationship between $F_{\mathrm{GPU}}^{\mathrm{eq}}$ and throughput and latency.

4. Algorithms

To investigate the scalability of deploying DL as a service for LHC experiments, we study three distinct algorithms. Together, these algorithms span LHC computing, from low-level tasks of local detector energy reconstruction to high-level tasks of offline object identification. They also exhibit a range of speedups on coprocessors. Each algorithm performs as well as a CPU reference algorithm at resolving physical quantities. We then accelerate these algorithms with GPUaaS in realistic LHC workflows. We report the results of these tests in the next sections.

While the emphasis in this paper is on DL algorithms because optimized GPU implementations already exist, many LHC algorithms are currently not ML-based and likely will remain that way in the future. Nonetheless, many of these tasks have been shown to benefit significantly in computational performance if deployed on coprocessors with custom implementations [55]. The technology we develop for the ML algorithms as a service is flexible and its extension to non-ML algorithms is straightforward.

4.1. Hadron calorimeter reconstruction

The simplest algorithm that we study is called Fast Calorimeter Learning (FACILE), a deep neural network consisting of 2000 parameters. This algorithm was trained on a CPU using simulated collisions at the LHC using generator-level information to reconstruct the energy deposited by particles in each cell of the CMS hadron calorimeter (HCAL). FACILE uses 15 inputs that contain information about raw charge collected by detector hardware, coordinates of the HCAL channel, and gain of the HCAL channel. FACILE consists of batch normalization [56] and dense layers with rectified linear unit (ReLU) activation functions [57, 58]. The layers consist of 30, 20, 10, 5, and 3 neurons each, respectively. The authors trained the network using a sample size exceeding 1 million HCAL channel events separated into training and testing datasets. A mean squared error loss function, batch size of 5000, and the Adam optimizer [59] were used in training.

The HCAL is a core component of LHC experiments and a prototypical subdetector for which to implement ML as-a-service reconstruction for several reasons. First, good resolution in the HCAL is important for sensitive measurements in particle physics, such as events with a Higgs boson decaying to bottom quarks. We find that local (e.g. HCAL channel energies) and global (e.g. jets and missing transverse momenta) physics objects reconstructed with FACILE have similar resolution to objects reconstructed with the nominal algorithm that does not use ML [60]. Second, the nominal HCAL reconstruction algorithm in CMS requires $X = 60$ ms of CPU time, accounting for approximately 15% of the online computing budget [61]. FACILE offers a significant improvement in computing performance when operated as a service by reducing the CPU time (approximating $S + L$) to less than 7 ms, resulting in an estimated $F_{\mathrm{GPU}}^{\mathrm{eq}} = 27$, with

Table 1. Summary of the specifications of each algorithm, including model parameters, GPU memory usage, and GPU utilization. The memory usage and GPU utilization are quoted at the point of maximum GPU throughput. For FACILE and DeepCalo, the quantities correspond to an NVIDIA V100 GPU, while for ResNet-50, the quantities are quoted for an NVIDIA T4 GPU.

Algorithm	Batch size	Architecture type	Trainable parameters	Number of layers	GFLOP per batch	GPU memory usage (GB)	GPU utilization (%)
FACILE	16 k	Dense	2 k	5	0.032	1	20
DeepCalo	5	Convolutional	2 M	13	0.43	2.6	40
ResNet-50	10	Convolutional	23 M	50	39	12	95

respect to the nominal algorithm on CPU, which we verify experimentally. The remote time (including $Y = 2$ ms of GPU latency) is reduced by the asynchronous, non-blocking ExternalWork feature employed by SONIC. Finally, by exploiting GPU performance for large batch sizes, FACILE offers enhanced physics potential by reconstructing all 16 000 HCAL channels in parallel with little added latency, instead of reconstructing only the highest energy channels.

In terms of physics and computational performance, FACILE is well-suited for both online and offline applications. We test it in both settings. For instance, we perform a high-bandwidth test designed to emulate, for the first time at scale, a realistic LHC online computing system with coprocessors as a service.

4.2. Electron regression

DeepCalo is a midsize convolutional neural network (CNN) trained for electron energy regression for the ATLAS detector [62]. It operates at a higher, more abstract level compared to FACILE since it reconstructs the energy from an entire region of a calorimeter subdetector. Compared to nominal techniques [63, 64], DeepCalo improves energy resolution and robustness against pileup, both of which are important for the HL-LHC [62]. The model is trained on electrons reconstructed from a Monte Carlo simulation of collisions spanning a wide range of energies. Each collision deposits energy in the electromagnetic calorimeter (ECAL) cells. These energy deposits are encoded as a 56×11 pixel image with four channels that represents a 2D patch of the detector of width 0.175 in η and 0.270 in ϕ. The four channels represent four separate layers of the ECAL and each pixel value represents the amount of energy deposited at that location and in that layer in η and ϕ. Using these images, DeepCalo estimates the energy of the electron.

DeepCalo is composed of 1.8 million parameters. The first component of the CNN consists of five convolutional blocks. The first block performs a 5×5 convolution, followed by batch normalization and a leaky ReLU activation function [65]. Each subsequent convolutional block performs a 2×2 maximum pooling, followed by two instances of a sub-block consisting of a 3×3 convolution, batch normalization, and a leaky ReLU activation function. The final component of DeepCalo consists of three fully-connected layers, with the last layer producing a prediction for the electron energy.

In this study, we deploy DeepCalo as a service on GPU coprocessors for offline reconstruction. In the offline test, we maximize the event throughput and compare the performance to on-site, CPU-based implementations. When deployed as a service, DeepCalo shows significant performance gains by reducing the latency per event from 75 to 1.5 ms and, when optimized with dynamic batching (as described in section 5.3.1), to 0.1 ms. This yields an estimated $F_{\mathrm{GPU}}^{\mathrm{eq}} = 50$ nominally and 750 after optimization.

4.3. Top quark tagging

ResNet-50 [66] is a CNN composed of 23 million parameters, 49 convolutional layers of 7×7, 3×3, and 1×1 convolutions with 'skip connections,' and 1 fully-connected layer, which predicts 1000 class probabilities for natural images. In earlier studies [20], the ResNet-50 CNN architecture was re-purposed to identify events containing top quarks (top quark tagging). In addition, CMS has implemented similar CNN-based top quark tagging algorithms for offline reconstructions [67]. Another study [68] showed that ResNet-50 could be modified to perform top quark tagging with performance rivaling leading ML algorithms. Of the three algorithms, ResNet-50 is the most complex and has the longest latency on GPU, and we estimate $F_{\mathrm{GPU}}^{\mathrm{eq}} = 150$. We choose it to enable benchmarking of a CPU-prohibitive algorithm as a service. In particular, ResNet-50 has a CPU latency of the order of seconds, which is prohibitively high for use even in offline reconstruction scenarios.

In [20], we observed a speedup by orders of magnitude by deploying ResNet-50 as a service on FPGA coprocessors. In this study, we extend our earlier studies by deploying ResNet-50 as a service on GPU coprocessors in LHC workflows. This enables top quark tagging to be performed in offline reconstruction. ResNet-50 also serves as a prototypical large benchmark algorithm comparable in burden to other major tasks in LHC computing, such as tracking. The specifications and GPU utilization of the three algorithms are summarized in table 1.

5. GPU performance studies

For online computing, we integrate FACILE into the HLT, the second tier of CMS data acquisition. For offline computing, we consider all three algorithms in stand-alone workflows. The client implementation is based on SONIC in a separate CMSSW fork [47]. We quantify the hardware and networking requirements to run these algorithms as a service in LHC computing. To achieve this, we measure the coprocessor throughput (in events processed per second), the number of servers and GPUs required to service a given number of clients (or simultaneous processes), and the network bandwidth limitations (arising from on-premises and external sources) in both LHC computing clusters and on the Google Cloud Platform.

We focus on achieving a hardware-efficient deployment of the algorithms by monitoring server properties and GPU utilization. We also measure how throughput scales with the number of GPUs by deploying many GPUs on a single server with a customized Google Kubernetes engine setup (as described in section 5.3). Finally, we investigate various optimizations for the GPUs to further increase the throughput. Ultimately, in section 6, we apply our findings to determine the hardware and networking requirements to perform full-scale LHC computing with coprocessors as a service.

5.1. Online computing

To study the use of coprocessors as a service in online computing, we run the full CMS HLT with local HCAL reconstruction performed by FACILE as a service. FACILE is particularly well-suited for an online computing application because the algorithm it replaces is responsible for 15% of the HLT latency per event. In this study, the clients are deployed as jobs running single-thread HLT instances on virtual machines in Google Cloud using the HEPCloud framework [69–71]. HEPCloud deploys jobs submitted on batch systems to CPU instances created dynamically at a cloud computing site. The jobs are synchronized by adding a waiting period such that each job begins processing information only when all jobs are ready. This ensures that all jobs send calls to the GPU server during the same time period, enabling an accurate measurement of GPU and network throughput. Since FACILE has a small GPU latency (2 ms) compared to the HLT (500 ms), it proved essential to run on CPUs absent of other jobs for a realistic emulation of the current system of dedicated HLT cores. The cloud enabled this by reducing systematic uncertainties arising from shared CPUs on-premises. The server was deployed at the same site and consisted of a Google compute instance with either one or four NVIDIA Tesla V100 GPUs. This client-server configuration realistically emulates a fraction of the dedicated HLT CPU farm at CERN with the addition of as-a-service computing.

The results of this test are shown in figure 3. Each client is allotted 7000 simulated LHC benchmark timing events. The timing distribution for the HLT running FACILE as a service is shown in the top panel for servers with one or four GPUs in red and blue violins. For the HLT tests, FACILE is operated at 4500 batch size per event; however, the algorithm operates at similar latency with the full HCAL detector (batch size 16 000). The average time to run the nominal HLT algorithm locally on the CPU is shown in a dotted black line. For fewer than 500 clients, a decrease of approximately 10% in the total time is observed with FACILE as a service with one GPU when compared to the nominal algorithm. This largely eliminates the CPU burden of HCAL reconstruction. Since the HLT farm at CERN operates under latency restrictions, this demonstrates an opportunity to increase the throughput of the current trigger system by 10%, or alternatively, partitioning 10% of the existing machines to be used for other tasks. An increase in aggregate HLT latency occurs only above 300 clients for a single GPU, and above 1000 clients for four GPUs. This increase represents the point where GPU throughput limitations begin to dominate, indicating that at least 300 HLT instances can be serviced by a single GPU without penalty. This slightly exceeds our expectation of 180 HLT instances, based on our computation of $F_{\mathrm{GPU}}^{\mathrm{eq}} = 27$ divided by the 15% CPU time fraction, but confirms the overall scaling. We attribute this overperformance to the fact that the actual number of HLT calls for the algorithm is less than the rate at which the algorithm is run. As a result, we conclude that operating reconstruction as a service is more efficient than having GPUs directly connected to CPUs, since more than 32 cores can be serviced by a single GPU. We explore this further by describing a scale design in section 6. We note that the long tails in the figure are caused by scheduler assignments where fewer jobs are run on certain machines, leading to improved throughput for a small number of jobs, but a negligible effect in overall throughput.

The HLT throughput with FACILE as a service is shown in the bottom panel of figure 3. For the single GPU server (red circles), the throughput starts demonstrating asymptotic behavior above 300 simultaneous processes, while for the four GPU server (blue triangles), it does not yet asymptote even up to 1000 simultaneous processes.

5.2. Offline reconstruction

In the offline computing scenario, a single GPU service can be used by several remote computing clusters at the same time as depicted in figure 4. We investigate the use of FACILE, DeepCalo, and ResNet-50 for LHC

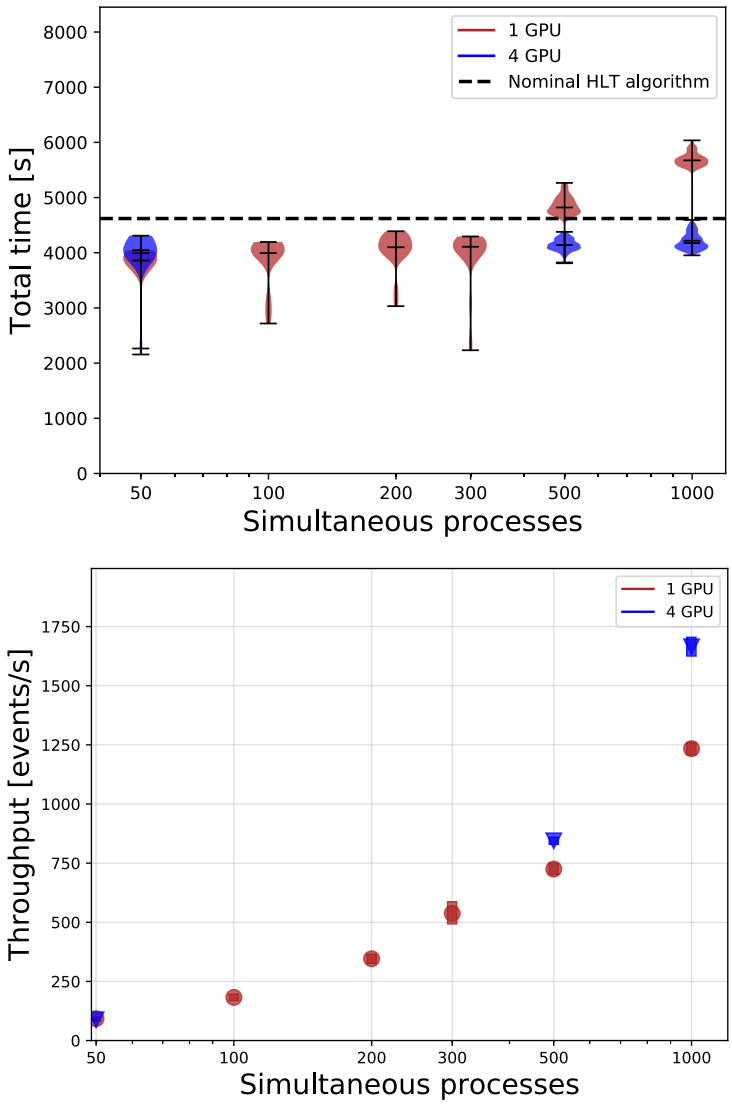

Figure 3. The distribution of the total time (including median and whiskers) to run the high level trigger with the HCAL reconstruction performed with FACILE as a service (upper). Servers with one and four GPUs are shown as red and blue violins. The average time taken to process the same events using the nominal HCAL reconstruction is shown as a dotted black line. High level trigger throughput running FACILE for servers with one and four GPUs in red and blue markers, respectively (lower). Data between 100–300 simultaneous processes are omitted for the four GPU server.

offline computing by executing a dedicated workflow process for each model. Given the approximately 150 000 CPU cores available for a single LHC experiment and that event reconstruction times are on the order of 30 s per event, our tests assume a benchmark LHC computing throughput of 5000 events per second, and we estimate the coprocessors necessary to attain this. The processing of each model includes realistic input, formatting, and output steps. To emulate a realistic global offline computing scenario with CPU workers, as shown in figure 4, we deploy clients to CPU clusters at MIT and Fermilab. These CPUs send gRPC requests over the internet to servers in Google Cloud's `us-central1-a` zone in Council Bluffs, Iowa. While not shown here, we repeated these same tests on-premises going from on-site CPUs to the GPU with Google Cloud and we observed nearly identical throughput saturation and networking effects to the tests observed when going from a remote location to the same GPU within Google Cloud. This implies that communication over distance is reliable at the network bandwidths of interest.

5.2.1. FACILE

The throughput of FACILE as a service is shown for different numbers of clients and GPUs in figure 5. A server with a single GPU is found to saturate at a throughput of 500 inferences per second for a V100 GPU. This limit is due to the hardware latency and occurs above 50 clients. The test is operated at a batch size of 16 000 per event, providing a conservative assumption of HCAL reconstruction requirements. The V100 GPU is used because it offers a 10%–20% gain over other GPU models.

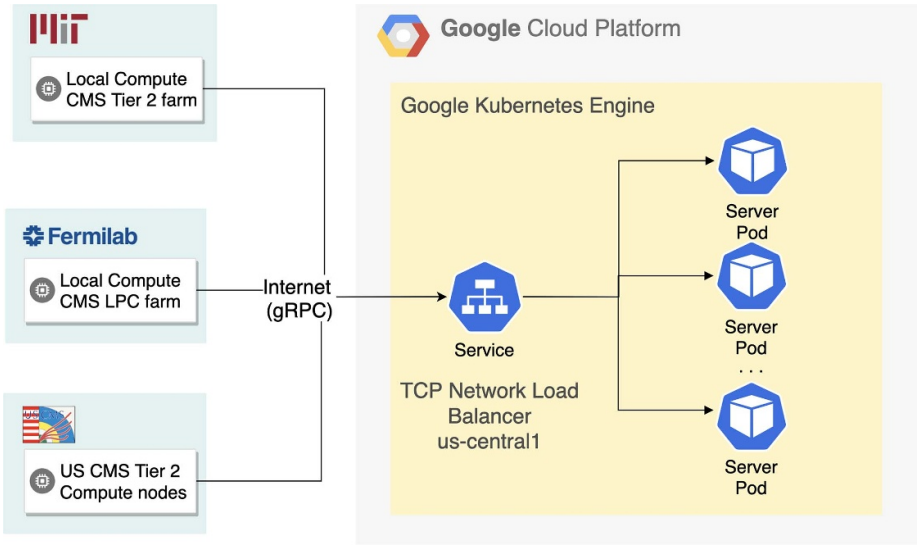

Figure 4. Diagram of the architecture for large-scale processing using GPU as a service. In this scenario, a GPU server within Google Cloud (right) is used to serve many offline computing centers processing LHC data (left). The calls are sent remotely over the internet as gRPC requests.

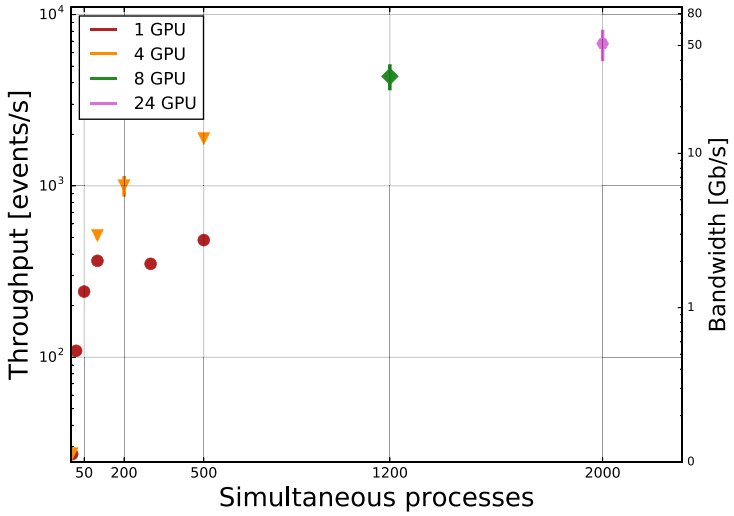

Figure 5. The throughput of FACILE operated as a service in events per second versus the number of simultaneous clients. The marker indicates the mean and the error bars indicate the standard deviation. The throughput tests are repeated with 1, 4, 8, and 24 GPUs connected to the server, represented by red circles, orange triangles, green diamonds, and pink hexagons, respectively. The network bandwidth from the transfer of inputs from the client to the server is shown on the right vertical axis.

As we increase the number of GPUs on the server using a customized Google Kubernetes Engine setup (see section 5.3.2, we find that the throughput scales linearly and with high efficiency. Servers with four and eight GPUs saturate at approximately 2000 and 4000 inferences per second, as shown in figure 5, respectively. Therefore, the LHC throughput requirement can be satisfied by a single 10 GPU server. This indicates that the Google Kubernetes Engine employed here is an efficient way to increase the throughput. The 24 GPU server test with 2000 clients, in pink, is designed to probe the limit on network bandwidth between the LHC clusters and Google Cloud. This test becomes limited by a network bottleneck of unknown origin and we observe a peak bandwidth exceeding 70 Gb s^{-1}. The number of clients at which saturation occurs also scales with the number of GPUs; for example, the 4 GPU server in orange does not saturate until nearly 500 clients. We note that we were not able to plot out the entire throughput distributions due to the expense of each test.

The throughput is highly sensitive to the server configuration. Initially, we deployed a server with a single four CPU ingress node handing off the request to nodes with GPUs. These tests proved to be limited to a throughput of 1500 inferences per second (12 Gb s^{-1}) regardless of client number, indicating there was a bandwidth limitation at the destination rather than between MIT and Google Cloud. As a result, we

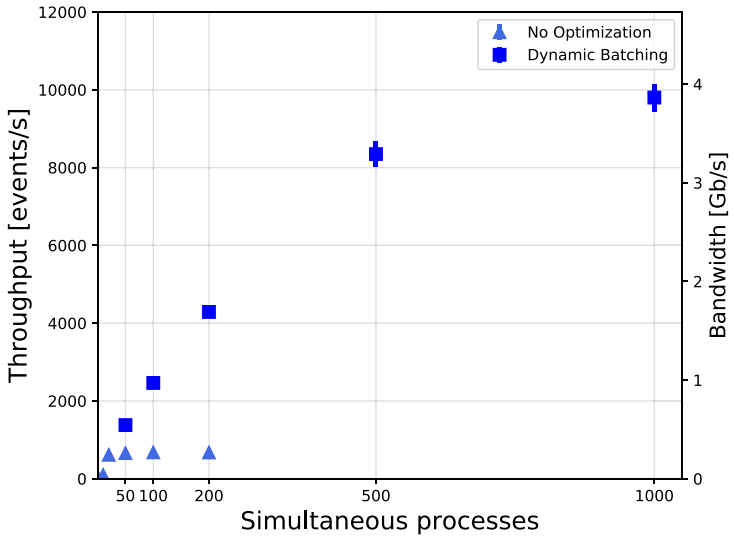

Figure 6. The throughput of DeepCalo as a service on an NVIDIA V100 GPU in events per second versus the number of simultaneous clients. The markers indicate the mean throughput and the error bars indicate the standard deviation. With server side optimizations, the batch size is configured dynamically to prefer a batch of 250 or 500 (as explained in section 5.3.1) and to use five concurrent model executions. The network bandwidth from the transfer of inputs from the client to the server is shown on the right vertical axis.

iteratively reconfigured our server to deploy multiple machines behind a load balancer, as described in section 5.3.2.

5.2.2. DeepCalo

As DeepCalo performs an image classification task, we expect it to be computationally bound rather than bandwidth limited. In our studies, we investigate the application of DeepCalo in offline reconstruction, which is throughput limited. We evaluate the performance of running DeepCalo as a service by running up to 1000 clients on-premises on Fermilab's computing cluster and deploying a GPU server in Google cloud. We set the batch size to 5 because this is the approximate number of electrons expected per reconstructed collision in a realistic LHC scenario.

We consider the case of a single NVIDIA V100 GPU server deployed on Google Cloud. The results are shown in figure 6. For a batch size of 5, the throughput increases rapidly until 20 simultaneous clients, and it saturates at 680 events per second between 20 and 50 simultaneous clients. At 200 simultaneous clients, the utilization of the GPU saturates at 45% and the bandwidth peaks at 270 Mb s^{-1}, suggesting that the batch size is limiting GPU utilization. Further optimizations to DeepCalo, namely dynamic batching, are discussed in section 5.3.1.

In a previous study, the latency on four Xeon E5-2698 CPUs was found to be 15 ms per electron–or 75 ms for an event of 5 electrons [62]. With our baseline GPU performance, we compute 680 events per second or 1.5 ms per event. Combining our GPU result and CPU result, we observe a factor of 50 improvement in the throughput.

5.2.3. ResNet-50

ResNet-50 is deployed as a service with clients on Fermilab's computing cluster and servers with one, four, and eight NVIDIA Tesla T4 GPUs in Google Cloud. The throughput obtained using ResNet-50 are shown in figure 7.

A single GPU server saturates at 25 (batch 10) inferences per second, at about 10 clients. We find that the throughput scales linearly with the number of GPUs, although with approximately 85% efficiency, slightly lower than for FACILE.

5.3. Optimizations

5.3.1. Dynamic batching

Dynamic batching is a feature of the Triton Inference Server that serves to increase both the throughput and hardware efficiency. It performs an added server-side queue of requests from the client until an optimal batch size is reached. The performance gains of dynamic batching are also related to the 'instance groups,' or simultaneous executions of a model. This poses an optimization problem between the dynamic batch size and model concurrency. The use of dynamic batching is particularly interesting in that it circumvents the

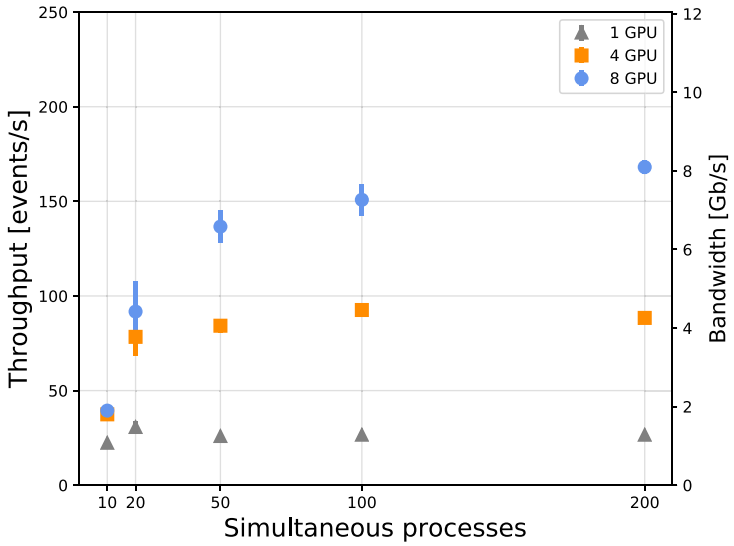

Figure 7. Throughput of ResNet-50 as a service in events per second versus the number of simultaneous clients. The markers indicate the mean throughput and the error bars indicate the standard deviation. The throughput tests are repeated with one, four, and eight GPUs connected to the server, represented by yellow triangles, gold squares, and orange circles, respectively. The network bandwidth from the transfer of inputs from the client to the server is shown on the right vertical axis.

LHC computing paradigm of splitting computations on an event-by-event basis. Here, multiple events can be processed simultaneously within a single computation, without redesigning the computing model. We stress that this type of scheduling is *only* beneficial when GPUs are servicing a large number of parallel processes.

In the initial DeepCalo measurements, we found that the choice of batch size limited the utilization and throughput of the GPU. In our studies, we found that a low number of model execution instances and a high dynamic batch size yielded the best throughput result. Figure 6 shows the throughput gains when using dynamic batching. At 200 simultaneous clients, the throughput shows no signs of saturation; at this point, the throughput is about 4200 events per second, representing a factor of 6 improvement. When extended to 1000 simultaneous clients, the throughput reaches 9800 events per second, representing a factor of 14 gain in throughput, and the GPU utilization increases to 80%. At 1000 clients, the bandwidth peaks at 3.9 Gb s^{-1}, which is roughly the same bandwidth limit observed with FACILE. On the other hand, dynamic batching for FACILE yielded no gain in throughput. We conclude that the most significant gains of dynamic batching are found for large models that naturally operate with small batch sizes.

5.3.2. Server optimization and monitoring

We performed tests on many different combinations of computing hardware, which provided us with a deeper understanding of networking limitations within Google Cloud and on-premises data centers. Even though the Triton Inference Server does not consume significant CPU power, the number of CPU cores provisioned for the node did have an impact on the maximum ingress bandwidth reached in our early tests.

To scale the GPU throughput flexibly, we deployed a Google Kubernetes Engine cluster for server side workloads. The cluster was configured using a Deployment and ReplicaSet [72], which control Pods, groups of one or more containers with shared storage and network and a specification to run the containers [73], and their resource requests. Additionally, a load balancing service was deployed which distributed incoming network traffic among the Pods. We implemented Prometheus-based monitoring [74] of overall system health and inference statistics. All metrics were visualized through a Grafana [75] instance, also deployed in the same cluster.

We note a Pod's contents are always co-located and co-scheduled, and run in a shared context within Kubernetes Nodes [73]. We kept the Pod to Node ratio at 1:1 throughout the studies, with each Pod running an instance of Triton Inference Server (v20.02-py3) from the NVIDIA Docker repository.

It can be naively assumed that a small instance `n1-standard-2` with 2 vCPUs, 7.5 GB of memory, and different GPU configurations (1, 2, 4, 8) would be able to handle the workload. However, Google Cloud imposes a hard limit on network bandwith per virtual CPU (vCPU). After performing several tests, we found that horizontal scaling would allow us to increase our ingress bandwidth since Google Cloud imposes a hard limit on network bandwidth at 2 Gb s^{-1} per vCPU up to a theoretical maximum of 16 Gb s^{-1} for each virtual machine [72].

Table 2. Summary of the three algorithms in terms of replacement with respect to a CPU ($F_{\mathrm{GPU}}^{\mathrm{eq}}$), inferences per second, individual packet size (Mb), and network bandwidth per GPU (Gb s^{-1}). These numbers are quoted for the desired batch per event. The asterisk (*) identifies an algorithm using dynamic batching.

Algorithm	$F_{\mathrm{GPU}}^{\mathrm{eq}}$	Inf./s (Hz)	Size/batch (Mb)	Throughput/GPU (Gb s^{-1})
FACILE	27	500	7.7	3.9
DeepCalo	50	680	0.4	0.3
DeepCalo*	750	9800	0.4	3.9
ResNet-50	150	25	48.2	1.2

Given these parameters, the ideal setup for optimizing ingress bandwidth was to provision multiple Pods on 16-vCPU machines with fewer GPUs per Pod. For GPU-intensive tests, we took advantage of having a single point of entry through Kubernetes load balancing and provisioning multiple identical Pods, where the sum of the GPUs attached to each Pod is the total GPU requirement.

5.3.3. Future optimizations

Throughout these tests, we monitored the GPU utilization. ResNet-50 largely saturated GPU utilization, whereas FACILE and DeepCalo used 20% and 45% of the GPU, respectively. This indicates the throughput is batch limited. Throughput and GPU utilization can be optimized using dynamic batching for DeepCalo, as described above, but not for FACILE. Follow-up studies will investigate optimizations for small models like FACILE.

6. Scaling

We now apply our findings to determine the GPU resources, networking and compute resources required for a modified LHC computing system in which algorithms with large speedups on coprocessors are deployed as a service. To assess the amount needed, we compute the required resources for the scaling of the three algorithms in either the HLT or offline computing cases. As a benchmark, we use the plateau performance numbers measured per GPU for each of the algorithms. We summarize these in table 2. As an estimate of the total amount of required resources, we make some assumptions for the typical latency and throughput that we would expect for the online reconstruction and offline reconstruction. We emphasize that these numbers are approximate and used here for illustrative purposes.

A typical LHC HLT consists of 1000 servers, each with 32 cores, for a total of 32 000 CPU cores. For a 0.5 s latency, this system would process events at a rate of 64 000 events per second. The goal of the HLT is decide whether the event is sufficiently interesting to preserve for further reconstruction. As a consequence, the HLT performs a sequential, tiered reconstruction and filtering of the event and immediately rejects the event if it fails any filter in the sequence in order to prevent further reconstruction [76, 77]. With an emulated HLT, we find that a single GPU running FACILE can serve 300–500 different HLT nodes while preserving a 10% reduction in the per event latency. This means that roughly 100 GPUs are needed to serve FACILE for the whole HLT. Moreover, if 100 GPUs were added to the system, 10% or 3200 CPU cores could be removed from the system. We emphasize that FACILE represents only the tip of the iceberg in the use of deep learning in low-level reconstruction at the LHC. The algorithm uses less than a tenth of the GPU memory, and its GFLOP is less than a tenth of the other algorithms (see table 1). As a result, it can be extended to carry out sophisticated reconstruction tasks (thus merging multiple algorithms) without large increases in latency.

While there is a significant reduction in the number of processing cores, there is an increase in network usage. With each GPU, an additional network bandwidth of 3.9 Gb s^{-1} is required. A server of 10 GPUs would thus require 40 Gb s^{-1} while simultaneously serving 100 HLT servers (3200 cores). While this bandwidth is large, it is already attainable. A setup of one 10 GPU server, serving 100 HLT servers, would be a logical design for the system that could be implemented with existing technology.

Lastly, we consider the option of adapting the DeepCalo algorithm or ResNet-50 to run in the HLT with our benchmark batch values. We expect that a six GPU server with a bandwidth of 24 Gb s^{-1} would be sufficient to run DeepCalo for the whole HLT system. ResNet-50 with the default batch size of 10 images would require 2500 GPUs with a total added bandwidth of 2.9 Tb s^{-1}. The large number of GPUs and the large bandwidth would require significant and costly modifications to the design of the current HLT, making it impractical. However, a ResNet-50 implementation using a batch size of 1, or equivalently an inference rate below the expectation for a batch size of 1 by applying the algorithm only to some events, would result in a comparable number of GPU servers to FACILE. Here, Resnet-50 is only used to identify hadronically decaying top quarks. The rate of objects within an event that have properties that would require top quark

identification is very low. Therefore, batch-1 or lower is more reflective of the true physics use case. This conclusion meshes well with the fact that high-energy top quark candidates are relatively rare so it may be reasonable to run a ResNet-50 top quark tagging algorithm once or less per event.

To run FACILE in the existing LHC offline computing system with the approximate throughput of 5000 reconstructed events per second, a single server of 10 GPUs operating with a bandwidth of 39 Gb s^{-1} would be needed to sustain the full reconstruction load. Applying the same scaling for DeepCalo, we find that one GPU would be sufficient to run the reconstruction for a whole LHC experiment. Lastly, for ResNet-50 at a batch size of 10, a setup of 200 GPUs with 240 Gb s^{-1} bandwidth would be sufficient to support 150 000 CPU cores. In this case, with ResNet-50 applied to all reconstructed events, a realistic scenario would consist of 10 separate GPU servers, each running with 24 Gb s^{-1} bandwidth. In contrast, for ResNet-50 inference on CPUs with a batch size of 10, the reconstruction time per event would increase by 18 s. This would require a 60% increase in the computing clusters or an additional 90 000 cores to sustain the same throughput.

In the context of LHC algorithms, DL algorithms are being developed at all levels of the detector reconstruction. Algorithms that run on aggregate event properties are comparable in size to ResNet-50, such as jet tagging algorithms [67, 68, 78]. The full collection of particles in a collision after reconstruction is found to be on the order of 1000 particles per event [79]. Given the number of particles, a computation of the event size ranges from 0.5 to 2.5 Mb, which is less than the size of a single event request performed in tests with FACILE. Consequently, we observe that data rates and throughput for future LHC algorithms are comparable to that of the results presented with FACILE. Therefore, at no added complexity in networking or design, the framework presented in this paper can be extended for algorithms designed for the future particle reconstruction at the LHC.

7. Conclusion

We have demonstrated a core framework that enables the use of deep learning (DL) as a service with direct applications to the processing of LHC data. Our framework, Services for Optimized Network Inference on Coprocessors (SONIC), utilizes gRPC to perform asynchronous, non-blocking calls to a GPU server. Our server infrastructure can address both small and large scale use cases. With our infrastructure, we have tested three algorithms that span a large range of DL model complexities and batch sizes. Together, these algorithms serve as benchmarks for a wide array of LHC reconstruction tasks. In each case, we have measured the algorithm throughput and demonstrated comparable throughput for as-a-service computing both remotely and on-site. We fully integrated a DL algorithm called FACILE for LHC reconstruction in the high-level trigger (HLT), the second tier of the LHC data processing and filter, and we found that this algorithm can lead to a 10% overall reduction in the computing demands. This latency reduction is almost equivalent to removing the hadron calorimeter reconstruction latency from the HLT entirely, and it matches the expected optimal performance when performing standalone algorithmic tests. Finally, we demonstrated the use of FACILE, DeepCalo, and ResNet-50 for offline reconstruction. A server implementation in the cloud was found to operate at data rates and inference times comparable to the demands set by LHC offline reconstruction. This is a concrete validation of the SONIC framework, demonstrating the viability of coprocessors as a service on representative scales for LHC computing.

While our focus was on accelerating DL algorithms with GPUs, this work can be applied to any algorithm that can be implemented on a GPU and appropriately integrated into a GPU server. This work is largely agnostic to the hardware and software implementations of the algorithm and can be adapted for other types of coprocessors and other scientific experiments. As DL accelerator tools are constantly evolving and improving, we expect the speedups observed in this paper to become even larger.

From our studies, we delineated certain considerations for designing an optimal system with GPUs as a service (GPUaaS) for the LHC. In particular, an optimized scheduling framework is needed to ensure that remote operations of algorithms incur minimal losses in performance. Additionally, sufficient bandwidth is needed to ensure that the full performance of the accelerator servers can be achieved for both remote and local as-a-service operation. Finally, both a load balancer and an optimized and flexible server infrastructure are needed to ensure robust operation. In this paper, we have demonstrated that all of these demands can be met with existing resources.

In the context of physics performance, our results lead to direct performance improvements that can be implemented immediately in the LHC computing model. In particular, we found that: (1) DL inference as a service can enable a significant increase in event throughput, (2) algorithms with complexities not previously attainable can be operated in the LHC reconstruction workflow, (3) optimized versions of algorithms can be implemented without disrupting the existing computing model, and (4) simultaneous multi-event processing is achievable in the reconstruction workflow. Concurrently with these studies, an extensive suite of new DL techniques for LHC reconstruction have been developed [25, 78, 80–96], which explore new

architectures such as equivariant neural networks, graph neural networks, and attention-based transformers. The current work will enable the integration of DL algorithms, including these, into the LHC computing model in a seamless and computationally efficient way. Our work can also be extended by adapting to the unique computing challenges that emerge from the next generation of DL algorithms. In the future, we plan to investigate the requirements and constraints at high performance computing (HPC) centers, in order to leverage their resources for reliable local operations of a large number of GPUs. This work can also help drive preparation towards a computing model for future high-luminosity LHC running, where larger rates and numbers of channels will further stretch computing capacity.

We would like to stress that this work represents the beginning of developments in coprocessor computing both at the LHC and other large scale experiments. This work and related studies are encouraging for physicists in other fields, such as neutrino physics [97], gravitational wave detection, and astrophysics, to pursue a similar computing model. As a consequence, we believe that this work may lead to a paradigm shift in the scientific computing model, enabling us to meet the enormous scientific computing demands in the next decade.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/Keb-L/trtmodels, https://github.com/mapsacosta/i2gpu_gketests, https://github.com/fastmachinelearning/SonicCMS/tree/master.

Acknowledgments

P H, and D R are supported by NSF Grants #1934700, #1931469, and the IRIS-HEP Grant #1836650. J K is supported by NSF Grant #190444 and NSERC PGS D. Cloud credits for this study were provided by Internet2 managed Exploring Cloud to accelerate Science (NSF Grant #190444). Additional support on networking was provided by Doug Burton, Dan Speck, and Aaron Strong of the Burwood group. Also, we thank Emma Levett from MIT for additional networking support. We thank Steven Timm for support with HEPClo. P. H. would like to thank Matthew Harris for discussion on GCP networking capabilities. M A F, B H, T K, M L, K P, N T are supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy (DOE), Office of Science, Office of High Energy Physics. K P is partially supported by the High Velocity Artificial Intelligence grant as part of the DOE High Energy Physics Computational HEP sessions program. J D is supported by the DOE, Office of Science, Office of High Energy Physics Early Career Research program under Award No. DE-SC0021187.

ORCID iDs

Jeffrey Krupa https://orcid.org/0000-0003-0785-7552
Kelvin Lin https://orcid.org/0000-0002-1494-1464
Maria Acosta Flechas https://orcid.org/0000-0001-5602-4704
Jack Dinsmore https://orcid.org/0000-0002-6401-778X
Javier Duarte https://orcid.org/0000-0002-5076-7096
Philip Harris https://orcid.org/0000-0001-8189-3741
Scott Hauck https://orcid.org/0000-0001-9516-0311
Burt Holzman https://orcid.org/0000-0001-5235-6314
Shih-Chieh Hsu https://orcid.org/0000-0001-6214-8500
Thomas Klijnsma https://orcid.org/0000-0003-1675-6040
Kevin Pedro https://orcid.org/0000-0003-2260-9151
Dylan Rankin https://orcid.org/0000-0001-8411-9620
Matt Trahms https://orcid.org/0000-0001-9884-2326
Nhan Tran https://orcid.org/0000-0002-8440-6854

References

[1] Evans L, Bryant P and Machine L H C 2008 *J. Instrum.* **3** S08001
[2] Boyd J 2020 LHC Run-2 and future prospects *2019 European School of High-Energy Physics* (arXiv:2001.04370)
[3] Gerber C E 2019 LHC highlights and prospects *10th CERN—Latin-American School of High-Energy Physics* (arXiv:1909.10919)
[4] ATLAS and CMS Collaborations 2019 Report on the physics at the HL-LHC and perspectives for the HE-LHC *Cern Yellow Reports: Monographs* vol 7/2019 (Geneva: CERN) (https://doi.org/10.23731/CYRM-2019-007)
[5] Alimena J *et al* 2020 Searching for long-lived particles beyond the standard model at the large Hadron collider *J. Phys.* G **47** 090501

[6] Abercrombie D *et al* 2020 Dark matter benchmark models for early LHC Run-2 searches: report of the ATLAS/CMS dark matter forum *Phys. Dark Universe* **27** 100371
[7] ATLAS Collaboration 2012 Observation of a new particle in the search for the standard model Higgs boson with the ATLAS detector at the LHC *Phys. Lett.* B **716** 1
[8] CMS Collaboration 2012 Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC *Phys. Lett.* B **716** 30
[9] CMS Collaboration 2013 Observation of a new boson with mass near 125 GeV in pp collisions at $\sqrt{s} = 7$ and 8 TeV *J. High Energy Phys.* **06** 081
[10] CMS Collaboration 2018 Search for new physics in final states with an energetic jet or a hadronically decaying W or Z boson and transverse momentum imbalance at $\sqrt{s} = 13$ TeV *Phys. Rev.* D **97** 092005
[11] CMS Collaboration 2020 Search for high mass dijet resonances with a new background prediction method in proton–proton collisions at $\sqrt{s} = 13$ TeV *J. High Energy Phys.* **05** 033
[12] CMS Collaboration 2019 Search for low mass vector resonances decaying into quark-antiquark pairs in proton–proton collisions at $\sqrt{s} = 13$ TeV *Phys. Rev.* D **100** 112007
[13] Alexander J *et al* 2016 Dark sectors 2016 workshop: community report *Dark Sectors 2016 Workshop 2016* (arXiv:1608.08632)
[14] HEP Software Foundation Collaboration 2019 A roadmap for HEP software and computing R&D for the 2020s *Comput. Softw. Big Sci.* **3** 7
[15] CMS Collaboration 2020 CMS offline and computing public results (available at: twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults)
[16] ATLAS Collaboration 2020 ATLAS computing and software public results (available at: twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults)
[17] HEPiX Benchmarking Working Group 2017 HEP-SPEC06 (available at: w3.hepix.org/benchmarking.html)
[18] Dennard R H *et al* 1974 Design of ion-implanted MOSFET's with very small physical dimensions *IEEE J. Solid-State Circuits* **9** 256
[19] Esmaeilzadeh H *et al* 2011 Dark silicon and the end of multicore scaling *Proc. 38th Annual Int. Symp. Computer Architecture, ISCA'11* (New York: ACM) p 365
[20] Duarte J *et al* 2019 FPGA-accelerated machine learning inference as a service for particle physics computing *Comput. Softw. Big Sci.* **3** 13
[21] Guest D, Cranmer K and Whiteson D 2018 Deep learning and its application to LHC physics *Annu. Rev. Nucl. Part. Sci.* **68** 161
[22] Albertsson K *et al* 022008 Machine Learning in High Energy Physics Community White Paper *J. Phys. Conf. Ser.* **1085** 2018
[23] Bourilkov D 2020 Machine and deep learning applications in particle physics *Int. J. Mod. Phys.* A **34** 1930019
[24] Larkoski A J, Moult I and Nachman B 2020 Jet substructure at the large Hadron collider: a review of recent advances in theory and machine learning *Phys. Rep.* **841** 1
[25] Qasim S R, Kieseler J, Iiyama Y and Pierini M 2019 Learning representations of irregular particle-detector geometry with distance-weighted graph networks *Eur. Phys. J.* C **79** 608
[26] Belayneh D *et al* 2020 Calorimetry with deep learning: particle simulation and reconstruction for collider physics *Eur. Phys. J.* C **80** 688
[27] Komiske P T, Metodiev E M and Schwartz M D 2017 Deep learning in color: towards automated quark/gluon jet discrimination *J. High Energy Phys.* **01** 110
[28] ATLAS Collaboration 2017 Quark versus gluon jet tagging using jet images with the ATLAS detector *ATLAS Public Note* ATL-PHYS-PUB-2017-017 (available at: http://cds.cern.ch/record/2275641)
[29] Andrews M *et al* 2020 End-to-end jet classification of quarks and gluons with the CMS Open Data *Nucl. Instrum. Methods Phys. Res.* A **977** 164304
[30] WLCG Grid Deployment Board 2019 Benchmarking (available at: indico.cern.ch/event/739897/)
[31] Mell P and Grance T 2011 The NIST definition of cloud computing *NIST Special Publication* SP 800-145 (available at: http://dx.doi.org/10.6028/NIST.SP.800-145)
[32] Schreiner H *et al* 2018 GooFit 2.0 *J. Phys. Conf. Ser.* **1085** 042014
[33] Heinrich L, Feickert M and Stark G scikit-hep/pyhf: v0.5.3 2020 (available at: http://dx.doi.org/10.5281/zenodo.4110938)
[34] Pata J and Spiropulu M 2019 Processing columnar collider data with GPU-accelerated kernels (arXiv:1906.06242)
[35] ALICE Collaboration 2019 Real-time data processing in the ALICE high level trigger at the LHC *Comput. Phys. Commun.* **242** 25
[36] Aaij R *et al* 2020 Allen: a high level trigger on GPUs for LHCb *Comput. Softw. Big Sci.* **4** 7
[37] Vom Bruch D 2020 Real-time data processing with GPUs in high energy physics *J. Instrum.* **15** C06010
[38] Bocci A *et al* 2020 Heterogeneous reconstruction of tracks and primary vertices with the CMS pixel tracker *Front. Big Data* **3** 49
[39] Funke D *et al* 2014 Parallel track reconstruction in CMS using the cellular automaton approach *J. Phys. Conf. Ser.* **513** 052010
[40] Pantaleo F *et al* 2016 Development of a phase-II track trigger based on GPUs for the CMS experiment *2015 IEEE Nuclear Symp. and Medical Conf. (NSS/MIC)* p 7581775
[41] Lamanna G 2014 Almagest, a new trackless ring finding algorithm *Nucl. Instr. Methods Phys. Res.* A **766** 241
[42] Färber C, Schwemmer R, Machen J and Neufeld N 2017 Particle identification on an FPGA accelerated compute platform for the LHCb upgrade *IEEE Trans. Nucl. Sci.* **64** 1994
[43] Mu3e Collaboration 2017 Online data reduction using track and vertex reconstruction on GPUs for the Mu3e experiment *Eur. Phys. J. Conf.* **150** 00013
[44] Ammendola R *et al* 2018 Real-time heterogeneous stream processing with NaNet in the NA62 experiment *J. Phys. Conf. Ser.* **1085** 032022
[45] Chirkin D *et al* 2019 Photon propagation using GPUs by the IceCube neutrino observatory *15th Int. Conf. EScience, EScience 2019 (San Diego, CA, 24–27 September 2019)* (IEEE) p 388
[46] Sfiligoi I, Wuerthwein F, Riedel B and Schultz D 2020 Running a pre-exascale, geographically distributed, multi-cloud scientific simulation *High Performance Computing* (Cham: Springer) p 23
[47] Pedro K 2020 SonicCMS (version v5.1.0) (available at: https://github.com/fastmachinelearning/SonicCMS) (accessed 22 April 2020)
[48] Caulfield A *et al* 2016 A cloud-scale acceleration architecture *2016 49th Annual IEEE/ACM Int. Symp. Microarchitecture (MICRO)* (IEEE) p 1
[49] ALICE Collaboration 2015 Technical design report for the upgrade of the online-offline computing system *ALICE Technical Design Report* (available at: cds.cern.ch/record/2011297)

[50] NVIDIA 2019 Triton Inference Server (version v1.8.0) (available at: https://docs.nvidia.com/deeplearning/sdk/triton-inference-server-guide/docs/index.html) (accessed 17 February 2020)
[51] Google 2018 gRPC (version v1.19.0) (available at: https://grpc.io/) (accessed 17 February 2020)
[52] CMS Collaboration 2006 CMS physics: technical design report volume 1: detector performance and software *CMS Technical Design Report* CERN-LHCC-2006-001 (available at: http://cds.cern.ch/record/922757)
[53] Intel 2018 Thread Building Blocks (version 2018_U1) (available at: www.threadingbuildingblocks.org) (accessed 11 February 2019)
[54] Bocci A *et al* 2020 Bringing heterogeneity to the CMS software framework *EPJ Conf.* **245** 05009
[55] Rovere M *et al* 2020 CLUE: a fast parallel clustering algorithm for high granularity calorimeters in high energy physics *Front. Big Data* **3** 41
[56] Ioffe S and Szegedy C 2015 Batch normalization: accelerating deep network training by reducing internal covariate shift *Proc. 32nd Int. Conf. Machine Learning* (*Proceedings of Machine Learning Research* vol 37) eds F Bach and D Blei (Lille: PMLR) p 448 (arXiv:1502.03167)
[57] Nair V and Hinton G E 2010 Rectified linear units improve restricted Boltzmann machines *Proc. 27th Int. Conf. Machine Learning, ICML'10* (Madison, WI: Omnipress) p 807
[58] Glorot X, Bordes A and Bengio Y 2011 Deep sparse rectifier neural networks *Proc. 14th Int. Conf. on Artificial Intelligence and Statistics (AISTATS) (Fort Lauderdale, FL)* (*JMLR Workshop Conference Proceedings* vol 15) eds G Gordon, D Dunson and M Dudík p 315 (http://proceedings.mlr.press/v37/ioffe15.html)
[59] Kingma D P and Ba J 2015 Adam: a method for stochastic optimization *3rd Int. Conf. Learning Representations, ICLR 2015* (*Conference Track Proceedings*) eds Y Bengio and Y LeCun (arXiv:1412.6980)
[60] Lawhorn J 2019 New method of out-of-time energy subtraction for the CMS hadronic calorimeter *J. Phys. Conf. Ser.* **1162** 012036
[61] Massironi A, Khristenko V and DAlfonso M 2020 Heterogeneous computing for the local reconstruction algorithms of the CMS calorimeters *J. Phys. Conf. Ser.* **1525** 012040
[62] Faye F 2019 Energy reconstruction of electrons and photons using convolutional neural networks Master's Thesis University of Copenhagen
[63] ATLAS Collaboration 2019 Electron and photon energy calibration with the ATLAS detector using 2015–2016 LHC proton-proton collision data *J. Instrum.* **14** 03017
[64] ATLAS Collaboration 2019 Electron and photon performance measurements with the ATLAS detector using the 2015–2017 LHC proton-proton collision data *J. Instrum.* **14** 12006
[65] Maas A, Jannun A and Ng A 2013 Rectifier nonlinearities improve neural network acoustic models *30th Int. Conf. Machine Learning (ICML), Workshop on Deep Learning for Audio, Speech and Language Processing (WDLASL)* (https://7e0826b8-a-62cb3a1a-s-sites.googlegroups.com/site/deeplearningicml2013/relu_hybrid_icml2013_final.pdf?attachauth=ANoY7cpiKG3yXJMENOqmuInYonIX_fpcApQzoQE_5W5YcQnMH-tl6qW5AMAv-zp4Qm-TmZj1uH91Fq-dal2n72mo0ngeW-tElIvvi8N4tGmBOy0UkJ5OzZhISxQa2ZGdmKofsFFvFUel3kVHh5c92B4ecGaqf8V-IwmpWdWwLTuUrOjX6dBtudLnJfyt8GHIKay-Lc-BdfBt3B8Z3sgQ8SfXbkB8K7kuyLuuBpB43hELYjxMWs6b_XARK1ihnFoTVUVt-PiH5b4y&attredirects=0)
[66] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *2016 Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE) p 770
[67] CMS Collaboration 2020 Identification of heavy, energetic, hadronically decaying particles using machine-learning techniques *J. Instrum.* **15** 06005
[68] Butter A *et al* 2019 The machine learning landscape of top taggers *SciPost Phys.* **7** 014
[69] Holzman B *et al* 2017 HEPCloud, a new paradigm for HEP facilities: CMS amazon web services investigation *Comput. Softw. Big Sci.* **1** 1
[70] Altunay M *et al* 2018 Intelligently-automated facilities expansion with the HEPCloud decision engine *2018 18th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing (CCGRID)* p 352
[71] Mhashilkar P *et al* 2019 HEPCloud, an elastic hybrid HEP facility using an intelligent decision support system *Eur. Phys. J. Conf.* **214** 03060
[72] Google LLC 2020 *Compute Engine Documentation—Concepts—Virtual Machine Instances—Machine Types* (available at: https://cloud.google.com/compute/docs/machine-types)
[73] Kubernetes Authors 2020 *Documentation—Concepts—Workloads—Pods—Pods* (available at: https://kubernetes.io/docs/concepts/workloads/pods/pod/)
[74] Prometheus Authors 2020 *Prometheus* (available at: https://prometheus.io/)
[75] Grafana Labs 2020 *Grafana* (available at: https://grafana.com/)
[76] Perrotta A 2015 Performance of the CMS High Level Trigger *J. Phys. Conf. Ser.* **664** 082044
[77] CMS Collaboration 2018 CMS trigger performance *Eur. Phys. J. Conf.* **182** 02037
[78] Qu H and Gouskos L 2020 ParticleNet: jet tagging via particle clouds *Phys. Rev.* D **101** 056019
[79] CMS Collaboration 2018 Measurement of charged particle spectra in minimum-bias events from proton–proton collisions at $\sqrt{s} = 13$TeV *Eur. Phys. J.* C **78** 697
[80] CMS Collaboration 2020 A deep neural network to search for new long-lived particles decaying to jets *Mach. Learn.: Sci. Technol.* **1** 035012
[81] Metodiev E M, Nachman B and Thaler J 2017 Classification without labels: learning from mixed samples in high energy physics *J. High Energy Phys.* **10** 174
[82] Nachman B and Shih D 2020 Anomaly detection with density estimation *Phys. Rev.* D **101** 075042
[83] Andreassen A, Nachman B and Shih D 2020 Simulation assisted likelihood-free anomaly detection *Phys. Rev.* D **101** 095004
[84] Collins J H, Howe K and Nachman B 2018 Anomaly detection for resonant new physics with machine learning *Phys. Rev. Lett.* **121** 241803
[85] Collins J H, Howe K and Nachman B 2019 Extending the search for new resonances with machine learning *Phys. Rev.* D **99** 014038
[86] Farina M, Nakai Y and Shih D 2020 Searching for new physics with deep autoencoders *Phys. Rev.* D **101** 075021
[87] Heimel T, Kasieczka G, Plehn T and Thompson J M 2019 QCD or what? *SciPost Phys.* **6** 030
[88] Farrell S *et al* 2018 Novel deep learning methods for track reconstruction *4th Int. Workshop Connecting the Dots 2018 (CTD2018)* (arXiv:1810.06111)
[89] Ju X *et al* 2019 Graph neural networks for particle reconstruction in high energy physics detectors *Machine Learning and the Physical Sciences Workshop at the 33rd Conf. Neural Information Processing Systems* (arXiv:2003.11603)
[90] Moreno E A *et al* 2020 JEDI-net: a jet identification algorithm based on interaction networks *Eur. Phys. J.* C **80** 58
[91] Moreno E A *et al* 2020 Interaction networks for the identification of boosted $H \to b\bar{b}$ decays *Phys. Rev.* D **102** 012010

[92] Choma N et al 2020 Track seeding and labelling with embedded-space graph neural networks 6th Int. Workshop Connecting the Dots 2020 (arxiv:2007.00149)
[93] Bogatskiy A 2020 et al Lorentz group equivariant neural network for particle physics (arXiv:2006.04780)
[94] Kieseler J 2020 Object condensation: one-stage grid-free multi-object reconstruction in physics detectors, graph and image data Eur. Phys. J. C 80 886
[95] Mikuni V and Canelli F 2020 ABCNet: an attention-based method for particle tagging Eur. Phys. J. Plus 135 463
[96] Pata J et al 2021 MLPF: efficient machine-learned particle-flow reconstruction using graph neural networks (arXiv:2101.08578)
[97] Wang M et al 2020 GPU-accelerated machine learning inference as a service for computing in neutrino experiments Front. Big Data 3 48