CNN Training Latency Prediction Using Hardware Metrics on Cloud GPUs

Yoonseo Hur
yoonseo@kookmin.ac.kr
Computer Science. Kookmin Univ.
Seoul, South Korea

Kyungyong Lee
leeky@kookmin.ac.kr
Computer Science. Kookmin Univ.
Seoul, South Korea

***Abstract*—Convolutional neural network (CNN) models are becoming larger and more sophisticated over time, and training requires a significant amount of time and computing resources. To meet computing demand, graphics processing units (GPUs) are widely used for training. Due to the excessive cost and overhead of maintaining a GPU cluster, users may prefer to use GPUs provided by a public cloud vendor rather than creating their own servers. The initial cloud service is offered in the Infrastructure-as-a-Service (IaaS) model. As the cloud evolves, the abstraction level of the public cloud service becomes higher, and serverless computing is considered the next-generation cloud service. In the new way of offering cloud services, vendors are required to provide an efficient environment for diverse workloads. To meet the new requirement of the evolving cloud service, this paper proposes heuristics to predict the training latency on various GPU devices without using model information to help cloud-service vendors prepare an efficient training environment with minimal exposure to users' model architectures. Unlike previous work that relies on internal model details for latency prediction, the proposed system uses only the hardware metrics that are extracted during training. Using the information, we first propose an algorithm to detect an epoch period that we aim to predict. The detected epoch period becomes the target latency to predict, for which we apply a stacked regressor to achieve superior prediction accuracy. Detailed experiments revealed that the average prediction accuracy of the proposed training latency prediction model is 15.14%, which is similar to the state-of-the-art approach that references the internal architecture of the model. Unlike previous work, the proposed work does not reference the model's internal architecture, which proves the applicability of this proposed work in the next-generation cloud service.**

***Index Terms*—CNN training, performance model, GPU, cloud computing**

I. Introduction

Recent advances in artificial intelligence have increased the prevalence of CNN [1] in a multitude of disciplines. The huge massive requirement for computing power and the increasing intricacy of CNN models indicate the superiority of GPUs that are specifically designed for parallel processing over CPUs for training. Given the financial and operational overhead associated with establishing a personal GPU training environment, there has been a discernible shift towards cloud services. Using cloud services facilitates an effortless environment setup tailored to user specifications, offering cost efficiency, as users are billed based on their consumption. Leading cloud platforms, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), offer a multitude of services. Due to the vast array of services provided by in the cloud, users often find themselves at a crossroads, deliberating which platforms, instance types, and services to choose.

Cloud computing services are developing in the direction of hiding complex service management with a higher degree of service abstraction. This evolution is represented by the development of serverless computing, which reduces the overhead of developers in building highly available systems [2]. The main components of serverless computing are various fully-managed services in which the service provider is responsible for choosing appropriate instance types and scaling decisions. To provide a higher degree of service abstraction layer, cloud-service providers should be able to choose optimal instance types for various implementations of deep neural network (DNN) models to provide an efficient serverless DNN development environment.

In the literature, many studies have proposed heuristics to predict latency when a model is trained on GPUs [3], [4], [5], [6]. All previous work has relied on the internal model architecture or profile output during the training phase to model training latency. Although previous work has shown decent prediction accuracy that has less than 10% mean absolute percentage error (MAPE), it is not feasible to apply these in a new public cloud service model. Previous methods require the help of users to provide an optimal environment, either by providing the source code of the model architecture [3], [7], [4] or profiling the output during training [5], [6]. The service provider and the consumer of the public cloud service are different, and most users are hesitant to provide highly confidential and private asset information to a service provider.

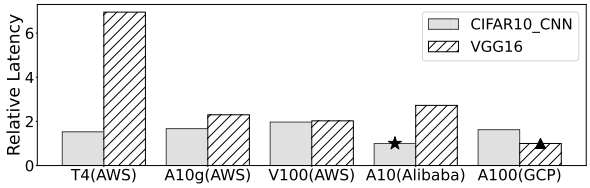
This paper proposes an algorithm to predict the training latency of various CNN models on cloud computing GPU instances using only metrics extracted from the hardware to minimize the users' help. This unique approach allows cloud vendors to suggest the most cost-effective and time-efficient environment without referencing the internal model architecture or profile results of a model.

The proposed system consists of two primary sections. First, we propose heuristics to detect the train epoch period from hardware metrics. Based on the detected epoch period, we design a model to forecast latency on various GPU instances using dynamic and static metric information as model features. A thorough evaluation reveals that the average MAPE of the

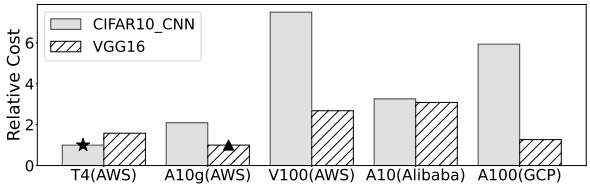
GPU Instance on Cloud Services									
GPU Characteristic					Cloud Instances				
GPU	Releasd Date	CC[1]	GPU Core	FLOPS (FP32)	AWS	Azure	IBM	GCP	Alibaba
K80	2014.11	3.7	2496	4.113	P2	NC-series		O	
M60	2015.08	5.2	2048	4.825	G3s	NV-series			
P100	2016.06	6.0	3584	9.526		NCv2-series	AC1	O	gn5
P4	2016.09	6.1	2560	5.704				O	gn5i
P40	2016.09	6.1	3840	11.76		ND-series			
V100	2017.12	7.0	5120	14.13	P3	NCv3-series	AC2	O	gn6e,gn6v
T4	2018.09	7.5	2560	8.141	G4dn	NCasT4 v3-series		O	gn6i
A100	2020.05	8.0	6912	19.49	P4d	ND A100 v4-series		O	gn7e,gn7
A10	2021.04	8.6	9216	31.24		NVadsA10 v5-series			gn7i
A10g	2021.04	8.6	9216	31.52	G5				
L4	2023.03	8.9	7680	31.33				O	

[1]Compute Capacity

TABLE I: Specification of different GPU Instances on Cloud Services



(a) Normalized latency to train a model



(b) Normaized cost to train a model

Fig. 1: The relative latency and cost to train different models. The lower values are better.

epoch-detection module is 9.6% and that the average MAPE of the latency prediction of various GPUs is 11.7%, similar to the prediction accuracy of previous work that references the internal model architecture. The decent prediction accuracy of this proposed system without using the model source code or model profile output presents the applicability of this work in a public cloud to provide an optimal training environment without the help of users.

The major contributions of this paper are as follows:

- Epoch period detection using dynamically changing hardware metric
- Creating a predictive model using hardware metric information without the internal model architecture
- Implementation of automatic various hardware metric gathering module

II. Train Latency on Diverse GPUs on Cloud

Due to the nature of CNN training, which requires a significant amount of computing power in parallel, GPUs are widely used for training. The prohibitive cost of a cluster of GPU devices and the difficulties of maintaining it motivate algorithm developers to conduct training in the cloud, where GPU devices are provided with an on-demand billing model. Table I lists the GPU models currently offered by cloud providers, including AWS, GCP, AZURE, IBM, and Alibaba. The table encompasses GPUs from the earliest K80 to the latest L4 models. Notably, newer GPUs do not necessarily equip more GPU cores despite their elevated computing capacities and floating-point operations per second (FLOPS). Additionally, the value of FLOPS does not increase linearly with the recency of the GPU release date. With the expanding diversity of GPU instances and their features, merely assessing GPU specifications may be insufficient for users to gauge the most efficient option. Furthermore, different cloud vendors have diverse GPU models, and a specific model may not be offered by a user's primary cloud service provider, which could urge a user to set up a multi-cloud environment where service operation can be very challenging [8], [9].

Figure 1 compares training time and cost to demonstrate the variability in performance when training a model using various GPUs. In the experiments, we train multiple layers of the CNN model [1] with the CIFAR10 dataset (which we call *CIFAR10_CNN*) and VGG16 [10] using various GPUs provided by AWS, Alibaba, and GCP, presented on the horizontal axis. Figure 1a compares the latency to train an epoch. The vertical axis displays the relative latency normalized to the best-performing instance type in each model. The best-performing GPU type of CIFAR10_CNN and VGG16 is marked with a star and a triangle symbol, respectively. Different GPUs perform best for distinct models, and the relative latency is different for the model and GPU combinations without any pattern.

In the cloud, cost engineering is a critical process, and

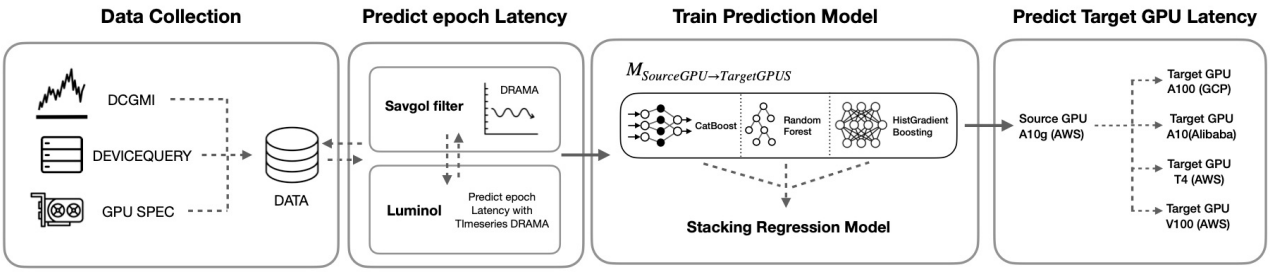


Fig. 2: The overall architecture of proposed training time prediction system relying on hardware metrics

Figure 1b compares the cost of training different models. Similarly to the latency distribution, the two CNN models had distinct cost-effective GPUs. The most cost-effective and fastest training GPUs are different, and the latency and cost-changing patterns are not intuitive if we consider only the hardware specifications summarized in Table I. Such distinct patterns can impose a significant burden on algorithm developers when building an optimal environment using cloud GPUs.

Suppose that one can accurately estimate the training time in diverse environments using distinct GPUs. In that case, it becomes relatively more straightforward to build an optimal training environment, because the cost to maintain a cloud GPU cluster can be calculated by the posted instance price. In the literature, to help build an optimal training environment, Paleo [3] employed an algorithm to predict the training time considering the model architecture, the size of the data set, the FLOPS of the GPU, and the bandwidth. Similarly, MLPredict [4] and Habitat [6] predict training times for various DNN models using model architecture and GPU hardware data, including detailed model information, such as the kernel size and input padding. Srifty [11] proposed a performance prediction model for multi-GPU distributed cloud systems using model and profile data.

Although the mentioned work accurately predicted the training time on GPU devices, all algorithms reference the internal model architecture to build a latency prediction model. These approaches are applicable if an algorithm developer builds and maintains the entire training environment. However, if a user builds an execution environment in the cloud, the circumstances change. In the initial cloud-service offering, represented as IaaS, users have flexibility in the environment setup. However, as the cloud computing service evolves, the abstraction layer of computing services increases, and building an optimal environment becomes the role of service providers, not the algorithm developers. In many cases, the model source code or internal architecture is confidential to an individual or an institution, and the developers are not willing to provide this information to cloud service providers merely to set up an efficient environment. Therefore, in the context of providing an optimal training environment in next-generation cloud services with a higher degree of service abstraction layer, represented as a function-as-a-service and serverless computing [12], it is critical to predict training latency without exposing any model details to cloud service providers.

III. Finding the Optimal Instance for Train

To help build an optimal training environment in the next-generation cloud, we propose a CNN training time prediction system without reference to the model architecture. Instead, the proposed system uses metrics collected from the hardware that cloud service providers can typically access, such as the cloud monitoring system, AWS CloudWatch, and Azure Cloud Monitor. Figure 2 presents the overall sequence of the proposed system. In the training hardware metric collection module (Section III-A), we introduce the hardware metrics for latency prediction and the mechanism for collecting them. The automatic training epoch-detection module (Section III-B) proposes an algorithm to detect the epoch based on hardware metric changes. The detected epoch becomes the prediction target in the training-latency prediction module (Section III-C).

A. Training Metrics Collected from Hardware

To collect effective metrics generated during the training phase to characterize the process, we used Data Center GPU Manager (DCGM [13]) and DeviceQuery [14]. DCGM, a software tool provided by Nvidia, enables real-time monitoring of various GPU metrics during training, including temperature, memory usage, power consumption, and error status. Although DCGM is used as a diagnostic tool that can identify and determine the cause of GPU problems, we used it primarily to track metrics to characterize a training task. We collect the DCGM logs collected every 0.1 second. In this paper, not all metrics collected from DCGM are utilized. Certain features are excluded, specifically those that do not show any variation over time, have identical values across all GPUs in the experiments, or cannot be collected from specific GPUs. DeviceQuery is one of the utility programs included in the Nvidia CUDA toolkit, which allows users to inquire about the primary features and capabilities of the installed Nvidia GPUs. We modified the Devicequery.cpp file within the Nvidia CUDA toolkit to save its output as text data for further processing. In Devicequery, not all metrics were utilized. Those values that were identical across all five GPUs, unsupported values, or values represented as binary were excluded.

1) Dynamic Dataset: This section summarizes the metrics collected to model training time. The following list presents the metrics whose values change over time during training.

- **SMACT**: The average ratio of active streaming multiprocessors (SMs) over a user-defined period (default 1 s).
- **SMOCC**: The proportion of warps actively working within an SM during a user-defined period. For example, if 7 out of a maximum of 10 warps are consistently active in a single SM, it yields a value of 0.7.
- **DRAMA**: The cycle ratio in which the device memory is actively sending or receiving data.
- **FP32A and FP16A**: FP32A denotes the ratio of cycles when the dfp32 pipe is active. A higher value indicates increased utilization of the FP32 core. In addition, FP16A is analogous to FP32A, differing only in the numerical designation.
- **PCIRX**: The total byte count of active PCIe receiving data, including the header and payload, including data copied from the host (CPU) to the device (GPU).
- **TENSO**: The proportion of cycles during which an SM has at least one warp assigned.
- **NVLTX**: The byte count of active NVLink transmit data, covering the header and payload.
- **GPUTL**: GPU utilization over a user-defined period.

2) Static Dataset: The mentioned dataset changes dynamically. Next, we summarize the static metrics used for latency modeling.

- **B1TTL**: Total base address register (BAR1) of the GPU in megabytes, mapping the frame buffer (FB) for direct access by the CPU or other devices.
- **B1FRE**: Amount of free BAR1 on the GPU in megabytes.
- **SMMAX**: Maximum supported SM clock speed for the device.
- **SHTMP**: Device shutdown temperature.
- **EPLMT**: Effective power limit enforced by the driver, accounting for all limiters.
- **PCILW**: Current PCIe link width.
- **TOTAL_GM**: Total global memory (in megabytes) or off-chip memory.
- **CUDA_CORES**: Computing power. However, identical numbers can still vary in efficiency.
- **CUDA_CORES/MP**: Cores per multiprocessor, indicating the core count of each multiprocessor.
- **GPU_Max_CL**: Maximum clock rate of the GPU in megahertz.
- **MEM_CL**: Memory clock rate.
- **MEM_BUSWIDTH**: Memory bus width in bits, indicating the number of memory input/output (I/O) lines.
- **MEM_SIZE**: Size of the DRAM.
- **TENSOR_CORES**: Specialized cores designed for mixed precision training.
- **L1/L2_CACHE_SIZE**: The L1 cache exists within each SM and shares space with shared memory, whereas the L2 cache is external to the SM.

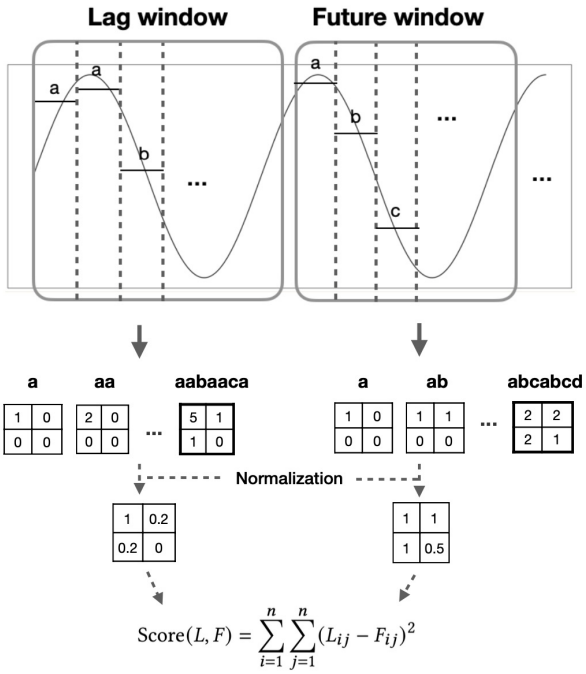


Fig. 3: Change point detection sequence using Luminol

- **TOTAL_SMPM**: Shared memory available per multiprocessor.
- **C_CK**: Concurrent copy and kernel execution copy engine.
- **FP16, FP32, FP64 TFLOPS**: Refers to 16-, 32-, and 64-bit floating point operations per second commonly used in computational research.
- **CC**: The computing capability of the GPU, defining the supported features.
- **MNG**: Maximum number of resident grids per device.
- **MNBWM**: Maximum resident blocks, warps, and memory per SM.
- **MNT**: Maximum threads per multiprocessor.
- **MNSM**: Maximum shared memory per thread block, measured in kilobytes.

B. Train Epoch Period Detection

The input features and target values should first be defined to build a training-latency prediction model. Considering the nature of the training, an epoch latency can be a good target value, but there is no method to detect an epoch period without referencing the source code. This section focuses on detecting the epoch period using a metric value.

1) Change Point Detection For Epoch Discovery: To capture the transition points where epochs begin and end, we leveraged the change point detection (CPD) algorithm. Among many metrics, although the boundaries marking the start and end of epochs may be visually observable in some models, there are numerous cases where this is not straightforward. Even when these transitions are visually apparent, generalizing the process to an algorithm can be challenging. In this paper,

we propose the use of Luminol [15] CPD algorithm to accurately identify these critical boundary points.

Luminol is a lightweight Python-based open-source library for time-series data analysis that offers two primary functionalities: anomaly detection and correlation analysis. This paper focuses on its anomaly detection capability. Luminol assigns a higher anomaly score to outliers in a time series and a lower score to regular patterns. When applied to time-series data, the Luminol approach generates a new graph that consists of only these scores.

Figure 3 presents the general operation of Luminol, illustrating the use of two windows for comparison in a time series, the lag and future windows. This method compares the similarity of these two windows using a sliding-window approach to derive anomaly scores. This computational process is referred to as the bitmap [16]. This approach involves partitioning time series data into segments and using the occurrence frequency of comparable segments to assess anomaly scores. In the bitmap detector operation of Luminol, the future window is referred to as the lead window. The continuous data within each window are converted into a string using the symbolic aggregate approximation method to facilitate comparison [17]. This method divides the range between the minimum and maximum values into categories, treating the values within each range as identical characters. In practice, this involves setting a specific interval in the window, calculating the average value of the data within that interval, and converting this average into a corresponding character based on its range. This process effectively discretizes continuous time-series data into a string sequence, such as $aabaaca$, simplifying the comparison. Next, using the chaos game [18] representation algorithm, these strings are converted into a $n \times n$ grid. In the figure, a 2×2 grid is used, where the frequency of characters in each cell is counted and filled. For a lag window string $aabaaca$, where a occurs five times and b once, the numbers are assigned accordingly. These grids are normalized, with the lowest value set at 0 and the highest at 1.

The anomaly score is determined by subtracting the corresponding values from the lag and future window grids, squaring and summing these differences. A higher score indicates closer proximity to an anomaly. This method allows us to reconstruct a new graph of anomalies from the time-series data using sliding windows. Applying Luminol, we aim to determine an epoch transition that can be regarded as an anomaly.

2) Smoothing Change Detection Score: Lumonol reconstructs raw metric data into anomaly detection scores. However, due to the high variability in the changed detection scores, using the score as an indication of the epoch period can generate too much noise, where values around an actual anomaly (a real epoch boundary) are also high and varying. To solve this problem, we use the Savitzky–Golay filter [19] to smooth the Luminol scores.

Figure 4 illustrates an example mechanism of the Savitzky–Golay filter to smooth discrete time-series data to a quadratic equation function. This algorithm collects data values within

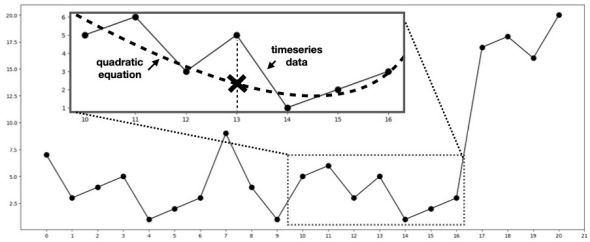
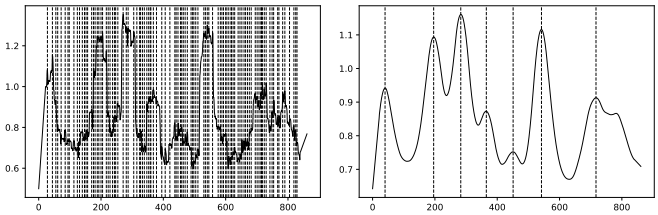


Fig. 4: Graph smoothing using Savgol Filter

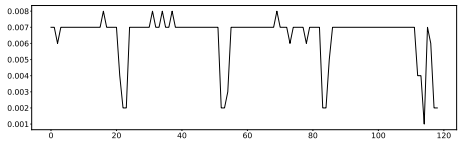


(a) Applying once (b) Applying three times

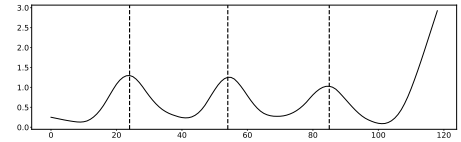
Fig. 5: Comparing the impact of applying Luminol and Savitzky-Golay multiple times

a specific range, creates a new regression function, and replaces the original values with those derived from this regression function to smooth the data effectively. First, we must generate a regression function using the data within the window. The least square principle, minimizing the difference between the smoothed regressor output and the real data, is used to determine the polynomial coefficients. For example, we consider data points within a window of $(-4, d_0), (-2, d_1), (0, d_2), (2, d_3), (4, d_4)$ and a second-order polynomial function $f(x) = a_0 + a_1 x + a_2 x^2$. We use the least square principle $[a_0 - 4a_1 + 16a_2 - d_0]^2 + [a_0 - 2a_1 + 4a_2 - d_1]^2 + [a_0 - d_2]^2 + [a_0 + 4a_1 + 16a_2 - d_3]^2 + [a_0 + 4a_1 + 16a_2 - d_4]^2$ to determine the coefficients to minimize the output, which is the regression error. This result is transformed into a matrix equation $A \cdot a = d$. By adding A^T to simplify the calculation, we solve for a in $(A^T \cdot A) \cdot a = A^T \cdot d$, determining the coefficients of the polynomial. The input value at each point is replaced by the polynomial value, smoothing the data.

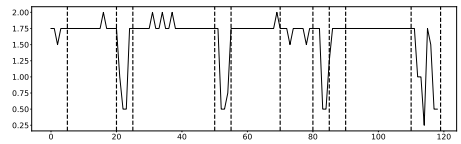
3) Train Epoch Period Detection: To detect a training epoch period using Luminol and the Savitzky–Golay filter, we mark the peak point in the processed data as a separate epoch. In the detection, we repeat a set of Luminol and Savitzky–Golay filters three times to improve the detection accuracy. Repetitively applying the combination is necessary because a single execution is not sufficient to smooth the data for peak value detection adequately. The rationale for combining Luminol with smoothing, rather than just repeating smoothing three times, is that continuous smoothing alone can obscure peak values, making them difficult to distinguish. Applying both techniques together in each set is crucial to preserve the distinctiveness of the peaks while smoothing. We detected spikes in the processed data using the *find_peaks* function from



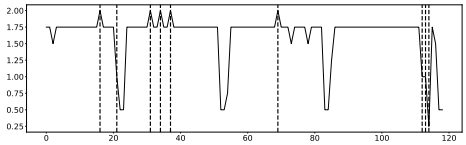
(a) DRAMA Dynamic Data



(b) Anomalies Detected with Luminol and Savgol Filtered Data



(c) Anomalies Detected Using Rupture



(d) Anomalies Detected Using jenkspy

Fig. 6: Visual comparison of the proposed train epoch detection algorithms with other well-known change point detection algorithms

SciPy's signal processing module [20].

Figure 5 compares the detection of the maximum value when applying Luminol and Savitzky–Golay filters just once (Figure 5a) versus three times (Figure 5b) to visually inspect the performance of the proposed heuristic. The figures demonstrate that applying the combination once does not sufficiently smooth the data and generates too many peaks with poor detection accuracy. We empirically verified that applying the combinations more than three times neither harms nor benefits the detection accuracy.

Figure 6 visually compares the results of the proposed epoch detection mechanism and other well-known CPD algorithms. Figure 6a presents the original DRAMA data collected during training. The results of applying three sets of the proposed Luminol and Savitzky–Golay filter combination are depicted in Figure 6b. Figure 6c illustrates the epoch detected using the Rupture CPD algorithm [21], and Figure 6d employs the Jenkspy CPD algorithm [22]. In each figure, the vertical dashed lines represent the detected epochs. The proposed method using the Luminol and Savitzky–Golay filters effectively reconstructs the graph and accurately identifies the major peaks, leading to accurate train epoch detection. The Rupture CPD algorithm tends to detect small peaks, but its main drawback is that it requires a definition of user-specified peak counts, which can be a significant limitation. The Jenkspy algorithm failed to detect the training epoch accurately. Jenkspy tends to detect outliers within a small range rather than significant changes in the DRAMA dataset. Consequently, it is highly effective in identifying minor variations within a limited scope. However, it is not well-suited for detecting major trends or significant changes within the overall flow of data. The proposed approach that uses a combination of Luminol and the Savitzky-Golay filter consistently presents good predictive accuracy with minimal deviation, and this comparative accuracy is discussed quantitatively in the evaluation section.

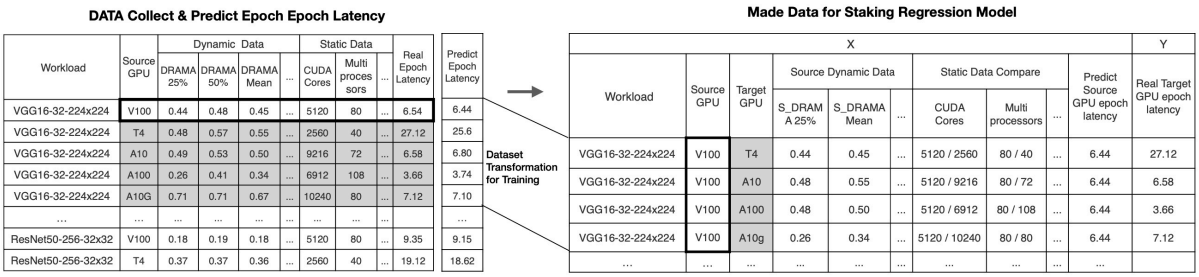
C. Train Epoch Latency Modeling

With the detected epoch using hardware training metrics, we built an epoch training-latency prediction model. This section describes how the input features and target values are constructed from hardware metrics.

1) Defining Input Features: To use the gathered training metrics as the model input features and target latency for prediction, they must be pre-processed. First, the diverse and dynamic data set whose value changes during the training phase must be aggregated to use the metrics as features. For aggregation, we extracted the distribution of each metric (mean, median, minimum, maximum, quartiles, and standard deviations) within the expected epoch. We did not aggregate the static data, using them raw. To use the static and aggregated dynamic dataset as input features to express the relationship between the source and target GPU instances, we used the ratio of the static dataset value between the source and target GPUs as input feature. In addition, the detected epoch latency is also added to the input feature. The target latency to predict is the model training time on a target GPU.

Figure 7 shows an example of building input features and target values. The aggregated dynamic dataset and the static raw dataset with the detected epoch latency are summarized in the left part of the figure. The aggregated dataset is transformed into the input of the prediction model on the right. In the input of the prediction model, the source and target GPUs are first defined. The aggregated dynamic metrics of the source GPU are used as the input feature $\mathbf{X}$. The hardware-related static dataset is added as the ratio value to express the relationship between the source and target instance types. For example, if the source GPU is V100, and the target GPU is T4, the CUDA cores feature is expressed as $\frac{5120}{2560}$, which is the fraction of the CUDA cores from V100 to T4. The detected epoch latency on the source GPU is added to the input feature. The epoch latency on the target GPU is defined as a target value, $\mathbf{Y}$.

2) Modeling Using Stacking Regressor: Using the proposed features, we built a training-latency prediction model by applying a stacking regressor heuristic [23]. Stacking is a prediction method that uses the predictions of various models as training data for a final model. The stacking model is divided into base regressors and a meta-regressor. In the stacking model, the predictions of each base regressor are used again as training



Workload	Source GPU	Dynamic Data				Static Data			Real Epoch Latency	Predict Epoch Latency
		DRAMA 25%	DRAMA 50%	DRAMA Mean	...	CUDA Cores	Multi processors	...		
VGG16-32-224x224	V100	0.44	0.48	0.45	...	5120	80	...	6.54	6.44
VGG16-32-224x224	T4	0.48	0.57	0.55	...	2560	40	...	27.12	25.6
VGG16-32-224x224	A10	0.49	0.53	0.50	...	9216	72	...	6.58	6.80
VGG16-32-224x224	A100	0.26	0.41	0.34	...	6912	108	...	3.66	3.74
VGG16-32-224x224	A10G	0.71	0.71	0.67	...	10240	80	...	7.12	7.10
...	...	...	...	...	...	...	...	...		...
ResNet50-256-32x32	V100	0.18	0.19	0.18	...	5120	80	...	9.35	9.15
ResNet50-256-32x32	T4	0.37	0.37	0.36	...	2560	40	...	19.12	18.62

X										Y
Workload	Source GPU	Target GPU	Source Dynamic Data			Static Data Compare			Predict Source GPU epoch latency	Real Target GPU epoch latency
			S_DRAMA 25%	S_DRAMA Mean	...	CUDA Cores	Multi processors	...		
VGG16-32-224x224	V100	T4	0.44	0.45	...	5120 / 2560	80 / 40	...	6.44	27.12
VGG16-32-224x224	V100	A10	0.48	0.55	...	5120 / 9216	80 / 72	...	6.44	6.58
VGG16-32-224x224	V100	A100	0.48	0.50	...	5120 / 6912	80 / 108	...	6.44	3.66
VGG16-32-224x224	V100	A10g	0.26	0.34	...	5120 / 10240	80 / 80	...	6.44	7.12
...	...	...	...	...	...	...	...	...	...	

Fig. 7: Reconstructing training data, including the previously obtained epoch latency, for the purpose of training model

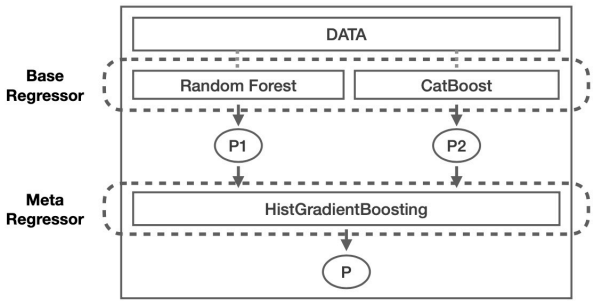


Fig. 8: The composition of training latency prediction model adopting the stacking regressors

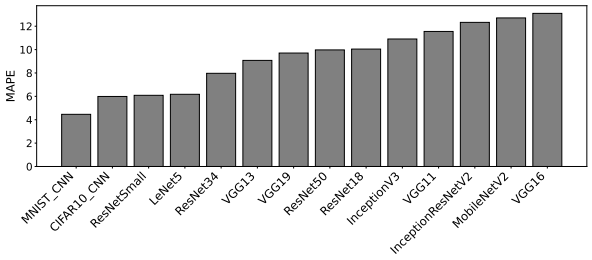


Fig. 9: Model-wise accuracy when comparing epoch latency using only hardware data

data. In this paper, the base models are the random forest [24] and categorical boosting (CatBoost) [25] methods. The random forest is an ensemble method that uses decision trees as its base model. It involves creating multiple decision trees and considering their results collectively to reach a conclusion. The second base model, CatBoost, is an efficient machine learning algorithm optimized for categorical data processing based on gradient-boosting decision tree ensembles.

The predictions from these base models are used as training data for the meta-regressor model. This paper applies the HistGradientBoosting regressor model [26] as a meta-regressor, which uses histogram-based gradient boosting, providing high speed and efficient memory usage in large datasets. Figure 8 illustrates the structure of the stacking regressor model in this paper.

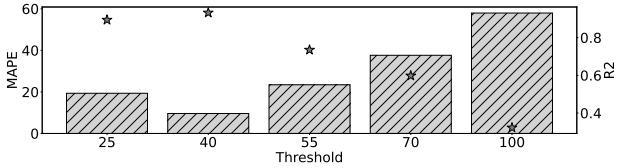


Fig. 10: Optimal criteria for smoothing based on dividing the entire data by a specific number in the savgol filter

GPU	T4	A10g	V100	A100	A10
Cloud	AWS	AWS	AWS	GCP	Alibaba
MAPE	11.66	7.59	10.59	9.51	8.66
R^2	0.93	0.91	0.96	0.93	0.93

TABLE II: Accuracy of predicting epoch latency based solely on hardware data, grouped by GPU

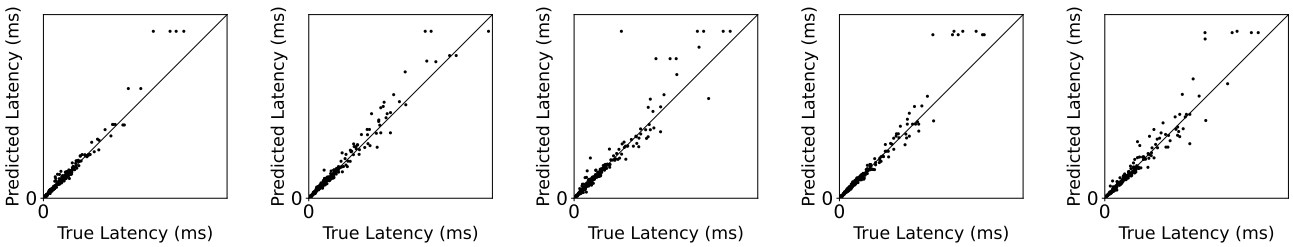
IV. Evaluation

This section, we evaluates the precision of the proposed system with overhead to maintain the system.

A. Experiment Setup

This study uses five GPU instances for evaluation. From AWS, we used three GPUs: the A10g GPU from the g5 instance, the T4 GPU from the G4dn instance, and the V100 GPU from the P3 instance. Additionally, we employed the A100 GPU from GCP's A2 instance and the A10 GPU from Alibaba Cloud's gn7i instance to gather data across a broad spectrum of cloud platforms. During the training phase, we employed the AWS Deep Learning AMI environment, specifically tailored for deep learning. A consistent environment was maintained across GCP and Alibaba Cloud for uniformity.

For the experiments, we applied a variety of CNN models, including VGG19, VGG16, VGG13, VGG11, MobileNetV2, InceptionV3, InceptionResNetV2, ResNet50, ResNet34, ResNet18, ResNetSmall, LeNet5, CIFAR10-CNN, and MNIST-CNN. The objective is to employ models of diverse sizes, from relatively smaller models, such as LeNet5 and MobileNetV2, to their larger counterparts, such as VGG19. The batch sizes implemented in the experiments were 16, 32, 64, 128, and 256, while training was carried out with input pixel sizes of 32×32, 64×64, 128×128, 224×224, and



(a) T4 (b) A10g (c) V100 (d) A10 (e) A100

Fig. 11: True (x-axis) and predicted (y-axis) latencies for different source GPU devices that are shown in each sub-figure

256×256. We trained each model by iterating 10 epochs for each. In addition, the automated hardware model collection system is designed to allow users to collect hardware data during training simultaneously. We applied the MAPE, the root mean squared error (RMSE) and R^2 as performance evaluation metrics. The R^2 value measures how well a predicted value follows the actual data. The value ranges from 0 to 1, where a higher value indicates better regressor quality.

B. Epoch Detection Accuracy

This section validates the accuracy of the previously discussed epoch-detection heuristic. Using the DRAMA metric collected through DCGM provided the best detection accuracy, and we used the metric as the basis for the proposed system. Table II presents the MAPE and R^2 of the epoch-detection heuristic measured for various GPU devices. The MAPE of the epoch-detection algorithm measures how much the predicted epoch period differs from the actual epoch period. Regardless of the GPU devices, the model demonstrates good prediction accuracy. The average MAPE is 9.606. The R^2 also indicates accurate epoch-detection quality.

Figure 9 displays the precision of epoch detection using the categorized MAPE metric for the models displayed on the horizontal axis. The MNIST_CNN achieved the best accuracy at 4.46, whereas VGG16 displayed the lowest accuracy at 13.1. As the size of the model increased, so did the MAPE, but no particular model exhibited exceptionally poor results, suggesting a consistent level of good accuracy across the models. This result underscores the feasibility of detecting the epoch boundary and latency solely on the basis of the DRAMA dynamic metric.

Figure 10 presents the precision based on the number of data points when smoothing is applied using the Savitzky–Golay filter. The threshold signifies the total number of dynamic data points divided by the threshold value, which is the basis for smoothing. The bar graph represents the MAPE, whose value is indicated on the primary vertical axis. The star markers depict the R^2 metric, whose value is presented in the secondary vertical axis. The horizontal axis displays the threshold. When smoothing was performed with 40 data points, as we adopted in the proposed system, MAPE and R^2 exhibited the highest precision, with a subsequent decrease in precision as the threshold changed.

CPD Algorithm	Rupture	Jenkspy
MAPE	163.38	172.86

TABLE III: Comparing the epoch detection accuracy when using well-known CPD algorithms

Source GPU	T4	A10g	V100	A100	A10
Cloud	AWS	AWS	AWS	GCP	Alibaba
MAPE	11.67	15.82	13.77	10.50	12.89
R^2	0.83	0.83	0.84	0.85	0.84

TABLE IV: Final predicted values for 1 epoch latency across different GPUs using the predictive model

This paper proposes the combination of the Luminol CPD algorithm and a Savitzky–Golay filter. To compare the performance of the proposed algorithm, we compared MAPE with other well-known CPD algorithms (Jenkspy and Rupture) in Table III. Rupture and Jenkspy demonstrated an error rate exceeding 100%, which appears to be due to significant deviations for the DRAMA dataset. Although the presented value is based on the DRAMA result for a fair comparison, the model exhibited a similar error rate for other dynamic datasets. This comparison allows us to assert the superiority of the detection accuracy of the proposed system.

C. Target GPU Latency Prediction Accuracy

This section evaluates the accuracy of the target GPU epoch latency prediction. Figure 11 visually presents using a scatterplot, with the horizontal axis indicating the actual training latency and the vertical axis marking the predicted latency. Points near the $y = x$ line suggest accurate predictions. Each subfigure indicates the source GPU type, and each scattered point is the prediction of other target GPUs. The predictions align closely with the actual CNN training times, implying robust accuracy.

Table IV reveals that the A100 GPU has a leading accuracy of 10.50, whereas the A10g GPU registers a less optimal value of 15.82. However, most instances maintain MAPE values of around 15, with all R^2 values exceeding 0.8.

Table V explains the rationale for using dynamic and static data for training. Training solely with dynamic data resulted in a MAPE of 71.61, while using only static hardware data

	Dynamic Data	Static Data	ALL Data
MAPE	71.61	20.1	11.7

TABLE V: Comparison of GPU latency prediction accuracy using dynamic, static, and combined datasets.

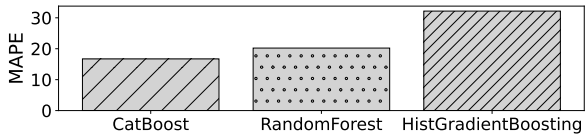


Fig. 12: Comparison of performance improvement when using a stacking regressor model over single models

yielded a MAPE of 20.01. Combining dynamic and static datasets improved the accuracy to 11.7, stating that using various datasets is a valid decision.

Figure 12 presents the effect of choosing a stacking regression model as predictor, where the MAPE is on the vertical axis. On the horizontal axis, a separate model is depicted. As mentioned, this ensemble model uses predictions from multiple base models in the horizontal axis to form a meta-model, outperforming single base models. The proposed stacking regression achieved a MAPE of 11.17, marking improvements of 16.7% over CatBoost, 20.2% over random forest, and 32% over HistGradientBoosting.

There may be a difference in latency when training is performed with and without the metric collection module. Figure 13 illustrates the additional latency due to the metric collection module to observe the overhead of metric measurement. In the figure, the horizontal axis indicates the model and the vertical axis indicates the percentage of the performance penalty. No specific model showed a significant disparity, with an average overhead of 5.5%. This outcome does not directly affect prediction performance; thus, overhead was not a primary concern in this study.

D. Comparison with Other Prediction Approaches

This study aims to compare the results with previous research on the prediction of training latency detection, specifically Paleo [3]. Paleo predicts training time considering factors such as model architecture, training data size, GPU FLOPS, and bandwidth. However, instead of relying solely on hardware devices, it incorporates hardware information into the model architecture and training data. Table VI presents a comparison using the VGG16 model, used in both this study and Paleo. Although the MAPE increased slightly from 10 to 15.14, considering that the prediction is based solely on hardware data without any model information, the discrepancy is not considered significant. In terms of RMSE, this study exhibits superior results with a value of 4.14, establishing its advantage in this regard. These findings demonstrate that even without disclosing the profiling results or the specific structure of the model, the proposed approach can produce results comparable to those of the leading studies.

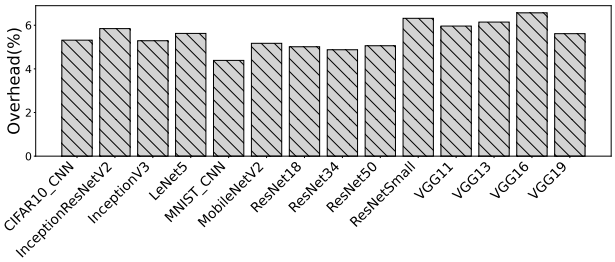


Fig. 13: Overhead occurring during data collection, shown by model

V. Related works

Training-Latency Prediction: Paleo [3] attempted to predict training time considering the model architecture, data size, GPU FLOPS, bandwidth, and other factors. However, there is concern that the security of the learning model would be compromised due to the inclusion of the model architecture if it were applied in a public cloud setup. The MLPredict [4] model also predicts the training time of various DNN models, including a variety of model architectures and GPU hardware information. However, the contribution of hardware data among all features is small, and the prediction model references specific model information, such as kernel size and input padding. NeuralPower [7] uses polynomial regression, featuring an internal model architecture to predict DNN training times for hardware environments and power usage limited by resources. Ruohan Wu et al. [27] propose a two-stage performance modeling framework that combines graph-level analysis and operator-based hotspot modeling to predict the execution time of DNN models. CDMPP [28] is a framework for predicting operator latency for training and inference in various DNN models and devices. All of the aforementioned methods used model architecture data, and model information cannot be protected. The algorithm proposed in this paper has the advantage of not having to reveal the model architecture or training details because it only uses hardware metric data.

Habitat [6] and PROFET [5] predict training time using features collected by profiling. DNNAbacus [29] analyzes the computational resource demands of classical DNN models and introduces a lightweight DNN performance prediction model. The study employed a profiling approach to generate training data by utilizing model information. DNNAbacus accurately predicts train time and memory costs for PyTorch and TensorFlow models showcasing its efficacy across diverse architectures. Black-box models, in comparison to white-box models, do not reveal detailed insights into the system's internal operations. Our approach, which excludes all data related to the model and training, demonstrates sufficient accuracy even in situations requiring rigorous security. Thus, it has the advantage of being applicable in public cloud environment setups.

Runtime Estimation on the Cloud with Cost-Time Optimization: Many studies have focused on performance

	PALEO	Our Method
MAPE	10.1072	15.14
RMSE	32.3637	4.14

TABLE VI: The prediction accuracy compare with PALEO

estimation in cloud environments. CherryPick [30] is a tool that employs Bayesian optimization algorithms to efficiently determine the optimal cloud configuration space for big data analyses. Paris [31] is a data-driven system that provides accurate performance estimates with minimal data collection, using a unique combination of off-line and online data collection and modeling to address the problem of optimal virtual machine selection. MPEC [32] is a system that can predict latency when performing distributed matrix multiplication tasks of various input sizes and shapes with different instance types and the number of worker nodes in a cloud computing environment. The mentioned work has a commonality with this work in the context of recommending an optimal environment in the cloud. However, the target domain and devices become diverse with recent advances in artificial intelligence applications, and previous work primarily focused on CPU devices. As deep learning becomes mainstream in the field, the setup of efficient GPU environments becomes a crucial problem.

Srifty [11] proposed a performance prediction model in a distributed cloud system composed of multiple GPUs using model information and profiling model data. Oikonomos [33] addresses the challenge of acquiring instances for deploying High-Performance Computing (HPC) applications in the cloud. It calculates the cost per instance and suggests the most suitable instance based on the user requirements. Accordia [34] is a system developed to handle the diverse nature of big data analysis jobs, identifying cost-optimal configurations in the context of variable cloud service costs. It leverages Gaussian Process Upper Confidence Bound (UCB) techniques to offer theoretical performance guarantees. These studies assist users in identifying the most optimal environment in the cloud, including GPU instances. However, due to the use of instance-related data and model data, these cannot be fully classified as black-box models.

VI. LIMITATIONS AND FUTURE WORK

While the prediction accuracy of the proposed system has not yet exceeded previous research that references model information, future studies should focus on improving prediction accuracy based solely on hardware capabilities. In the future, we anticipate employing a wider variety of algorithms such as Neural Architecture Search (NAS) [35] to improve the accuracy of the prediction model beyond its current state. The proposed work mainly relies on hardware metrics to avoid referencing internal model architecture for latency modeling, other features need to be discovered to enhance prediction model accuracy.

It is also essential to explore predictions for diverse hardware beyond GPUs. In the experiments, we diversified the GPU devices across different cloud vendors and could verify the generality of the proposed algorithm. Assuming that newly released GPUs keep providing similar hardware metrics, it is expected to provide decent prediction accuracy. For special-purpose hardware other than GPUs, such as Tensor Process Units (TPU) [36] by Google or Tranium by AWS, the hardware metrics and libraries provided are different from those provided by GPUs. The overall procedure of this proposed work composed of the epoch period detection and the training time model can be applied to new hardware. However, the specific metrics to characterize DNN training jobs need to be further analyzed, which is remained for future work.

The current work supports the prediction of CNN models only, but it is also important to exapnd the prediction to other DNN model categories, such as RNN and Transformer models. With the recent trend of increasing model and dataset sizes, it would be beneficial to expand predictions to multi-GPU environments, not just single GPUs. Furthermore, creating a prediction model that includes not just training, but also inference, can provide an even more comprehensive optimal cloud environment for DNN.

VII. CONCLUSIONS

This paper proposes a system to predict training time on cloud GPU instances without relying on model-specific details, enabling cloud providers to offer time-efficient and cost-effective training environments without disclosing model internal architecture to users. To achieve the goal, the proposed work leverages metrics generated from training hardware, which does not need to reveal the internal architecture of DNN models. Using hardware metrics as features, the proposed system predicts the training latency of multiple DNN models when they are conducted on various GPU devices. As the proposed work necessitates minimal model-related information, it can become a basis work to provide an optimal environment for security-conscious users, especially in a public cloud environment. To accomplish this, we developed an epoch-detection algorithm by observing active device memory usage, and using the detected epoch, we proposed a stacking regressor for training latency prediction. The accuracy of the prediction model is quantified at MAPE of 15.14%, which does not diverge significantly from the prediction accuracy of recent studies, including Paleo [3] and Habitat [6], which refer to the internal architecture of DNN models. Given that most recent similar work utilizes an internal model architecture to predict training latency, the ability to produce comparable accuracy without referencing internal model detail and using metrics from hardware data exclusively is a major contribution of this proposed work, and we believe it will provide a new opportunity to offer a fully-managed cost and time optimal DNN training environment on public cloud services.

VIII. ACKNOWLEDGMENTS

This work is supported by the National Research Foundation (NRF) Grant funded by the Korean Government (NRF-2020R1A2C1102544), AWS Cloud Credits for Research program, and the SW Star Lab (RS-2022-00144309) of IITP.

References

[1] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference 2014*, 2014.

[2] Joseph M. Hellerstein, Jose M. Faleiro, Joseph Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. Serverless computing: One step forward, two steps back. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org, 2019.

[3] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In *Proceedings of the International Conference on Learning Representations*, 2017.

[4] D. Justus, J. Brennan, S. Bonner, and A. S. McGough. Predicting the computational cost of deep learning models, 2018.

[5] Sungjae Lee, Yoonseo Hur, Subin Park, and Kyungyong Lee. Profet: Profiling-based cnn training latency prophet for gpu cloud instances. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 186–193, 2022.

[6] Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A Runtime-Based computational performance predictor for deep neural network training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 503–521. USENIX Association, July 2021.

[7] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. *Asian Conference on Machine Learning*, 2017.

[8] Dana Petcu. Multi-cloud: Expectations and current approaches. In *Proceedings of the 2013 International Workshop on Multi-Cloud Applications and Federated Clouds*, MultiCloud '13, page 1–6, New York, NY, USA, 2013. Association for Computing Machinery.

[9] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, and Ion Stoica. SkyPilot: An intercloud broker for sky computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 437–455, Boston, MA, April 2023. USENIX Association.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[11] Liang Luo, Peter West, Arvind Krishnamurthy, and Luis Ceze. Srifty: Swift and thrifty distributed training on the cloud, 2022.

[12] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Jayant Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. Cloud programming simplified: A berkeley view on serverless computing. *CoRR*, abs/1902.03383, 2019.

[13] NVIDIA. Create an op. https://www.tensorflow.org/guide/create_op, 2023.

[14] NVIDIA. devicequery - device query. https://github.com/NVIDIA/cuda-samples/tree/master/Samples/1_Utilities/deviceQuery, 2022.

[15] LinkedIn. luminol. https://github.com/linkedin/luminol, 2015.

[16] Nitin Kumar, Venkata Lolla, Eamonn Keogh, Stefano Lonardi, and Chotirat Ratanamahatana. Time-series bitmaps: a practical visualization tool for working with large time series databases. 04 2005.

[17] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, DMKD '03, page 2–11, New York, NY, USA, 2003. Association for Computing Machinery.

[18] Copyright. In MICHAEL F. BARNSLEY, editor, *Fractals Everywhere (Second Edition)*, page IV. Academic Press, second edition edition, 1993.

[19] Abraham Savitzky and Marcel J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36:1627–1639, 1964.

[20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[21] Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, 2020.

[22] George F. Jenks. The data model concept in statistical mapping. 1967.

[23] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[24] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[25] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features, 2019.

[26] Deep dive into scikit-learn's histgradientboosting classifier and regressor. histgradientboosting, 2019.

[27] Ruohan Wu, Mingfan Li, Hanxi Li, Tianxiang Chen, Xinghui Tian, Xiaoxin Xu, Bin Zhou, Junshi Chen, and Hong An. Machine learning-enabled performance model for dnn applications and ai accelerator. *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 25–34, 2022.

[28] Hanpeng Hu, Junwei Su, Juntao Zhao, Yanghua Peng, Yibo Zhu, Haibin Lin, and Chuan Wu. Cdmpp: A device-model agnostic framework for latency prediction of tensor programs. *ArXiv*, abs/2311.09690, 2023.

[29] Lu Bai, Weixing Ji, Qinyuan Li, Xi Yao, Wei Xin, and Wanyi Zhu. Dnnabacus: Toward accurate computational cost prediction for deep neural networks. *ArXiv*, abs/2205.12095, 2022.

[30] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 469–482, Boston, MA, 2017. USENIX Association.

[31] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith, and Randy H. Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 452–465, New York, NY, USA, 2017. ACM.

[32] M. Son and K. Lee. Distributed matrix multiplication performance estimator for machine learning jobs in cloud computing. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, volume 00, pages 638–645, Jul 2018.

[33] Jan Harm Betting, Dimitrios Liakopoulos, Max Engelen, and Christos Strydis. Oikonomos: An opportunistic, deep-learning, resource-recommendation system for cloud hpc. In *Proceedings - 2023 IEEE 34th International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2023*, Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors, pages 188–196, United States, 2023. Institute of Electrical and Electronics Engineers (IEEE). 34th IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2023 ; Conference date: 19-07-2023 Through 21-07-2023.

[34] Y. Liu, H. Xu, and W. Lau. Cloud configuration optimization for recurring batch-processing applications. *IEEE Transactions on Parallel amp; Distributed Systems*, 34(05):1495–1507, may 2023.

[35] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

[36] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, et al. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, June 2017.